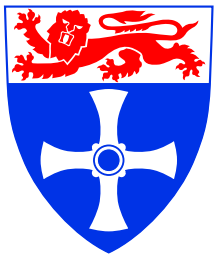
UNIVERSITY OF
NEWCASTLE UPON TYNE



Proceedings of the First UK Embedded Forum

A. Koelmans, A. Bystrov, M. Pont (Editors)

Foreword

This volume contains a selection of papers from the first UK Embedded Forum, a satellite event to the Embedded Systems Show (a yearly exhibition of embedded software and hardware products, organised by EDA Exhibitions Ltd). The aim of the Embedded Forum is to bring academic researchers in this economically important discipline together. The Forum aims to give PhD students from the UK the chance to present their work to their peers, and receive valuable feedback in return. A total of sixteen papers were presented, of which fifteen are included in this volume. Their topics span an impressive range of research activities, from System-on-Chip design issues to software scheduling techniques.

The event took place on October 13th and 14th at the National Exhibition Centre, Birmingham. We are deeply indebted to EDA Exhibitions Ltd, especially Jeremy Kenyon and Andrew Porter, for the local arrangements and general support.

The event was a success, and we hope there will be many more to come.

Newcastle, Leicester
December 2004
The Editors

Published 2004, University of Newcastle upon Tyne.
ISBN 0-7017-0180-3.

Contents

ViPERS - A Virtual Prototyping Methodology for hand-held electronic devices

P.F. Lister, V. Trignano, M.C. Bassett, P.L. Watten, B.J.C. Jackson.
Centre of VLSI and Computer Graphics,
University of Sussex,
Brighton, BN1 9QH, UK
P.F.Lister@sussex.ac.uk

1 Abstract

This paper presents the ViPERS methodology [1],[2], for the design and development of electronic hand-held devices. The concepts, the implementation, and the experiments presented in this paper were developed at the University of Sussex (UoS) in the Centre of VLSI and Computer Graphics as part of an EU project [3].

The fast increase in silicon technology and the pressure of a shortened time-to-market (TTM) are increasing the need for new design methodologies and techniques for the design and development of System-On-Chips (SoC). The ViPERS methodology was developed to face the contemporary challenges of SoC by integrating key design methodologies with the graphical and interactive features of virtual prototyping.

This paper illustrates how the ViPERS methodology and tools are applied to the design of an industrial RF remote control device used to control some functions of a cooking stack. The outlined design flow includes the modelling of both the digital and graphical features of the remote control. A photorealistic two-dimensional ViPERS simulation environment will be illustrated and a novel three-dimensional simulation feature for enhanced user experience will be introduced.

2 Introduction

Embedded systems have become part of our every day life. The complexity of these products is increasing very rapidly due to the constant developments in silicon technology. In conjunction to the growth in complexity, the time-to-market is becoming shorter than ever, posing an even greater pressure on manufacturing companies. The *International Technology Roadmap for Semiconductors (ITRS)* in 2003 [4] predicted that the number of transistor per chip will keep on growing at a high pace. With the current pressure on manufacturing companies and the current predictions for silicon technology, it is clear that new design methodologies and techniques are needed for design teams to cope with the complexity of SoCs.

The ViPERS (**Vi**rtual **P**rototyping of **E**mbedded **R**eal-Time **S**ystems) methodology explores key design trends in SoC design, such as system level design, and links them to the modern graphical and interactive features offered by virtual prototyping. Virtual prototyping, in the context of the ViPERS methodology, is seen as a visual layer that lies on top of the digital design of electronic devices to help engineers better explore their designs.

This paper is based on the collaboration between the University of Sussex and Ikerlan [5], for the design of an RF remote control; the purpose of the remote control is to control some of the functions related to a cooking stack such as the temperature of the oven, the ventilation speed of the hood, etc. This experiment will highlight some of the benefits of applying the ViPERS methodology to the design of hand-held electronic systems; the visual layer will demonstrate the advantages in the exploration of requirements, behaviour, and interfaces.

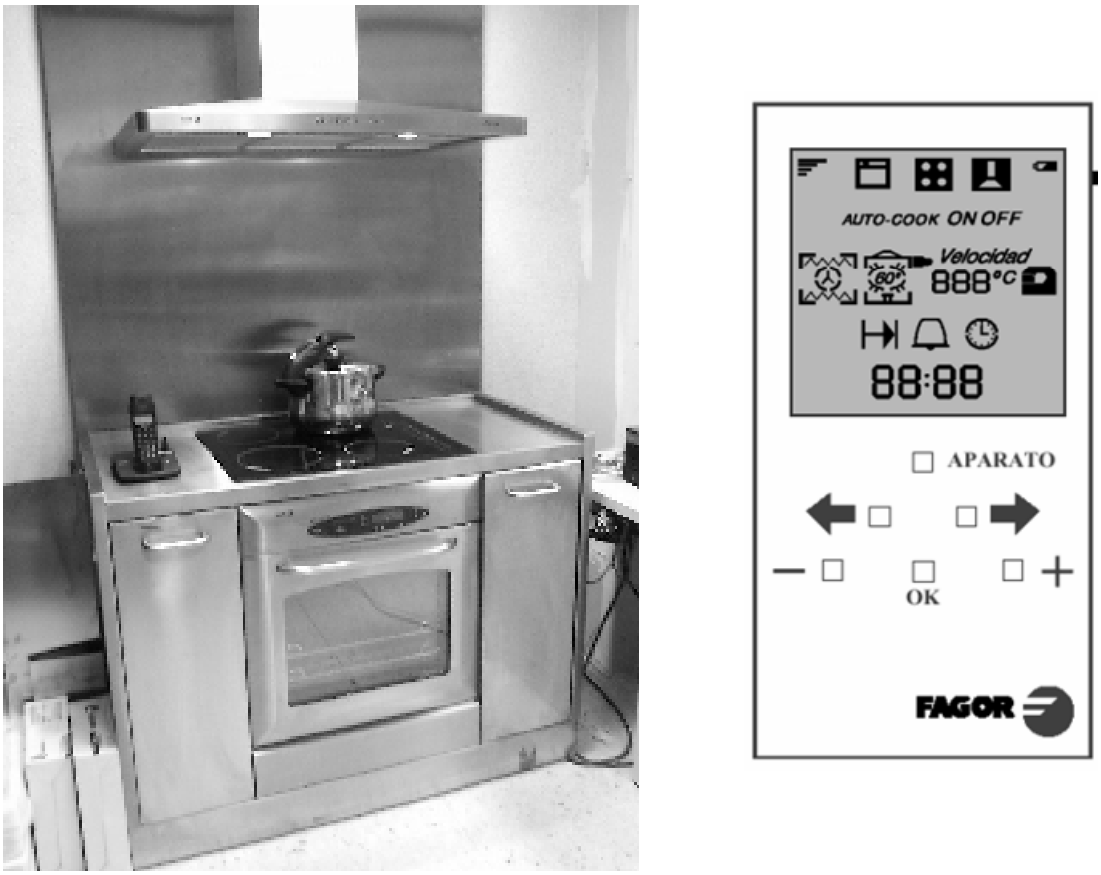


Figure 1 – photo of the cooking stack and sketch of the remote control

Figure 1, shows a picture of the cooking stack and a sketch of the associated remote control. This plus a written requirement document of the remote control was the information provided by Ikerlan at the beginning of the collaboration.

The work undertaken by the ViPERS team included a detailed requirement analysis and the implementation of a high level design representation of the remote control. UML [6] was chosen as the modelling language for the requirement analysis and its real-time profile, know as the UML Profile for Schedulability, Performance and Time [7], used to implement the executable model for requirements exploration. SystemC was the modelling language of choice for the digital design of the remote control. SystemC [8][9][10] is a C++ based modelling platform that supports different levels of abstraction, from system level, through behavioural to register transfer level. SystemC is geared to deal with the heterogeneity of today's systems; it consists of a class library and a simulation kernel which provide a series of advantages, the most important being the common environment for co-design, IP reuse, and test benches reuse across different levels of modelling abstraction.

High level models can be put together with minimal effort, allowing rapid design exploration in the early stages of the design phase. The initial description of the remote was later refined and further details added to the design. The ViPERS methodology can roughly be divided into three main modelling tasks where appearance and functionality of the electronic devices are modelled and then refined. Figure 2 shows the modelling tasks:

- **UML modelling**: includes the creation of the various UML diagrams plus the implementation of an executable model in UML that can be used in a simulation environment to test the interfaces of the device.
- **SystemC modelling**: this includes the implementation of an executable SystemC model and its related refinement processes.

- **Graphics modelling**: it includes the 2D/3D graphical modelling of the devices and environments.

Not all the tasks mentioned above are compulsory in the context of the ViPERS methodology but some are included as extra features that designers can exploit. The 3D modelling of the environment for example can be used to enhance user experience in a complex simulation environment but it is not necessary for the purposes of simulation. Tasks are shown in parallel since, at present, each phase is independent of others. A design flow will be explored in this paper, to illustrate the suggested flow for the ViPERS methodology.

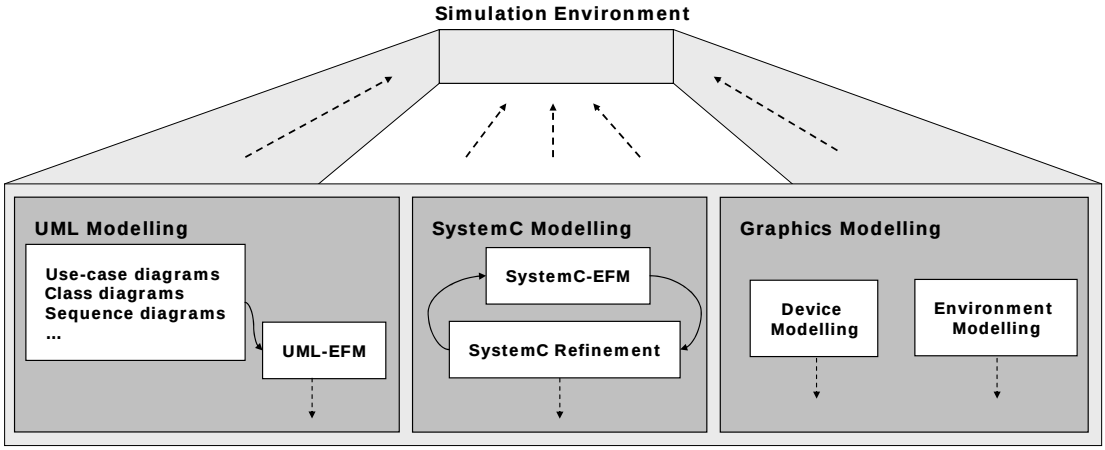


Figure 2 – Modelling tasks

3 UML Modelling

Determining accurate product requirements and specifications is a vital stage in the development of a commercially viable device. Virtual prototyping can help extend the value and meaning of the requirements and specification phase to further ensure the validity of this work before implementation begins [11]. A key benefit of virtual prototyping is the bringing together of the various stakeholders in a project, these might include all commercial interests, such as marketing, engineering and senior management.

Existing tooling for SoC design tends to assume that an accurate specification of the device behaviour has been developed as an isolated activity at the start of the design process. It is clear from the research in the field of requirements and specification development [12] that the specification work is unlikely to be confined to the period before implementation begins. Hence there is a need to rapidly feed changes in the requirements into the implementation tool chain in an evolutionary way.

The ViPERS approach can be used to reduce the imagination gap between the appearance and style of the virtual prototype and the final product. If the intention of the virtual prototype is to ensure that all stakeholders have a common frame of reference for the final product then any lack of detail on the graphical quality could have the result that the final product does not satisfy the aesthetic requirements of diverse stakeholders.

It is common for a design house to be given a written specification for a prototype device. Often the specification is not complete enough for the first resultant prototype to be satisfactory to the client, resulting in some design iterations. If the design house were to build a virtual prototype or even several alternative schemes, the client can clarify the specification before any hardware or software is built, the virtual prototype is a form of communication and reference in addition to the written specification [13].

The first step in the design flow adopted in the ViPERS methodology is represented by a UML exploration of the requirements. Initially UML diagrams are created to describe the device; documentation relative to the diagrams can be auto generated. For the requirements

analysis and the development of the UML diagrams, such as use-case, sequence, class, etc.. we made use of Rational Rose RealTime; the choice was determined by the possibility of reusing the UML diagrams to support the executable model.

Once a first UML description of the remote control was achieved, dynamic features were added in order to create a UML executable functional model (UML-EFM) [14] with the aid of the real-time features [15] offered by UML. For the purpose of this paper we define the Executable Functional Models as models that describe a number of properties associated with the functionality of an electronic system; they are stand-alone executables which contain the means to communicate with an external application for simulation purposes. The UML-EFM was implemented using Rational Rose RealTime. The functionality of the remote control was represented by a state machine. The interaction of the UML-EFM model with its graphical counterpart constitutes the simulation environment where the virtual prototype is used to explore requirements issues related with the functionality and interfaces of the remote control. Interaction of the state machine of the remote control with the graphical model is achieved by the use of an external capsule (Socket Capsule) which acts as a server application between the EFM and the graphical model. The Socket Capsule is part of a complete UML package (SxUMLSocket Package [14]), show in Figure 3, developed at the University of Sussex to enable rapid connection of a state machine with external applications. Communication with the graphical models has to comply with a XML-like communication protocol.

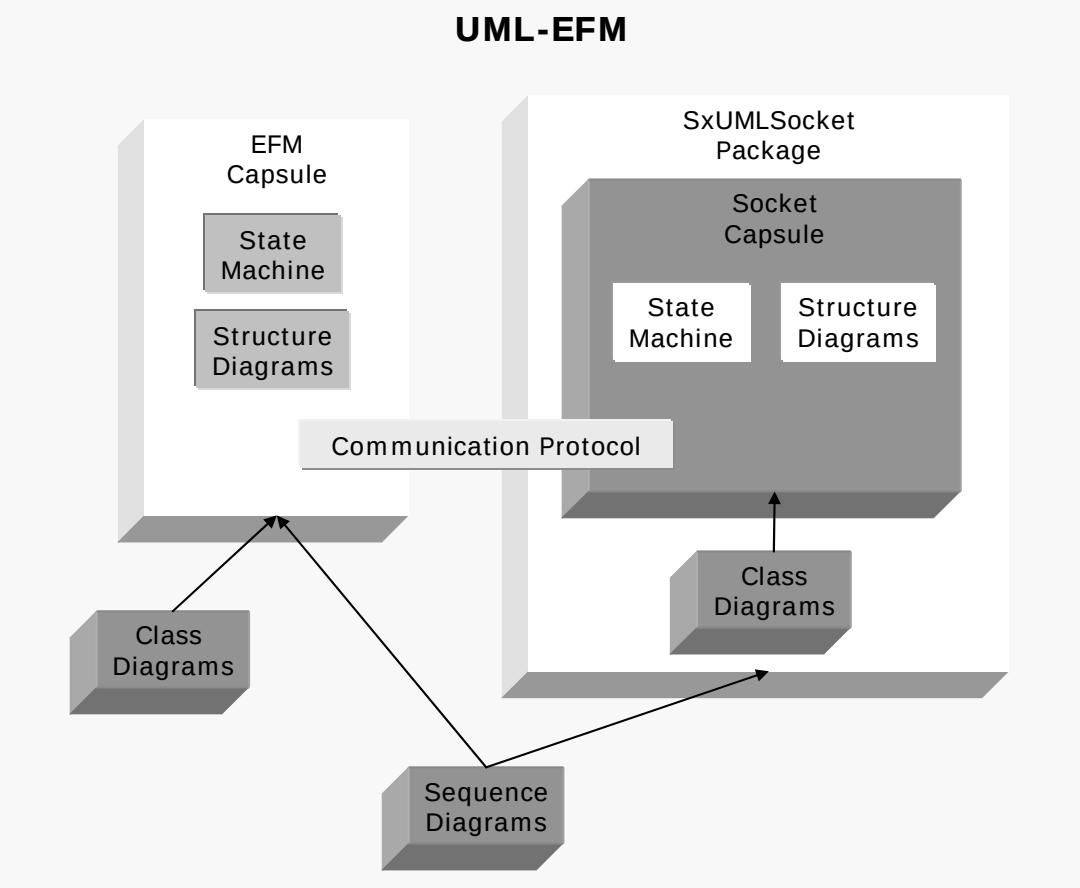


Figure 3 - UML-EFM structure

At this stage the focus is on the interfaces of the devices. Users with very different background can “use” the virtual prototype of the remote control and provide useful feedback. The virtual prototype can easily be modified and different interfaces easily tested; changes in the requirement documents are automatically generated.

4 Graphics Modelling

Earlier in this paper virtual prototyping in the context of the ViPERS methodology was described as a visual layer on top of the digital design. The modelling of the visual layer includes the modelling of all the graphical elements plus their interactivity features.

In this section two types of visual modelling will be explored, device and environment modelling. Environment modelling [16] represents a novel and optional phase in the ViPERS methodology that involves the creation of a virtual environment for the target devices. In the specific case of the remote control the environment is represented by a three dimensional kitchen which contains the cooking stack as shown in Figure 5 .

4.1 Device Modelling

Device modelling is the process of creating two dimensional photorealistic models of the target device, and/or other components which could take part in the simulation environment. The modelling of the graphics in the ViPERS toolset is achieved through the use of the IDE VDM [1] (**V**irtual **D**evice **M**odeller).

The photorealistic virtual prototype model is pulled together from graphics files within the VDM environment. The VDM environment has been created to satisfy the ViPERS requirement to focus some energy on a photorealistic appearance and on the accurate modelling of the device behaviour from a user-interaction perspective. It is common to find reasonable quality modelling of buttons and other user input methods on existing virtual prototype tools [17], but an area that is neglected is the accurate modelling of displays. One of the features required of a virtual prototyping environment to meet the ViPERS requirements is a real-time alpha-blending capability. The ViPERS VDM environment offers visually accurate modelling of both input devices and custom displays to increase the level of realism using technologies such as alpha-blending [18]. These button and display element files are created in a bitmap graphics editing application such as Adobe Photoshop. The development offers a logical split of design style between the scripted graphical appearance coding and the device behaviour that will eventually be implemented in software or hardware.

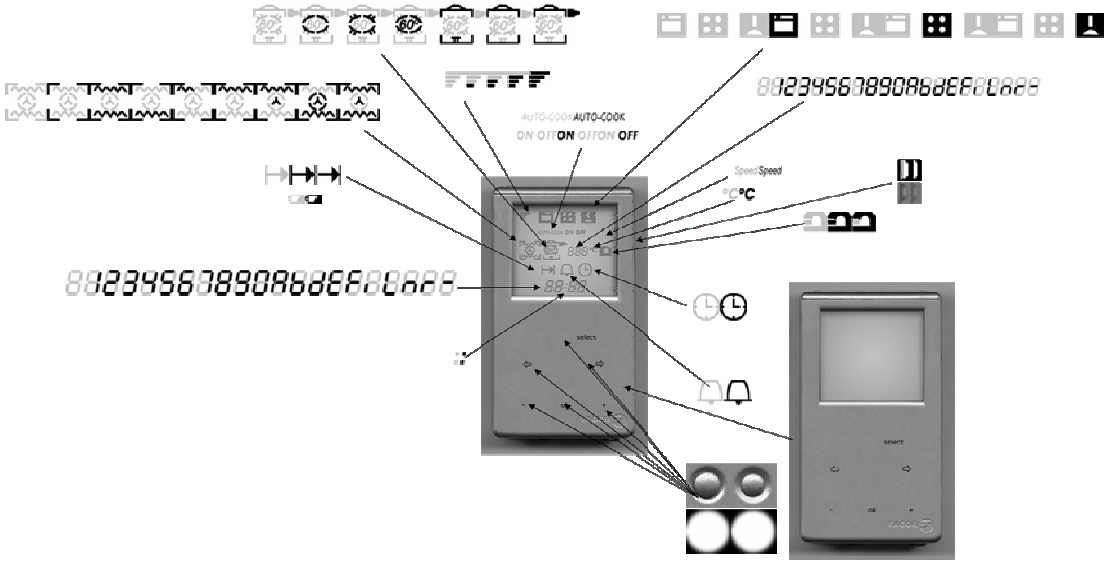


Figure 4 - Remote Control

The ViPERS VDM development environment allows the importing of standard graphics files that become the buttons and display elements. The environment uses an XML like file to store the description of buttons or display elements as well as the top level virtual prototype. The user is given a head-start by the library of existing high-quality buttons and display elements. The user can also create new interaction elements and save the graphics and interaction scripts in one or more libraries.

Figure 4 shows the combination of the graphical layers that were blended together to create the interactive graphical model of the remote control.

4.2 Environment Modelling

In the methodology that we propose the modelling of the environment is an optional phase that can be undertaken to add value and reality to the simulation experience. The value of a virtual environment is not extensive in a remote control for a cooking top. More relevance can be found in environments were several devices are working and collaborating; a rich multimedia environment. In such cases virtual environments can be used to test devices and experience changes in the environment especially in the presence of wireless devices where constraints have been imposed upon them in the environment. The virtual environment can be created with off-the-shelf software tools as 3D StudioMax. The interactive components can be implemented as 3D models or alternatively as 2D models and then superimposed in the virtual environment. The means by which the 3D model and the environment communicate is the same as for the 2D models.

Figure 5 –Virtual Kitchen Model

5 SystemC Modelling

The SystemC Modelling starts with the creation of executable functional models in SystemC (SystemC-EFM) which at simulation time can be linked to the graphical prototype to deliver its functionality. The system models created in the early stages of design are later refined into RTL models which can eventually be synthesised.

For the digital design of SoCs the ViPERS methodology adopted the existing SystemC design flow, as shown in Figure 6. High level functional model are implemented in the early stages for design exploration, performance analysis and Hardware/Software partitioning. Once the system has been designed and the hardware/software partitioning made the refinement of the

design commences. Software engineers can implement the RTOS code while, in parallel, the hardware engineers can refine the system to Bus-Cycle-Accurate and eventually to Cycle-Accurate for synthesis.

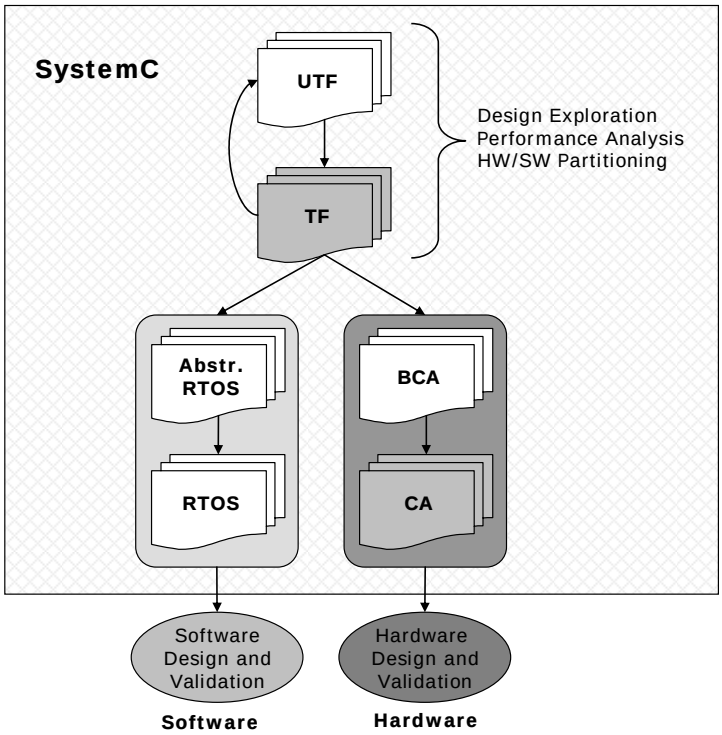


Figure 6 – SystemC Design Flow

For the remote control two SystemC models were created:

1. A SystemC executable model with CoCentric System Studio (SystemC-EFM)
2. A SystemC executable model with the SystemC free-toolkit and Visual Studio.Net (SystemC-EFM)

CoCentric System Studio [20] from Synopsis is a system-level design environment to assist in the design and simulation of System-On-Chips. The graphical user interface allows designers to build SystemC models graphically while auto generating the corresponding SystemC. It consists of tools, methodologies and libraries that support design abstraction, system-level reuse, and hardware-software co-design.

The communication of the SystemC models with its graphical counterparts is achieved by the use of a stand-alone server application, LCCS (Local Communication Control Services), which uses TCP/IP sockets. For models implemented in CoCentric an extra library (SxSockets Library [21]) is used to connect the CoCentric model to LCCS. For models implemented with the SystemC free-toolkit a SystemC container application is needed;

At present, in the ViPERS framework, there is not a solid link or automated process to link the UML and SystemC models; this means that models have to be implemented almost from scratch. Reuse currently applies to the UML written description and diagrams which are used by the SystemC designers to aid their implementation. Efforts are being made by major industrial companies as illustrated in [19] to link SystemC and UML. SystemC and UML EFMs have different nature and purpose. UML models are easier and faster to implement and are mainly used for requirements analysis. In later stages of the design the UML model might also be used for the creation of the RTOS code for the embedded processor. SystemC models are used for the digital design of the target electronic device.

5.1 Refinement Process

The first SystemC implementation of the remote electronics is a high-level functional description of the user interaction. This model executes very rapidly and is best used to represent the design while experimenting with user interactivity issues. Once the development work has reached the implementation stage, it is necessary to consider the architecture of the remote electronics, including partitioning of the functions between hardware and software. It is possible to consider several alternative solutions within the same virtual prototype environment.

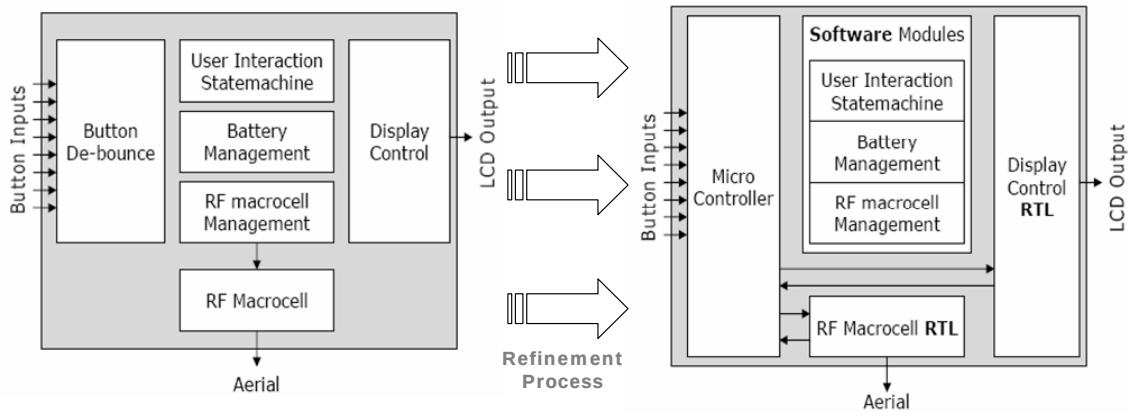


Figure 7 – Refinement process for the remote control

Figure 7 indicates a simplified block architecture of the remote control, the only block carried forward from the purely behavioural model is the user interaction state machine. The other blocks can be represented in various degrees of detail. A key question in the implementation of the device is going to be the choice of including or not a microcontroller. In the remote-control case, which includes an RF macrocell, a microcontroller is normally required to program and manage this block. Once a microcontroller is included within a design, the trade-off between moving more functions to a more capable microcontroller becomes an option. For each of the other blocks in the remote architecture it is necessary to explore discrete hardware implementation via existing IP or newly created RTL. The other design choice is software running on the microcontroller; again this can be pre-existing or newly developed.

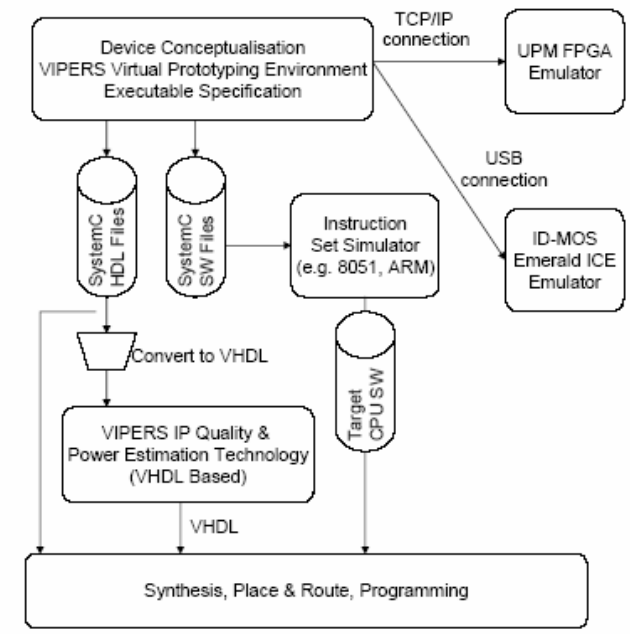


Figure 8 – ViPERS File/Net connections

Figure 7 shows how one possible refinement of the generic architecture maintains the display control and RF macrocell functions as RTL VHDL, whilst the user interaction, battery and RF management functions are implemented as software modules. The software modules of the remote are implemented as SystemC IP blocks at a high-level, the main difference between the above and the original high-level implementation is that only one of these modules is allowed to operate at any given instant.

Figure 8 gives an indication of the file and data interchange between tools within the ViPERS and includes details of the various tools being developed with the ViPERS project. Tools include the virtual prototype tooling, emulation platforms of both generic and RF macrocell focused types and tool/platform interoperability applications. The use of an instruction set simulator will form part of the next phase of methodology development.

6 Simulation Environment

When a virtual prototype has a candidate graphical representation in the VDM environment and an executable functional model in the SystemC or UML environment, the user switches from the design to the run-time mode to test and execute the virtual prototype model. The run-time mode of VDM uses TCP/IP sockets to communicate events between the virtual prototype and the functional model of the target electronic device. The communication through standard TCP/IP sockets enables models to communicate through the network. Different functional and graphical models can therefore be executed on different machines; this means that the graphical model can be distributed to the users. When the model is executed it would use the network to connect to its functional counterpart on a secure server. This simulation environment allows users to test the virtual prototype with its functional model without having access to the code of the functional model. Simulation can be performed on one isolated computer but processing power might become an issue, if the functional model is very complex or if many devices are being executed in parallel.

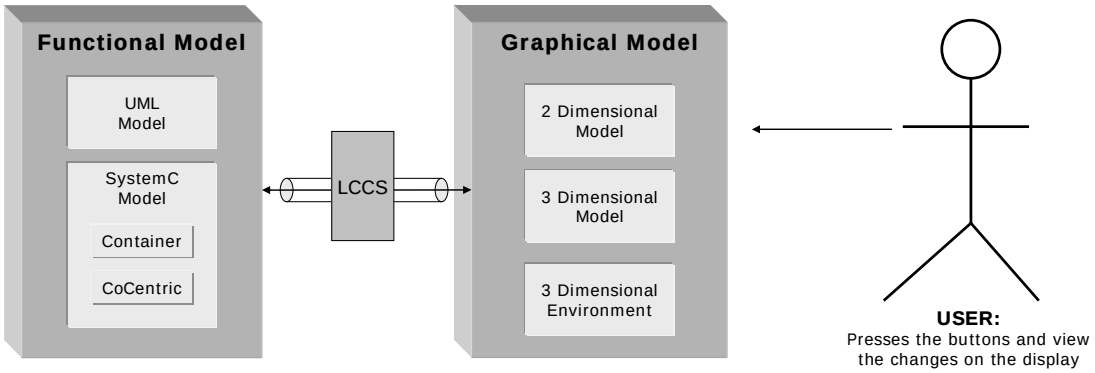


Figure 9 – Simulation Interaction

In the case of the remote control the simulation environment was made up of two virtual prototypes, the two-dimensional remote control and the two-dimensional cooking stack. Figure 9 describes the general working of the ViPERS simulation environment. A user interacts with a virtual device (graphical model) by pressing buttons or any other input mean. Changes on the display or other graphical components of the simulation are the means by which the user tests the target electronic devices. Messages between the graphical model and the functional model comply with a communication protocol and are entirely under the control of LCCS.

The diagram also shows the different options designers have when implementing both the graphical or functional models. Figure 10 (a) and (b) show a running simulation of the remote control and the cooking stack. Figure 10(a) shows the graphical model of the remote control connected with its CoCentric counterpart while in Figure 10 (b) the graphical model of the cooking stack is connected to its functional model through the Container application and therefore using the SystemC free-toolkit.

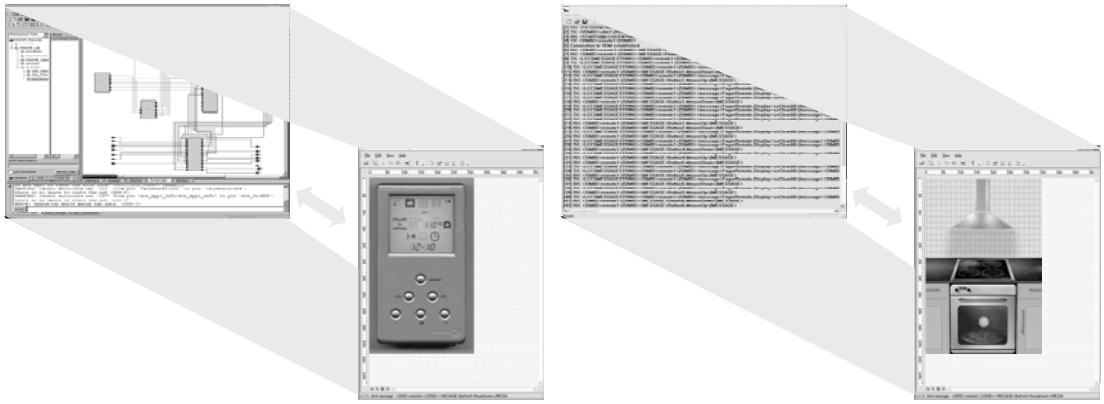


Figure a) Figure b)

Figure 10 – Simulation of the Remote Control

During the execution of the models, if a user interacts with the virtual prototype of the remote control, the interaction will show on both the display of the remote control and on the cooking stack. If a user presses a sequence of button which enables the remote control to switch one of the hobs on, the display icons will show that the hob is on and the virtual prototype of the cooking stack will graphically show the hob ring becoming red.

Figure 11 shows another simulation where the two dimensional remote control is driven by a UML state machine and it is interacting within the three dimensional virtual kitchen.

Figure 11 - Virtual prototyping simulation

7 Conclusions and Future Work

The case study of the remote control provided the ViPERS team with the possibility to experiment with the methodology in a real industrial design context. The methodology offered the exploration of key design trends such as the integration of virtual prototyping and systems level design. Major benefits of the visual layer were noted in the requirement analysis where the prototype provided a real mean to explore the interfaces early in the design stage.

Future work will be based in two main directions, one focused on SoC design and the other one on the graphics modelling.

For SoCs the focus will be on platform based designs for virtual prototypes. The idea is to provide designers with set platforms that can be customised by the addition of hardware and software blocks, either implemented or bought as IPs from IP vendors. The visual layer provided by virtual prototyping would then be integrated in the context of the platform from the exploration of the target device. Designers can then check features and performances of the device by changing blocks of the platform such as the processors or the memories. Other issues that will be explored are co-simulation of VHDL and SystemC blocks, to take further advantage of existing IP that can be reused.

For the graphics major focus will be given to virtual environments. Rich multimedia environments constituted of several devices and interfaces will provide enriched user experiences. These multimedia simulation environments will provide constraints related to the transmission and traffic imposed by the use of the virtual wireless devices. Such environments can be applied to test the usability of such complex environments as airports or stations where large amounts of people and devices interact.

8 Acknowledgment

This work was funded as part of the ESPRIT framework 5 VIPERS project IST-2000-30023. We are particularly grateful to Javier Mendigutxia and his colleagues at Ikerlan SA for their contribution to this work. The cooperation of our colleagues Teresa Riesgo and Eduardo de la Torre of Universidad Politécnica de Madrid and Sabastian Pantoja of Celestica Valencia is acknowledged.

9 Reference

[1] *Electronic Simulation for Virtual Reality: Virtual Prototyping*, P.F. Lister, P.L. Watten, M.R. Lewis, P.F. Newbury, M. White, M.C. Bassett, B.J.C. Jackson, V. Trignano, Theory and Practice of Computer Graphics 2004 (TPCG04), Southampton, UK, June 2004.

[2] *Virtual Reality for Electronic Product Development of Hand Held Devices*, P.F. Lister, P. Watten, P. Newbury, M. Bassett, B. Jackson, V. Trignano, Design Automation and test in Europe 2004 (DATE ’04), Paris, February 2004.

[3] *VIPERS Project references and web pages*, http://www.upmdie.upm.es/projects/vipers/

[4] *International Technology Roadmap for Semiconductors* (ITRS), 2003 Edition.

[5] Ikerlan, http://www.ikerlan.es/pub/cast/index.htm

[6] *Unified Modelling Language (UML)* – Version 1.5, OMG document formal/2003-03-01, Object Management Group, Needham MA, 2003.

[7] *UML profile for Schedulability, Performance, and Time*, OMG document ptc/03-02-03, Object Management Group, Needham MA, 2002.

[8] *Open SystemC Initiative*. See http://www.systemc.org/

[9] SystemC Version 2.0.1 *Users Guide*.

[10] *System Design with SystemC*, Thotsten Grotker, Stan Liao, Grant Martin, Stuart Swan, KAP, 2002, 1-4020-7072-1.

[11] *Product Development with Mathematical Modeling*, Rapid Prototyping, and Virtual Prototyping, Ikuo Kimura, Shaker Verlag, June 2002, ISBN 3-8322-0896-8, Chapter 1.

[12] *Rational Unified Process® for Systems Engineering*, http://www.rational.com/

[13] *Interaction Design, beyond human-computer interaction*, Jennifer Preece, Yvonne Rogers, Helen Sharp, John Wiley & Sons, Inc. 2002, ISBN 0-471-49278-7.

[14] *UML-Executable Functional Models of electronic systems in the VIPERS Virtual Prototyping Methodology*, P.F. Lister, V. Trignano, M.C. Bassett, P.L. Watten, FDL '04, Lille-France, 13-17 September 2004.

[15] *Real-Time UML Second Edition, Developing Efficient Objects for Embedded Systems*, B. P. Douglass, Addison-Wesley, 1999, 0201657848.

[16] *Virtual Reality in Electronic Systems*, Lister, P. F., Newbury, P. F., Watten, P. L., Senkoro, L., Dountsis, A., Midha, M., Banerjee, I., Trignano, I., and White, M., Proceedings of 5th International Conference on Business Information Systems, Poznan, Poland, April 2002. Pp. 390-394.

[17] *RAPID virtual prototyping tools*, e-SIM LTD, http://www.e-sim.com/

[18] *High Level Modelling and Design of Display Systems*, Philip Laurence Watten, Doctoral Thesis, September 2002, School of Engineering and Information Technology, University of Sussex, Brighton.

[19] *SoC Design with UML and SystemC*, Alberto Sardini, European SystemC, 6.Users Group Meeting, Lago Maggiore, October 2002.

[20] Synopsys. CoCentric System Studio *User Guide Manual*.

[21] *Extending SystemC for high-level multi-platform SoC simulations*, Vincenzo Trignano, Mike Bassett, Phil Watten, Paul Lister, IEE Postgraduate Colloquium on System-on-Chip Design, Test and Technology, Cardiff University, September 2, 2003.

Design and test of a task guardian for use in TTCS embedded systems

Zemian M. Hughes and Michael J. Pont

Embedded Systems Laboratory, University of Leicester,

University Road, LEICESTER LE1 7RH, UK.

Abstract

If software for embedded processors is based on a time-triggered, co-operatively-scheduled (TTCS) software architecture, the resulting system can have very predictable behaviour. Such a system characteristic is highly desirable in many applications, including (but not restricted to) those with safety-related or safety-critical functions. Despite many advantages, TTCS designs can be compromised by tasks that fail to complete within their allotted period. In this paper, we discuss the impact that such task overruns can have, and describe the implementation of a range of "task guardians" that can address this problem. We then demonstrate the application of a task guardian in a simple case study.

1. Introduction

Embedded systems are often designed and implemented as a collection of communicating tasks (e.g. Nissanke, 1997; Shaw, 2001). The various possible system architectures may then be characterised in terms of these tasks: for example, if the tasks are invoked by aperiodic events (typically implemented as 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.[1]

Various studies have demonstrated that, compared to pre-emptive schedulers, co-operative schedulers have a number of desirable features, particularly for use in safety-related systems (Allworth, 1981; Ward, 1991; Nissanke, 1997; Bate, 2000). For example, Nissanke (1997, p.237) notes: *"[Pre-emptive] schedules carry greater runtime overheads because of the need for context switching - storage and retrieval of partially computed results. [Co-operative] algorithms do not incur such overheads. Other advantages of [co-operative] algorithms include their better understandability, greater predictability, ease of testing and their inherent capability for guaranteeing exclusive access to any shared resource or data."*. Allworth (1981, p.53-54) notes: *"Significant advantages are obtained when using this [co-operative] technique. Since the processes are not interruptable, poor synchronisation does not give rise to the problem of shared data. Shared subroutines can be implemented without producing re-entrant code or implementing lock and unlock mechanisms"*. Also, in a recent presentation, Bate (2000) identified the following four advantages of co-operative scheduling, compared to pre-emptive alternatives: [1] The scheduler is simpler; [2] The overheads are reduced; [3] Testing is easier; [4] Certification authorities tend to support this form of scheduling.

Overall, time-triggered, co-operatively scheduled (TTCS) system architectures are known to provide a simple, low-cost and highly predictable platform. While such an architecture is not suitable for all systems, we have demonstrated in previous studies (e.g. Pont, 2001; Pont, 2003; Pont and Banner, 2004) that a TTCS approach can prove effective in a very wide range of single- and multi-processor systems.

Despite many advantages, a pure TTCS architecture has a failure mode which has the potential to greatly impair system performance: this failure mode relates to the possibility of task overruns (see Figure 1).

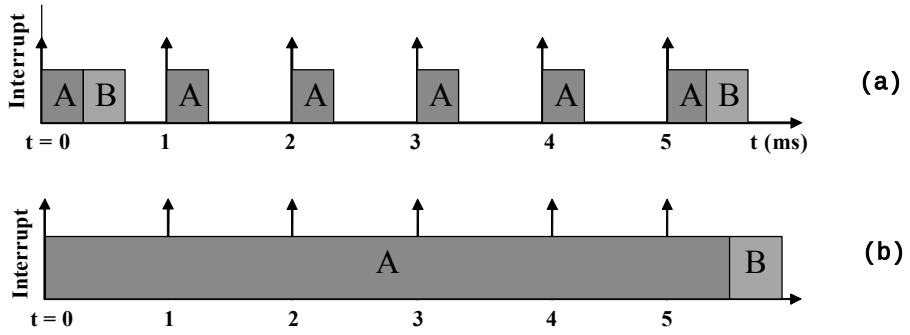


Figure 1: Illustrating the impact of task overruns on a TTCS system. See text for details.

Figure 1a illustrates a TTCS design running two tasks, A and B. Task A runs every millisecond, Task B is a "slack stealing" task that runs every 5ms. This system operates as required, since the duration of Task A never exceeds 0.4 ms. Figure 1b illustrates the problems that result when Task A overruns: in

[1] Note that these categories are not exclusive. For example, a complex system may support both time-triggered and event-triggered tasks; some of these may be co-operative in nature while others are pre-emptive.

this case, we assume that the duration of Task A increases to approximately 5.5ms. The co-operative nature of the scheduling in this architecture means that this task overrun has very serious consequences. In practice, the situation may be even more extreme: in this example, if Task A never completes, then Task B will never run again.

In this paper, we describe a number of "task guardian" mechanisms which may be used to address this problem, by allowing the scheduler to shut down and - in some cases - replace tasks that exceed a pre-determined maximum duration. The examples used are based on the TTCS architecture described in detail previously (see Pont, 2001). In this case, we worked with a version of this scheduler ported to an ARM7 platform: the specific processor used was the Philips LPC2106.

2. Basic solution

We begin by describing a basic solution to the problem of task overruns.

2.1 Normal operation of the TTCS architecture

During normal operation of the TTCS architecture described by Pont (2001), the first function to be run (after the startup code) is Main. Main calls Dispatch which in turn launches any tasks which are currently scheduled to execute. Once these tasks are completed, Dispatch calls Sleep, placing the processor into a suitable "idle" mode. A timer-based interrupt occurs every millisecond (in most implementations) which wakes the processor up from the idle state and invokes the ISR Update. Update identifies tasks which are due to be launched during the next execution of Dispatch. The function calls return all the way back to Main, and Dispatch is called again. The cycle thereby continues.

This process is discussed in more detail by Pont (2001) and is summarised in Figure 2.

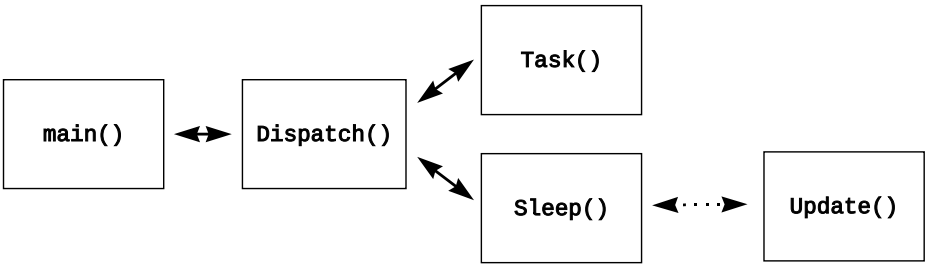


Figure 2: Function call tree for the TTCS architecture (normal operation).

If a task overrun occurs, then - instead of Sleep being interrupted by the ISR - the overrunning task is interrupted (Figure 3).

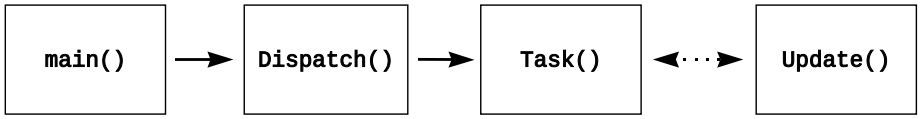


Figure 3: Function call tree for the TTCS architecture (task overrun condition).

If Task keeps running then - in the standard TTCS scheduler - it will be periodically (very briefly) interrupted by Update. However, it cannot be shut down.

2.2 Summary of required Task Guardian behaviour

The Task Guardian - implemented mainly in a revised version of Update - is required to shut down any task which is found to be executing when the Update ISR is invoked.

This will be carried out as follows (see Figure 4):

- The Update ISR will detect the task overrun.

- Update will return control to End_Task (rather than to the problematic task). End_Task will be responsible for unwinding the stack, as required, to locate the return address to Dispatch.
- Control will then be returned to Dispatch, and - as far as possible - normal program operation will continue.

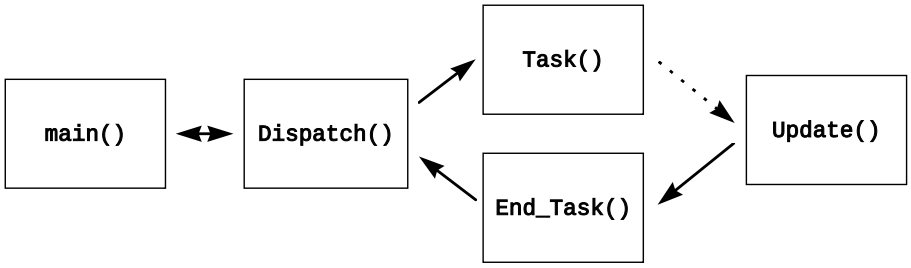


Figure 4: Function call tree when recovering from a task overrun.

Each of the stages summarised above will now be considered in turn.

2.3 Detecting task overruns

In order for the update task function to know that an overrun has occurred and take appropriate action, a simple and reliable method is required to detect overruns. Modifying the code in Dispatch, where the tasks are launched, enables this to be achieved.

Using a variable such as 'Taskover' the task ID can be stored before the task is executed (Figure 5). When the task complete, 'Taskover' is assigned a value of 255, a reserved ID to indicate successful task completion.

```
Taskover = Index;               // Store task ID
(*SCH_tasks_G[Index].pTask)();  // Run the task
Taskover = 255;                 // Task completed
```

Figure 5: Overrun detection in Dispatch.

As it stands, this process may generate a "false alarm", in situations where an interrupt occurs between the completion of the task and the assignment of the value 255 to 'Taskover'. Although unlikely, this problem is potentially serious since, in these circumstances, the Task Guardian would identify Dispatch as a problematic function and shut it down.

To resolve this problem code labels may be used (Figure 6). In the event of a suspected overrun, these labels allow the Guardian to check whether the current code location lies within Dispatch.

```
DISP_CRITICAL_START:
Taskover = Index;               // Store task ID
(*SCH_tasks_G[Index].pTask)();  // Run the task
Taskover = 255;                 // Task completed
DISP_CRITICAL_END:
```

Figure 6: Safer overrun detection in Dispatch using code labels.

2.4 Returning from Update

If a task overrun has occurred then Update must alter its return address so that - instead of returning to the overrunning task - it returns to End_Task (see Figure 4).

Please note that the End_Task function is required because Update is an FIQ (Fast Interrupt Request) ISR which - in the ARM architecture used here - has a separate stack and hence a different set of frames: unlike Update, End_Task runs in User mode and therefore has access to the stack and frames

used by the overrunning task. It is the job of End_Task to back-trace and rewind the function calls until the return address to Dispatch is located. The end task function then returns control to Dispatch.

Update must determine how the return address is stored and check if the address lies outside the critical code labels (as described in Section 2.3), indicating that the task has not returned before setting the 'taskover' variable. The original Update return address is stored so that End_Task can determine where the task overran. The Update return address is then replaced with the start address of End_Task.

Controlling the transfer of control from Update to End_Task requires care because there are three ways in which the return address for Update may be stored. These are [1] through frames (following the ATPCS standard: see ARM (1998)); [2] directly storing the return address on the stack, or [3] storing the address in the link register (lr, also known as 'r14').

2.5 Shutting down the task

Having detected a task overrun in Update and changed the return address, control is transferred to End_Task which must shut down the overrunning task.

End_Task determines whether the overrunning task was a leaf function or a function containing sub functions (with frames in the stack). The frame pointer (fp) register is compared with the saved fp register value in Dispatch: if these values are equal, this indicates that the overrunning task is a leaf function. If the values are different then the function calls are back traced (using the ATPCS standard) until the frame-stored fp register is equal to the fp register value saved in Dispatch. The current processor fp register is then made equal to the stored fp register.

End_Task is then able to check if any registers contents were stored on the stack when the task was called. The store instruction is found by looking at the contents of the address referred to by the frame pointer (which identifies the first line of code in the function). By subtracting 12 from this address, locating it and comparing the contents with the instruction number for a block transfer function, a store instruction can be identified.

The store block transfer instruction is then decoded to recover important settings (such as the names of registers which were stored on the stack, and whether the stack is ascending or descending). The required bits of the store instruction are modified to convert it to a load instruction. The new instruction is then pointed to by the address in r13, which is loaded into the program counter. The microprocessor then runs this instruction from RAM: this initiates the return sequence to Dispatch with the saved registers restored. Note that all the important registers are stored before and after the task is called to add an extra level of security from register corruption.

If the overrunning task is a leaf function then the end task function simply returns where the r14 (link) register, which contains the return address of the dispatch task function, is loaded into the PC (program counter) register.

When a task is to be shutdown a flag is set in Update so that Dispatch will loop through the task array again: this avoids the possibility of a tick offset error.

3. Adding support for backup tasks

The ability to shut-down tasks that overrun, as described in Section 2, has the potential to greatly improve the reliability of TTCS designs. However, avoiding scheduler "jams" may not completely address the underlying problem, particularly if a critical system operation is rendered inoperative through the shutdown process. In such circumstances, we require a way of invoking backup tasks in place of a task or tasks that have overrun. Alternatively, if a backup task does not exist then the

scheduler can make the overrunning task the lowest priority, so that - should it overrun again - its impact will be reduced.

This section describes the extra code required to implement these features.

3.1 Backup tasks solution

To support backup tasks, a second function pointer is required in the task array (for each task). This will hold the address to the backup task if one exists. The method of setting up a task in the scheduler with a backup task is shown in Figure 7.

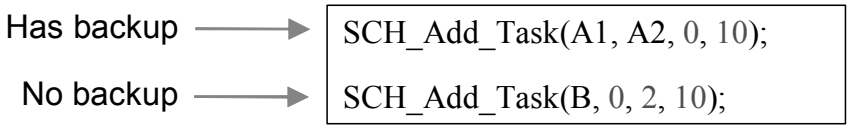


Figure 7: Setting up tasks with and without backup tasks

When - in Update - a task overrun is detected then the backup task address in the tasks array is checked for a non-zero value. If such a value is found, the original task is replaced. A variable - 'Backup_G' - is set to a value of 2, so that when the overrunning task is shutdown the backup task will run immediately (before running any other pending tasks).

```
// Setup backup task if one exists and not already running backup task
if ((SCH_tasks_G[Task_Overrun_G].pTask_2 > 0) & (SCH_tasks_G[Task_Overrun_G].pTask
      != SCH_tasks_G[Task_Overrun_G].pTask_2))
     {
     // Set backup task
     SCH_tasks_G[Task_Overrun_G].pTask = SCH_tasks_G[Task_Overrun_G].pTask_2;

     // Set it to run immediately
     Backup_G = 2;
     }
```

Figure 8: Update tasks setting of backup task to run instead of overrunning task.

If tasks are setup as shown in Figure 7, then the response shown in Figure 9 will be seen in the event that A1 overruns.

Figure 9: Task A overruns so backup task A' is run immediately to recover

3.2 Task Priority solution

If the backup task itself overruns or there is no backup task then the next best thing to do may be to set the overrunning task to the lowest priority. This means that the task will become the last task to be run in any given tick interval. This is achieved by holding an array called 'Priority' of index values for the tasks in the task array. In this way the order of the tasks in the tasks array can be altered without shifting all the values in the task array around. When the tasks are initially set up the indexes in the priority array are in order of first task to last. If a tasks priority is to be changed then its index value is placed at the end of the priority array and the other values shifted up. This can be seen in the code in Figure 10.

```
// Otherwise all we can do is change priority
else
    {
    // If not already lowest priority
    if (Task_Overrun_G != Priority[SCH_MAX_TASKS - 1])
        {
        // Store current task id
        tmp = Priority[Task_Overrun_G];

        // Shift lower priority tasks up
        for (Index = Task_Overrun_G; Index < SCH_MAX_TASKS - 1; Index++)
            {
            Priority[Index] = Priority[Index + 1];
            }

        // Place overrun task in lowest position
        Priority[SCH_MAX_TASKS - 1] = tmp;
        }
    }
```

Figure 10: Update tasks changing of task priorities

As can be seen in Figure 11, Figure 12 and Figure 13, the effect of the task priority system helps to prevent task overruns affecting other tasks. Figure 12 shows how Task B will become offset due to the overrunning task A. With the task priority system the tasks can operate in there normal tick intervals and task A given the spare processing time to try to recover itself as seen in Figure 13.

Figure 11: Tasks A and B without overrun

Figure 12: Effect with no task priority system where task A overruns

Figure 13: Effect with task priority system where task A overruns

4. Implementation costs

Clearly, the addition of a task guardian and related features adds to the complexity of the code.

To illustrate the typical costs, the code size changes for one implementation of the mechanism described in this paper are shown in Table 1.

Features	Instructions	Increase in instructions
Base Scheduler (reduced jitter)	163	-
Basic Task Guardian	214	31%
+ Tasks modification protection	259	59%
+ Recover stack variables	277	70%
+ Backup tasks feature	359	120%
+ Priority tasks feature	421	158%

Table 1: A comparison of the number of instructions required to implement various versions of the scheduler.

5. Simple case study: Data acquisition

This section illustrates the operation of the Task Guardian described in the previous sections.

5.1 Test platform

This test was carried out using two MAX1110 ADCs (where one is for backup), and a MAX548 DAC as seen in Figure 14. They interface with the ARM processor via the SPI serial interface and three chip select pins.

In normal operation, analogue values are read from the first ADC and then reported back out to the DAC. After ten samples are read the function reading the first ADC will overrun, and the scheduler will run the backup function which will read samples off the second ADC and send them to the DAC.

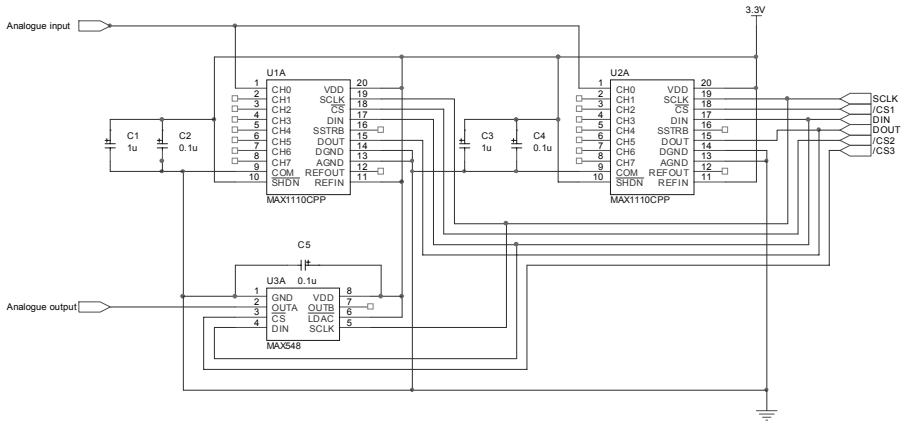


Figure 14: Test circuit used with dual ADCs and a single DAC. See text for details.

The test made use of critical and backup functions, where the code for the critical function is show in Figure 15. The code is very simple, where a counter variable counts the number of times the function has been called. After 10 executions, the function will overrun. The sample is read from the first ADC by sending 0 to the function, and the data returned in to temporary variable, 'data': these data are then sent out to the DAC.

```
// ADC and DAC critical function
int ADC_DAC_CRTL(void)
   {
   static tByte Count;
   tByte data;

   // Overrun every 10 times the function is called
   if (++Count == 10)
      {
      Count = 0;

      // Generate an overrun
      while(1);
      }

   // Read analogue value from first ADC
   data = SPI_MAX1110_Read_Byte(0);

   // Send value to DAC
   SPI_MAX548A_Send_Byte(data);

   return 0;
   }
```

Figure 15: ADC and DAC critical function used for tests

Figure 16 shows the backup function which reads a sample of the second ADC by sending a 1 to the ADC function. The value is then sent directly to DAC, where normal operation should continue for another 10 samples.

```
// ADC and DAC backup function
void ADC_DAC_Backup()
   {
   tByte data;

   // Read analogue value from BACKUP ADC
   data = SPI_MAX1110_Read_Byte(1);

   // Send value to DAC
   SPI_MAX548A_Send_Byte(data);
   }
```

Figure 16: Backup ADC DAC function.

5.2 Results

The ADC DAC test was carried out using a 500Hz sine wave produced by a signal generator.

Please note that the system code was modified so that when the overrun begins the DAC outputs its maximum value and when the backup function is called, the DAC it outputs its minimum value. This was done so that the overruns and recoveries would be clearly visible.

From the waveform in Figure 17 it can be seen that the scheduler is repairing the task overruns and continuing its normal operation between overruns.

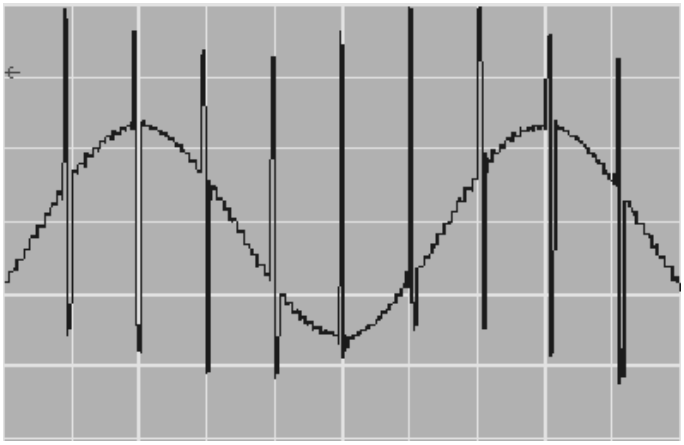


Figure 17: Output waveform from the DAC during system test.

6. Conclusions

In this paper, we have introduced a simple but effective task guardian which may be used to reduce the impact of task overruns on TTCS embedded systems. We have illustrated the use of this task guardian in a simple case study.

7. References

Allworth, S.T. (1981) *"An Introduction to Real-Time Software Design"*, Macmillan, London.

ARM (1998), "The ARM-THUMB Procedure Call Standard", SWS ESPC 0002 A-05, http://www.arm.com

Bate, I. (2000) "Introduction to scheduling and timing analysis", in *"The Use of Ada in Real-Time System"* (6 April, 2000). IEE Conference Publication 00/034.

Nissanke, N. (1997) *"Realtime Systems"*, Prentice-Hall.

Pont, M.J. (2001), "Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of micro controllers", Addison-Wesley / ACM Press.

Pont, M.J. (2003) "Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns", *Informatica*, **27**: 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.

Ward, N. J. (1991) "The static analysis of a safety-critical avionics control system", in Corbyn, D.E. and Bray, N. P. (Eds.) "*Air Transport Safety: Proceedings of the Safety and Reliability Society Spring Conference, 1991"* Published by SaRS, Ltd.

GAMMA RAY PEAK DETECTION ALGORITHMS USING EMBEDDED DSP

M.W.Raad[1] , J.M.Noras[2] , M. Shafiq[3], A. Aksoy[4]

[1]Computer Engineering Department,
King Fahd University of Petroleum and Minerals,
Dhahran, Saudi Arabia
mwraad@ccse.kfupm.edu.sa

[2]School of Engineering,
University of Bradford, Bradford, UK
jmnoras@bradford@ac.uk

[3]System Engineering Dept
King Fahd University of Petroleum and Minerals,
Dhahran, Saudi Arabia
mshafiq@ccse.kfupm.edu.sa

[4]Center for Applied Physical Sciences,
King Fahd University of Petroleum and Minerals,
Dhahran, Saudi Arabia
aaksoy@kfupm.edu.sa

Abstract

A fast waveform sampling facility has been recently developed and integrated into the VAX-based data acquisition system at the Centre for Applied Physical Sciences (CAPS). This study uses the above facility in developing algorithms for digitally determining the basic pulse parameters and tackling the problem of pulse pileup in Gamma-ray spectroscopy. We have developed a number of parameter estimation and digital online peak detection algorithms including a pulse classification technique that uses a simple peak search routine based on the smoothed first derivative method. This algorithm gives a percentage error of peak amplitude of less than 1%. A moment-preserving finite input deconvolution filter of 3 coefficients and 4 coefficients have been tested successfully to resolve pileup to an average percentage of 93% pileup free, or approximately eight fold pileup suppression, with the 3-point filter giving the minimum percentage error in peak detection of less than 1% compared to other peak detection algorithms. The classification technique has the unique feature of cutting down the computation largely by only allowing the event of interest to be executed by a particular algorithm. The set-up was also tested with random signals from a ^{137}CS test source. Gamma-ray pulses from a 3^{-inch} Na(TI) scintillation detector were captured as single and double pulses for the purpose of testing the peak detection algorithms. The pulse classification technique was tested successfully in real-time on a TMS320C6711 high performance floating-point processor, using the code composer studio and evaluation DSP board. The total execution time was around 2 msec, giving a throughput of approximately 486 events per second without losing information. The moment preserving 3-point deconvolution filter took only 52µsec. This constitutes the basis for future use of the TMS320C6711 as an embedded processor in the Gamma-ray peak detection instrumentation devices.

Keywords: *Deconvolution, Pileup, Spectroscopy, Real-time, Ballistic deficit, Savitzky Golay, moment-preserving*, DSP, embedded.

I. PULSE PILEUP IN GAMMA-RAY SPECTROSCOPY

A common problem in nuclear spectroscopy is pulse pileup caused by the non-zero response time of the detection system. For germanium detectors, the time required to collect all the ionisation current associated with an event ranges from 0.5 to 6.0 μs [4]. The fact that pulses from a radiation detector are randomly spaced in time can lead to interfering effects between pulses when counting rates are not low. These effects are generally called pileup and can be minimised by making the total width of the pulses as small as possible.[6]. Pileup phenomena are of two types. The first type is known as tail pileup and involves the superposition of pulses on the long duration tail from a preceding pulse (see Fig 1).

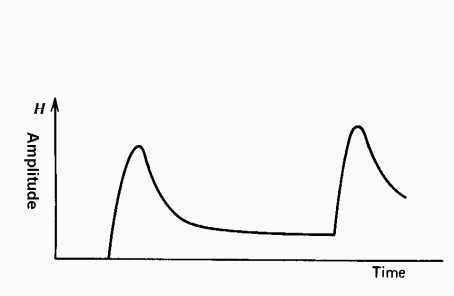


Figure 1. pileup effect on a pulse peak from the tail of a preceding pulse

Tails can persist for relatively long periods of time so that tail pileup can be significant even at relatively low counting rates. A second type of pileup is the peak pileup, which occurs when the mutual pulse spacing between the two overlapping pulses is less than T_p where T_p is the peaking time or peak location [20]. Researchers often simply reject pileup, when it is recognised [3]. In our case, we do not wish to lose the information associated with such events, but we propose to reject peak pileup since it occurs less frequently than tail pileup in nuclear spectrometry, so we use a window-based pulse classification technique based on a smoothed derivative search technique, which has been developed and verified to give a percentage error in peak amplitude of less than 1% in a previous publication [19]. The advent of very fast flash ADCs (FADCs) together with fast memory buffers and high speed digital signal processing devices has made it possible to digitize and acquire the whole pulse waveform with adequate resolution in both time and amplitude, and compute pulse parameters of interest at near real time speed. The motivation of this study was to try to come up with a practical pileup rejection criteria in addition to implementing the classification technique in real time using a floating point C6000 DSP processor. Another scope of the study was to develop peak detection algorithms to resolve pileup in Gamma Ray events.

II. PEAK DETECTION USING DECONVOLUTION

A linear time-invariant system takes an input signal x(n) and produces an output signal y(n), which is the convolution of x(n) with the unit sample response h(n) of the system.

In many practical applications we are given an input signal x(n)= xd(n)+ xi(n), where xd(n) represents a desired signal sequence and xi(n) represents some undesired interference or noise component, and we are asked to design a system that will suppress the undesired interference component. In such a case, the objective is to filter out the additive interference and noise while preserving the characteristics of the desired input signal xd(n).

There is another class of problem which exist in nuclear spectrometry applications, where we are given an output signal measured from a shaping amplifier system whose characteristics are well known and modelled, and we are asked to determine the input signal. We know that the shaping system has some effect on the input pulse expressed in terms of the impulse response of the system. So the problem is to design a corrective system which, when cascaded with the original system, produces an output which is in some sense a replica of the input. In linear system theory this corrective system is called an inverse system, because the corrective system has a frequency response that is the reciprocal of the frequency response of the shaping system. Furthermore, since the shaping system gives an output y(n) that is the convolution of the input x(n) with the impulse response h(n), the inverse system operation that takes y(n) and produces x(n) is called deconvolution [6].

Many researchers have used deconvolution algorithm to reconstruct the initial detector impulse signal from the shaped CR-RC amplifier pulse. The simple CR-RC impulse response, $(t/\tau).e^{-t/\tau}$, was chosen because of its good signal-to-noise ratio while it lends itself to making the deconvolution equation very simple. The ideal pulse shape, from a noise point of view, is a cusp-like pulse [7]. Gaussian and triangular pulses are almost as good, but are very difficult to build. The amplifier output after being digitised through an ADC, can be processed to yield amplitude and timing of the charge pulse from the detector. The deconvolution achieves this by forming a weighted sum of three or more samples of the amplifier output [8,9,10]. It is well known that v(t) the output of the shaping system, h(t) the impulse response of the amplifier/shaper, and s(t) the input signal, are related in the time domain by a convolution integral. This can be written

$$v(t)= \int h(t-t').s(t').dt' \qquad (1)$$

If we sample output v(t) at regular intervals, then we can write the equation in a matrix form as

$$V_i= \sum H_{ij}S_j \quad \text{or} \quad V=H.S \qquad (2)$$

The original signal impulse can be recovered by performing the inverse operation

$$S=W.V= H^{-1}.H.S \qquad (3)$$

and the elements of the weight matrix W can be found by numerical matrix inversion of H. The transfer function for the CR-RC shaping system is:

$$\frac{\tau 1}{C_i.(1+s\tau 1).(1+s\tau 2)} \qquad (4)$$

which shows that the system is an all pole system. The impulse response of the system has the form:

$$\frac{\tau 1\left[e^{\frac{-t}{\tau_1}} - e^{\frac{-t}{\tau_2}}\right]}{C_i.\left[\tau_1 - \tau_2\right]} \quad \text{for } t>0. \qquad (5)$$

$\tau 1$ is chosen to be equal to $\tau 2$ which gives the best combination of flat pulse top and quick return to zero and corresponds to maximum signal to noise ratio. For $\tau 1=\tau 2$ the above expression be resolved by l'Hopital's rule which yields an impulse response of

$$\frac{1}{C_i.\tau_1}.t.e^{\frac{-t}{\tau 1}} \quad \text{for } t>0. \qquad (6)$$

The actual value of $\tau 1$ in a practical system would be set by the requirements of ballistic deficit, pileup and noise considerations. For scintillation counters (as in our system), the value of $\tau 1$ is not usually critical and a value of 1 μs is chosen [5]. Hence the system response is described by

$$V(t) = \frac{t}{\tau}.e^{\frac{-t}{\tau}} \qquad (7)$$

where the theoretical peak is expected at $t=\tau$, and the peak value = $v(\tau)=\exp(-1)$.

By matrix inversion of the impulse response matrix, it is shown that only three non-zero weights are necessary for the CR-RC and

$$S_k=w_1.v_k+w_2.v_{k-1}+w3.v_{k-2} \quad (8)$$

$$\text{with values} \quad W_1 = \frac{1}{x} \cdot e^{x-1} \quad (9)$$

$$W_2 = -2.\frac{1}{x} \cdot e^{-1} \quad (10)$$

$$W_3 = \frac{1}{x} \cdot e^{-x-1} \quad (11)$$

where $x=\Delta t/\tau$ and Δt is the sample interval. This implies that a filter performing this operation can be constructed by forming the weighted sum of three consecutive voltage samples in time [1,2,11,12,13].

It is easily shown that for $\theta=0$, the CR-RC impulse response reaches its maximum peak = $(1/\tau).\exp(-1)$ at $t = \tau$, and for $\tau = 0.5$ μsec, this value would be 0.147 , and this is the expected peak value without noise. Hence, the expected peak location without noise is at $t= \tau$, or $\theta+ \tau$ if θ is not zero.
Researchers showed that the 3-point deconvolution produces zero whenever all three samples are on the pulse. The only time when it is possible for the deconvolution to produce non-zero data is when either one or two samples are on a pulse, and the third is on the pedestal. This shows that it is the sample amplitude of the amplifier output, which is first multiplied with w1, which determines the true value of the deconvoluted output. This single sample time unit resolution is very difficult to attain, so researchers were satisfied with double pulse resolution [17].

It has been shown by Hall that the weighting function of the 3-point deconvolution is a triangular shape response with a sharp edge corresponding to complete input reconstruction. A robust algorithm to maximise signal at the expense of some loss in time resolution is to make a sum of two deconvoluted samples.

$$S_k=S_k+S_{k+1}\,, \quad (12) \quad \text{where}$$
$$S_k=w_1.v_k+w_2.v_{k-1}+w_3.v_{k-2} \text{ and} \quad (13)$$

$$S_{k+1}=w_1.v_{k+1}+w_2.v_k+w_3.v_{k-1} \quad (14)$$

This will lead to

$$S_{k=}\, w_1.v_{k+1} + (w1+w2).\, v_k +(w2+w3).\, v_{k-1} + w3.\, v_{k-2} \quad (15)$$

This is equivalent to using an algorithm with 4 weights that are easily calculated from the three original weights [16]. The resulting weighting function is the sum of the two individual weighting functions [12]. The weighting function for a 3-point deconvolution was shown to be of triangular shape in Hall, and its importance that it quantifies effect of timing errors on deconvolution [14]. Using the same procedure of a combination of more than two 3-point deconvolution, can be easily proved to lead to the following 5-point deconvolution equation:

$$S_k= w1.v_{k+2} +(2.w1+w2).v_{k+1} +(w1+2.w2+w3).v_k +(w2+2.w3).v_{k-1} +w3.v_{k-2} \quad (16)$$

Hence, deconvolution provides a way to remove pulse broadening that was deliberately introduced by the preamplifier. Researchers at CERN used this algorithm to give them a deconvoluted pulse that was confined to only two successive non-zero samples, attaining the required timing resolution [15]. In order to remove part of the noise added by deconvolution algorithms, we have compared the performance of 3-point and 4-point deconvolution with their performance after adding a pre-filter stage. Two well known filters used in spectroscopic applications have been considered, the moving average filter and the Savitzky Golay filter or moment-preserving filter. Moving average filter provides significant noise reduction. However, signal distortion also occurs especially in the higher moments. To reduce this distortion while retaining noise attenuation, simple polynomials can be fitted to data. Savitzky and Golay involve

fitting a polynomial to a set of noisy data within a window of points. Using higher order polynomials should allow fitting of data that change rapidly within the filter window. As long as the noise continues to change more rapidly than the data, good noise rejection will be possible. With low order polynomials, more noise is filtered, but at the cost of signal distortion. Also increasing the size of the window used in the filtering decreases noise, as with the moving average filter, but at the cost of significant signal distortion [37]. A Savitzky Golay filter of order 4 and window 27 points has been found optimum.

One of the performance measures used to compare the performance of different deconvolution algorithms in peak detection is the mean percentage error in a particular parameter, or MPE. The MPE is evaluated according to the formula: 100*(Mean expected value-Mean estimated)/Mean expected).

After comparing the mean percentage error in the peak amplitude for various peak detection algorithms, the 3-point deconvolution with savitzky golay pre-filter gave the best performance corresponding to less than 1% of mean percentage error in peak amplitude for very low SNR, with the extra cost of additional computational complexity by approximately 12 fold compared to 3-point deconvolution, as shown in figures 6 and 7.

III. PEAK DETECTION IN THE PRESENCE OF PILEUP

For the purpose of removing tail pileup only, we have evaluated the three coefficient and four-coefficient deconvolution in this context. In order to be able to cut down the computational complexity of the pileup processing techniques, we have to look at the case of peak pileup in a more qualitative manner. See figure 2 for the effect of varying degrees of overlap on the pulse pileup.

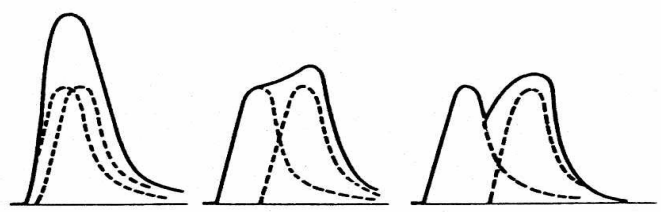


Figure 2. Shows effect of varying degrees of overlap on the Gamma-ray pileup

An extensive study was made on simulating a pileup of two superimposing events of same height and separated by varying displacement in time from $d=\tau/10$ (complete overlap) to $d=7\tau$ (complete pulse separation), using the CR-RC pulse model. It is well known that in tail pileup, the amplitude of the first pulse is not distorted while the amplitude of the second pulse is distorted. However, when the pulse separation is less than the rise time, a partial or composite sum occurs. A special case of this, is when the two pulses completely superimpose, or the pulse separation is much less than the rise time, the two pulses appear as one distorted pulse of double the original amplitude, where both of these cases are categorised as peak pileup. This phenomenon has been verified by simulating the pileup of two pulses of equal and different peak amplitudes separated by less than τ. See figure 2.

By observing the simulation carefully, we found that starting from a pulse separation of $\approx \tau$ (peak portion of the pulse), the first pulse amplitude is preserved while the second pulse amplitude is distorted, and this occurs at a sampling index of $52t_s$, where t_s is the sampling interval. The pulse width 7τ value can be considered as the pulse separation threshold below which pileup occurs.

By comparing the simulated results and the real scenario results, we conclude that both the simulation and the real results agree in the capability of 3-point deconvolution and the 4-point to resolve tail pileup except in some few cases where the time separation of the two events is less than τ, which is categorised as a peak pileup. Since the 3-point deconvolution fails to resolve such pileup, it is best to reject. Researchers have already proposed a number of digital pileup rejecter circuits that can easily be incorporated into the whole pulse analysis system [18].

The set-up was also tested with random signals from a ^{137}CS and a Na-22 test sources. Gamma pulses from a 3'' Na(TI) scintillation detector were captured as single and double pulses for the purpose of testing the peak detection algorithms. The energy resolution of the detector was measured to be 53 keV at 662 keV of ^{137}Cs. The initial percentage of pileup

measured from the Gamma pulse content, was estimated to 56% of Gamma., the 3-point deconvolution technique applied to tail pileup events digitised at the CAPS facility, succeeded in resolving pileup by attaining a performance of approximately 93% pileup free events or 7% remaining tail pileup. This amounts to approximately eight-fold pileup suppression [19]. See figures 3 and 4 for pulse height spectrum of Cs-137 and Na-22 sources respectively in the case of pileup rejection.

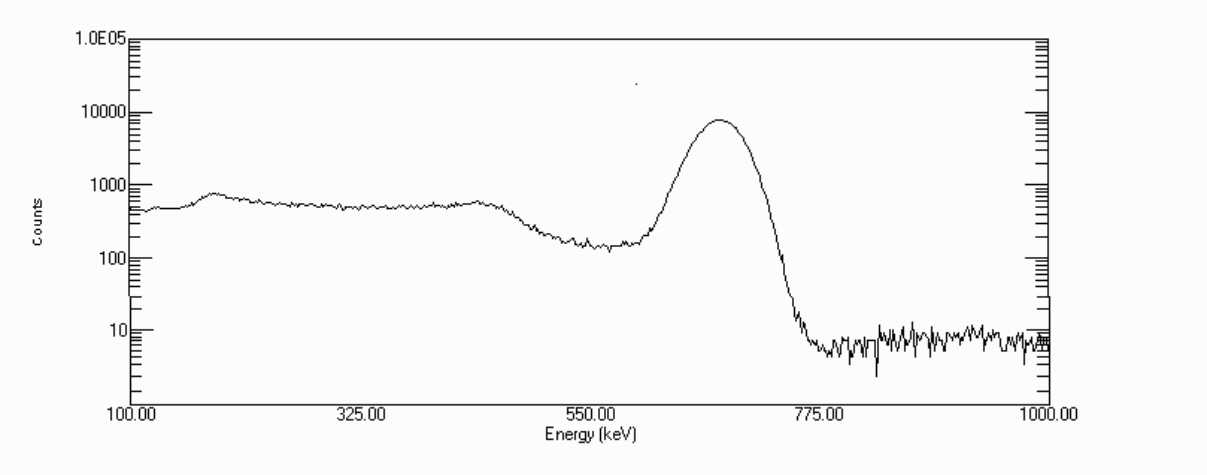


Figure 3. Cs-137 spectrum with pileup rejection

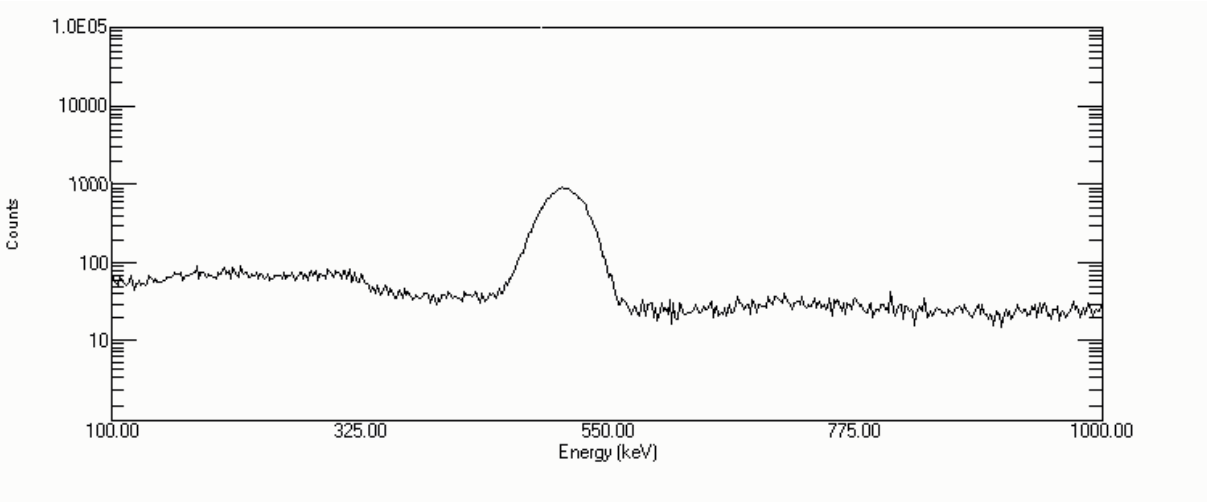


Figure 4. Na-22 Spectrum with pileup rejection

IV- Gamma Ray Algorithms implementation

Finally, the attraction of DSP comes from key advantages such as the following: guaranteed accuracy, perfect reproducibility, no drift in performance with temperature or age, greater flexibility, superior performance, speed and cost effectiveness. The advance in DSP technology came at last with the TMS320C67x floating point DSP processor utilizing the parallelism to increase both the number of instructions and the number of operations in a cycle to meet ever increasing challenges of computationally intensive applications. The exceptional high performance exhibited by this processor motivated us to choose the TMS320C6711 development board whose cost is less than 500$, and with

MATLAB 6.5 and the code composer, we could develop our algorithms in SIMULINK, then compile and download to target board for execution [21,22].

The 3-point, 4-point and 5-point deconvolution algorithms have been implemented as a simple direct structure FIR . The operation shown in figure 4 requires N multiply-accumulates (MACs), for which a DSP is optimised. The algorithms were built and compiled using Simulink and Code composer studio with embedded Texas Instrument C6000 support. Benchmarking shows that the TMS320C6711 DSP processor has the capability to perform 300 million MACs/sec, amounting to approximately 2 MACS/cycle. This implies that the C6711 can perform the 3-point deconvolution in N/2 CPU cycles for each sample, where N is number of filter coefficients.

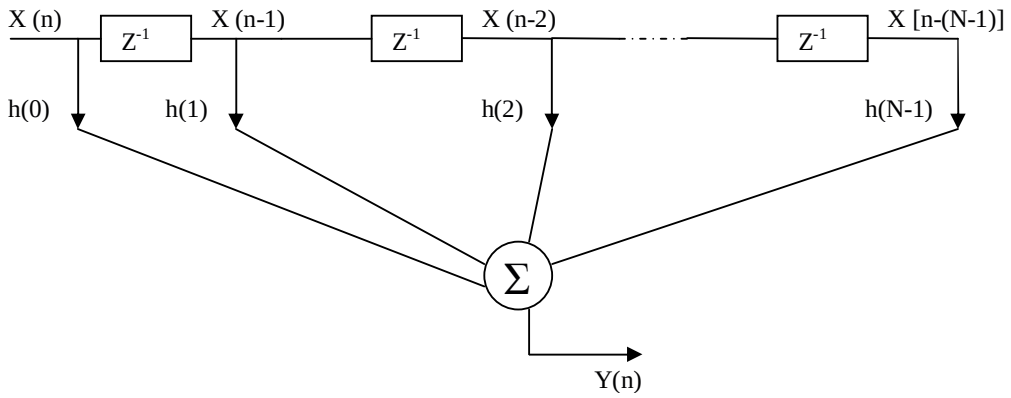


Figure 4. Finite impulse response deconvolution filter flow graph

See table 1 for results of real-time analysis of the Gamma Ray spectroscopic algorithms using the profiler of the Code Composer Studio and the TMS320C6711 DSP evaluation board. The 3-point deconvolution filter took approximately 4µsec in computation time using the TMS320C6711 at 150 MHz. This was the least complex among all other Gamma ray algorithms as expected. Figures 6 and 7 show a comparison among various Gamma ray algorithms in complexity for real multiplications and real additions respectively.

The pulse classification technique has been tested on a TMS320C6711 DSP board. The DSP board used is based on the high performance C6000 floating-point processor running on a 150MHz speed, and delivering up to 600 million floating-point operations per second, see figure 5 for the complete setup including the detector/shaping setup. The pulse classification technique used to classify single from double Gamma pulses, has been written in C code, compiled and run on the target board using the C code composer for real time analysis. Using a number of optimisation stages the classification technique plus the moving average and time series analysis for estimating the pulse parameters took approximately 2 ms, giving a throughput of 486 events per second without losing information. Knowing that for Na-22 Gamma source, the average event rate has been measured to be 398 c/s, leaving an available interval between Gamma Ray events equal to 2.5 ms. Adding to this the moment preserving 3-point deconvolution computation, approximately 52µs, which is still within the available frame. Hence the throughput obtained using the C6000 satisfies the real-time constraint of our spectroscopic system used. Benchmarking results drawn from the code composer studio using the TMS320C6711 evaluation board and summarized in table1, show that the all of the Gamma ray deconvolution algorithms computation times comply with the Gamma ray spectroscopic real time requirement.

Table 1. Comparison of Real-time Computation for Gamma ray algorithms using the TMS320C6711

Algorithm Name	Number of DSP Clock Cycles
3 pointsdeconvolution	637
3 points with moving average filter	6527
3 points with Savitzky Golay filter	7785
4 points Deconvolution	657
4 points with moving average filter	6537
4 points with Savitzky Golay filter	7801
5 points Deconvolution	675
5 points with moving average filter	6613
5 points with Savitzky Golay filter	7817

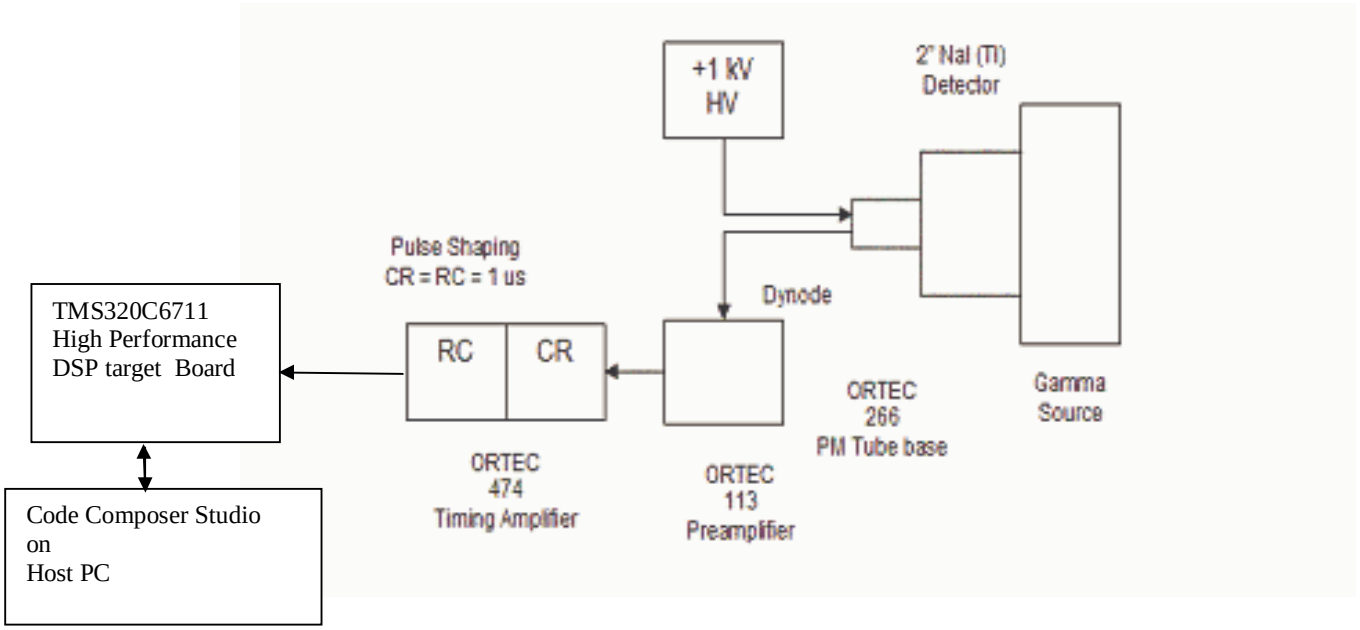


Figure 5. The detector/pulse shaping setup at CAPS for testing Gamma ray peak detection algorithms including the Code Composer Studio and TMS320C6711 board

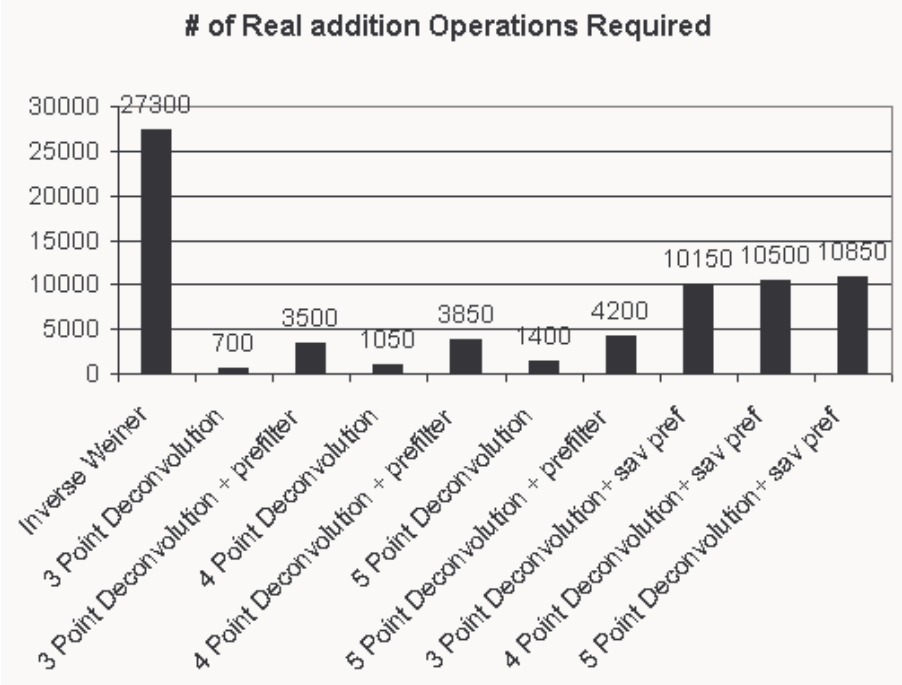


Figure 6. Comparison of Gamma ray algorithm complexity for real additions

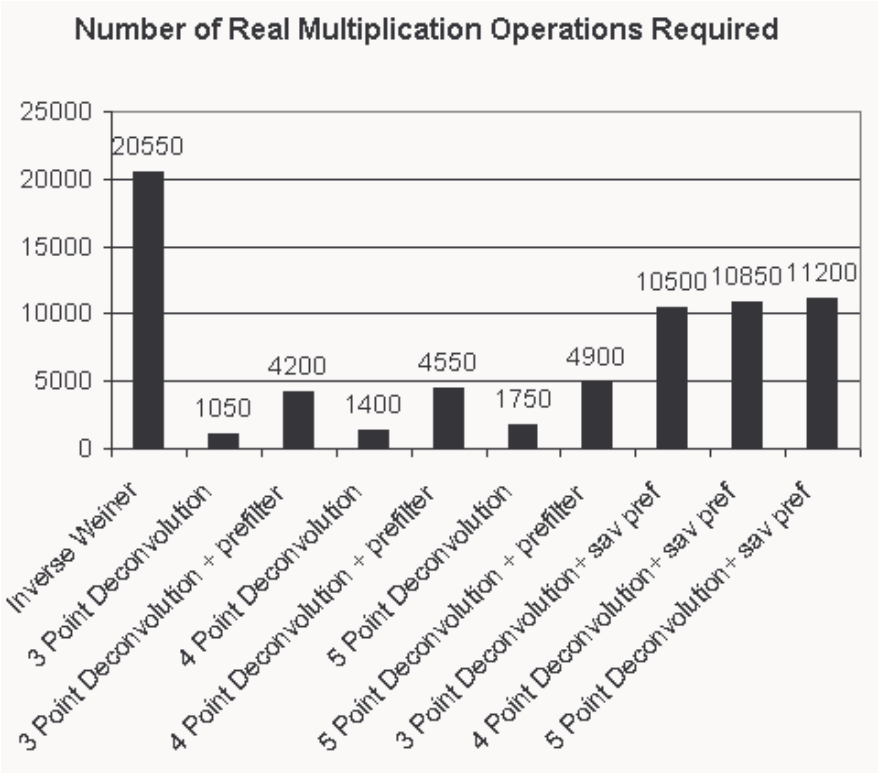


Figure 7. Comparison of Gamma ray algorithm complexity for real multiplications

V. CONCLUSIONS

A number of parameter estimation and digital online peak localisation algorithms are being developed for the purpose of Gamma-ray pulse identification in absence of noise and pileup. A 3-point and 4-point deconvolution techniques have been successfully tested and compared and found to resolve pileup up to approximately 93% or give a pileup suppression of eight fold, with the 3-point filter giving the minimum mean percentage error in peak detection of less than 1% for very low SNR compared to other peak detection algorithms. A window-based pulse classification technique including a peak search routine has been devised to classify single and double pulses and reject peak pileup giving a large reduction in computation time. The pulse classification technique was tested on a TMS320C6711 high performance floating-point DSP processor with execution time down to approximately 2 ms. The pulse classification algorithm in addition to the deconvolution algorithms, were successfully implemented in real-time on the TMS320C6711 board for the case of test Gamma source. Work is under progress to utilise the full performance capabilities of the TMS320C6000 processor to meet all the real-time requirements of Gamma ray Spectroscopic systems, in addition of using the TMS320C6711 as an embedded processor in the Gamma ray peak detection instrumentation devices.

ACKNOWLEDGEMENTS: Support by the EE department University of Bradford UK and Computer Engineering Department and the Research Institute of King Fahd University of Petroleum and Minerals is gratefully acknowledged.

REFERENCES

[1] S.Gadomski, G.Hall, T.Hogh, P.Jalocha, E.Nygard and .Weilhammer, Nucl.Instr. and Meth.., A320, pp. 217- 227.,1992
[2] N.Bingefors, S.Boavier, S.Gadomski, G.Hall, T.S.Hogh, P.Jalocha, H.VD.Lippe, J.Michel, E.Nydard, M.Raymond,A.Rudge, R.Sachdeva, P.Weilhammer and K.Yoshioka, Nucl. Instr. and Meth., A326, pp. 112-119, 1993
[3] Joao M.R. Cardoso, J. Basilio Simoes, Carlos M.B.A. Correia, " A mixed analog-digital pulse spectrometer", Nucl. Instr. And Meth., A422, pp. 400-404, 1999.
[4] R.E.Chrien and R.J.Sutter, Nucl. Instr. and Meth., A249, pp.421-425, 1986
[5] P.W.Nicholson, Nuclear Electronics, John Wiley & Sons, 1974.
[6] John G. Proakis and Dimitris G. Manolakis, Digital Signal Processing, Macmillan Publishing Company, New York, 1992.
[7] Valentine Jordanov and Glenn F. Knoll, IEEE Trans. Nucl. Sci. , Vol. 40, No.4, August 1993.
[8] R. Brenner, H. von der Lippe, J. Michel, E. Nygard, T. Odegaard, N.A. Smith, P. Weilhammer, K. Yoshioka, Nucl. Instr. And Meth. A 339, pp.564-569, 1994.
[9] R. Brenner, J. Kaplon, H. von der Lippe, E. Nygard, S. Roe, P. Weillhammer, K. Yoshioka, Nucl. Instr. And Meths. A 339, pp. 477-484, 1994.
[10] R. Sachdeva, RD20 technical notes, RD20/TN8, September 1992.
[11] G. Hall, RD20 technical notes, IC/HEP/91/4, October 1991.
[12] G. Hall, RD20 technical notes, IC/HEP/92/5, April 1992.
[13] G. Hall, IEEE Transactions on Nuclear Science, VOL. 41, NO.4, August 1994.
[14] Rajiv Sachdeva, Signal Processing Algorithms and Radiation Hard Electronics for the CMS Tracking Detector, PH.D, Imperial College, February 1996.
[15] R. Sachdeva, R. Bellazzini, G. Hall, M. Spezziga, Nucl. Instr. And. Meths., A 348, pp. 378-382, 1994
[16] G. Hall, Imperial College Internal Note IC/HEP/92/5, April 1992.
[17] Timothy Lawrence Bienz, STRANGEONIUM Spectroscopy at 11 GEV/C AND Cherenkov Ring Imaging at the SLD, PHD, Stanford university, July 1990.
[18] W. H. Wong and F. Li, " A Scintillation Detector Signal Processing technique with Active Pileup Prevention for extending Scintillation Count Rates", IEEE Transactions on Nuclear Science, Vol. 45, NO. 3, June 1998.
[19] M.W. Raad, R.E. Abdel-Aal, J.M. Noras, and F. Elguibaly, " Novel Digital Pulse Pileup identification and parameter estimation algorithms in nuclear spectrometry", in the III International Workshop Advances in Signal Processing for NDE of Materials, University Laval, Quebec, Canada, August 5-8, 1997.
[20] Glenn F. Knoll, Radiation Detection & Measurement, Jon Wiley & Sons, 2000.
[21] Nasser Kehtarnavaz and Mansour Keramat, DSP System Design Using The TMS320C6000, Prentice Hall, 2001.
[22] Naim Dahnoun, Digital Signal Processing Implementation Using the TMS320C6000 DSP platform, ation , 2000.

Code generation supported by a pattern-based design methodology

Chisanga Mwelwa[1], Michael J. Pont[1] and David Ward[2]

[1] Embedded Systems Laboratory, University of Leicester,
University Road, Leicester LE1 7RH, UK
{cm55, M.Pont}@le.ac.uk
http://www.le.ac.uk/eg/embedded/

[2] MIRA Ltd, Watling Street, Nuneaton, Warwickshire CV10 0TU, UK
david.ward@mira.co.uk
http://www.mira.co.uk/

Abstract

Automatic code generation from high-level models (such as those based on UML) are becoming increasingly common. Various researchers have sought to carry out a similar code production process beginning with a pattern-based representation: such efforts have not proved overwhelmingly successful. In this paper, we consider some of the challenges involved in creating code from patterns, and argue that the one-pattern-to-many-implementations relationship – which is fundamental to a pattern-based design – makes it very difficult to create general-purpose code-generation tools. We go on to propose a solution using components as an intermediate representation. We illustrate our discussions using a prototype tool that uses this form of representation to support code generation using design patterns in the development of high-reliability embedded systems.

Acknowledgments

This work is supported by the UK Government (through an Industrial CASE award from EPSRC) and by MIRA Ltd.

1. Introduction

Use of embedded processors in passenger cars, mobile telephones, medical equipment, aerospace systems and defence systems is widespread, and even everyday domestic appliances such as dishwashers, televisions, washing machines and video recorders now include at least one such device [Heiner and Thurner, 1998; Mullerburg, 1999; Storey, 1996]. For example, it is estimated that nearly a third of the cost of developing high-class passenger cars is spent on electronics and software, such as airbag control and electronic stability systems [Mullerburg, 1999]. Today there is a trend in the automotive industry for an increasing number of safety-related electronic systems in vehicles that are directly responsible for active and passive vehicle safety. These electronic systems are expected to play a key part in achieving the main transport policy goal of the European Commission, which is to reduce fatalities on European roads by 50% by the year 2010 [eSafety-Working-Group, 2003].

The process of creating such embedded systems will typically involve severe technical, financial and time to market constraints; despite this, an ad hoc approach to the design and implementation is still the norm, with engineers frequently depending on experience and intuition [McGinnity and Maguire, 2001]. As designs grow more widespread and complex, and take on an increasing role in system safety, support for the developers of embedded systems is clearly required [Camposano and Wilberg, 1996].

We have previously argued (see: [Pont and Banner, 2004; Pont, 2001]) that use of appropriate "design patterns" can assist in the creation of reliable embedded systems. Most current work on software design patterns was inspired by the work of Christopher Alexander and his colleagues [Alexander *et al.*, 1977; Alexander, 1979]. Cunningham and Beck used some of Alexander's techniques as the basis for a small pattern language intended to provide guidance to novice Smalltalk programmers [Cunningham and Beck, 1987]. This work was subsequently built upon by Erich Gamma and his colleagues who, in 1994, published an influential book on general-purpose object-oriented software design patterns [Gamma *et al.*, 1995]. Since 1994 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 telecommunication systems [see, for example: Rising, 2001] and systems with hardware constraints [Noble and Weir, 2001].

Our own work focuses on a pattern language for embedded systems development. Our work in this area began in 1996, when we began to assemble a collection of patterns to support the development of embedded systems. The particular focus of

this work was on "time-triggered" systems [e.g. see Kopetz, 1997]. We have now described more than seventy patterns [see: Pont, 2001], which we will refer to in this paper as the "PTTES (Patterns for Time-Triggered Embedded Systems) collection".

In this paper, we consider ways in which software tools can be used to support pattern-based design of embedded systems. Specifically, we consider ways in which patterns can be used to assist in the automatic code generation process.

This paper is organized as follows: in Section 2 we review previous work in automatic code generation. In Section 3, we review previous attempts at pattern-based code generation and in Section 4 we consider a different approach to pattern-based code generation. Section 5 then discusses the techniques used in this different approach to pattern-based code generation. Section 6 gives an overview of our prototype tool that implements the techniques described in Section 5. In Section 7 we discuss the interim conclusions on the current prototype tool and Section 8 goes on to summarise the paper and also draws some conclusions on the current status of our research work.

2. Previous work on automated code generation

It has been argued that the programming language FORTRAN has probably had more impact on the computer field than any other single software development [Sammet, 1981]. Its major technical contribution was to demonstrate that a compiler could produce efficient object code and as a result, it became clear that productivity of programmers using high-level languages could be significantly greater than that of programmers working in assembly languages.

Today code generation is not restricted to programming languages. Indeed, Audsley *et al.* [Audsley *et al.*, 2003] describe automatic code generation as a technology for generating software from design analysis models with little, if any, human intervention. Such an approach holds the promise of eliminating much of the time and effort required to implement safety-critical systems, while at the same time eliminating errors introduced in this stage of development [Whalen and Heimdahl, 1999]. Industries such as aerospace and automotive have made extensive use of automatic code generation tools aimed at control and signal processing systems [Marsh, 2003; O'Halloran, 2000; Schatz *et al.*, 2003]. They are used first to model systems and then to generate code. Originally, code was generated automatically for prototyping platforms or PCs. More recently, code generation has become a more practical means of generating production code for embedded hardware. Hundreds of thousands of cars now rely on code generated using these techniques [Marsh, 2003].

3. Previous work on pattern-based code generation

There have been several attempts to develop tools that support the implementation of design patterns [Budinsky *et al.*, 1996; Viljamaa, 1997; Florijn *et al.*, 1997]. We briefly summarise some of the previous work in this area

Budinsky et al. [Budinsky *et al.*, 1996] describe a tool that generates code from the Gamma object-oriented pattern collection [Gamma *et al.*, 1995]. The tool incorporates an online hypertext version of the Gamma pattern collection that displays sections of the patterns in separate cross-referenced pages. In addition, there is a code generation page associated with every pattern. The user enters the application-specific names and selects implementation-specific trade-offs, and the tool then generates the corresponding C++ declarations and implementations. Essentially the tool takes care of the mundane aspects of pattern implementation so that the developer can focus on optimising the design itself: however, the tool does not support the integration of the patterns into complete systems.

FRED (Framework Editor), a joint project between Tampere University of Technology and the University of Helsinki, is a tool that generates application frameworks from design patterns. FRED[1] supports framework implementation by allowing automatic code generation by systematic guidance for framework adaptation [Viljamaa, 2001]. By incorporating adaptive code generation and documentation, the tool effectively demonstrates a method to adopt as well as systematically specialise a framework, or to re-use architectural standards such as Java Beans.

Each of these tools provides a developer with an automated approach to pattern instantiation. They apply domain specific information, supplied by the user, to instantiate design patterns; code fragments are then generated that have to be manually adapted by the developer to meet the specifications of the software system under development. Crucially, these tools do not directly support code generation from multiple, linked, patterns or pattern languages: the task of linking patterns together is left to the developer.

This is a significant limitation. Indeed, MacDonald *et al*. have argued that design patterns are not widely used as generative constructs that support code re-use because design patterns describe a set of solutions to a family of related design problems and it is difficult to generate a single body of code that adequately solves each problem in the family [MacDonald *et al.*, 2002].

[1] FRED is freely available for download at its homepage: http://practise.cs.tut.fi/fred/

There is perhaps a more fundamental reason why code generation from patterns has not become widespread: this is the fact that a pattern seeks to distil good practice from a wide range of different designs. In turn, a pattern should be widely applicable: by definition, if there is only one possible implementation (or small group of implementations) then we are implementing code from a component, not a pattern.

4. A different approach to pattern-based code generation

The problems identified in the previous section are not insurmountable. One possible solution – and the one which we adopt – is not to attempt a direct translation from patterns to code but, instead, to use a pattern language to assist developers in the assembly of domain-specific software.

This solution is not, itself, unique. A pattern language restricts the domain in which an application is modelled; as a result we are able to capture domain specific knowledge (i.e. special algorithms or structures) in patterns. There are various pattern languages for different domains. Restricted domains allow patterns to be concrete enough that automatic code generation is possible [Heister *et al.*, 1997].

In general, tool-based embedded software development consists of two code generation phases. The first phase consists of source code generation from a domain meta-model – normally modelled using an industry standard such as UML. The second phase is compilation, where a compiler generates object code (i.e. machine instructions) from the source code generated in the first phase. It is the object code, in the form of an executable file, which is programmed onto a processor or microcontroller. Figure 1 gives an overview of this process. Our work focuses on the first phase i.e. generating source code from a domain meta-model, which in our case consists of the PTTES collection.

Please note that we are concerned, at present, with what is sometimes known as "passive" code generation [Herrington, 2003]. A passive code generator produces a set of code which may later be altered or edited by the user.

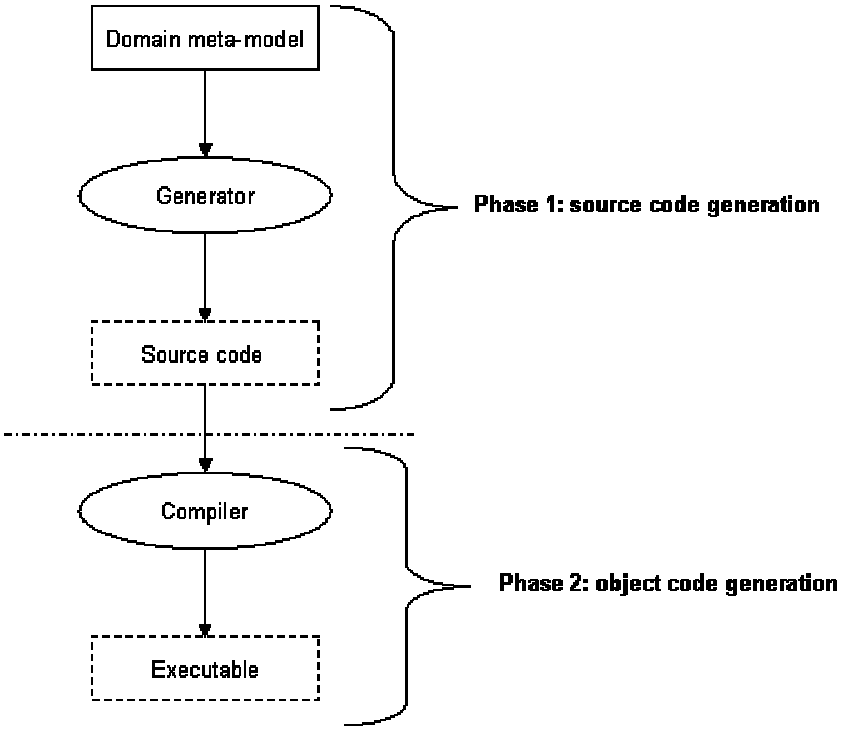


Figure 1: Overview of two typical code generation phases in embedded systems development: source and object code generation

5. Representation of the PTTES collection using XML

An important decision related to code generators is the format used for the representation of the data model and the implementation logic used to generate code. At present, there is a range of commercial tools on the market that base their code generation process on UML data models. UML models are typically represented using the XMI standard (XML Meta-data Interchange) [OMG, 2004]. XMI, an OMG (Object Management Group) specification, is a model-driven XML (eXtensible Mark-up Language) integration framework for defining, interchanging, manipulating and integrating XML data and objects [OMG, 2004]. As our code generation is based on a pattern language as opposed to UML models, we needed to identify an appropriate means of representing the patterns in the tool without compromising the integrity of the pattern format.

As we looked for an appropriate representation, we did not attempt to create a complete, formal, representation of all the patterns in the PTTES collection. Instead, we looked for a way of representing an instance of the patterns in the collection, in a form suitable for use in a tool. In particular we looked at how we could represent the interfaces between patterns in a form suitable for code generation that could support integrating patterns.

To achieve this, we decided to use XML. XML was designed for describing structured data and has established itself as a widely accepted industry standard.

XML structures can be readily accessed from many programming languages using third party components: for example, XPath (XML Path Language) has proven to be a very convenient approach. XPath is an expression language for selecting nodes in an XML tree and is related to XSLT (eXtensible Stylesheet Language Transformation). XSLT is in turn a language used to transform and format XML documents [XML.COM, 2004]. XSLT operates on the tree representation of XML documents while XPath defines the syntax that is used within XSLT scripts to address parts of the XML document tree being transformed or analysed.

XML flexibility and popularity made it an obvious choice for the representation of the PTTES collection in our tool. Consequently, XSLT was the natural choice of approach to generate code from the XML documents.

The essence of XML is the separation of content from presentation. In our case, it is used to separate the core of the patterns from their implementations. For example, Figure 2 to Figure 4 show code snippets from one simple pattern ("Heartbeat LED"), represented in XML. Figure 2 represents the pattern documentation whilst Figure 3 represents the implementation code.

Two points are worth highlighting in this example. In Figure 3 element <sTask> represents tasks called by the scheduler. This element will vary for each implementation depending on the system requirements (for example, task execution times). By contrast, the CDATA section in the XSLT template in Figure 4 represents source code that will remain unchanged for every implementation of this pattern: this is combined with material represented in Figure 3 in order to generate source code.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE PATTERN [
   <!-- Heartbeat_LED Implementation -->
   <!ENTITY pHL-module-Heartbeat_LED SYSTEM "pHLmoduleHeartbeat_LED.xml">
]>
<!-- Contents of Heartbeat LED pattern -->
<PATTERN>
   <CONTEXT><![CDATA[
CONTEXT:
--------
* You are developing (or maintaining) an embedded application based on a microcontroller
* You are programming in C (or a similar language).
* Your application has an architecture based on some form of scheduler.
]]></CONTEXT>
   <PROBLEM><![CDATA[
PROBLEM:
--------
How can you tell, at a glance, if your system is "alive"?
]]></PROBLEM>
```

Figure 2: Heartbeat LED meta-pattern (XML form) representing the pattern documentation

```
<MODULE name="Heartbeat_LED"
            xmlns="http://www.le.ac.uk/eg/cm55/heartbeatled/pHLmoduleHeartbeatLed">
   <SOURCE_FILE name="Heartbeat_LED.c">
      <INIT_TASKS>
         <iTASK>HEARTBEAT_LED_Init</iTASK>
      </INIT_TASKS>
      <SCHEDULER_TASKS>
         <sTASK>HEARTBEAT_LED_Update</sTASK>
      </SCHEDULER_TASKS>
   </SOURCE_FILE>
   <HEADER_FILE name="Heartbeat_LED.h"/>
</MODULE>
```

Figure 3: Heartbeat LED meta-pattern (XML form) contains source code (e.g. the HEARTBEAT_LED_Update task) which needs to be adapted according to the user's requirements (e.g. the frequency of the LED activity).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:output method="text" indent="no"/>
   <xsl:template match="PATTERN">
        <xsl:apply-templates select="IMPLEMENTATION_EXAMPLE/MODULE/SOURCE_FILE"/>
   </xsl:template>

<xsl:template match="IMPLEMENTATION_EXAMPLE/MODULE/SOURCE_FILE">
/*-----------------------------------------------------------------
  This code has been generated by the PRES Tool.
  File: <xsl:value-of select="@name"/>
  Author:
  Last updated:
-----------------------------------------------------------------*/

<![CDATA[
#include "Main.h"
#include "Port.h"
#include "Heartbeat_LED.h"

// ------ Private variable definitions -----------------------------

static bit Heartbeat_led_state_G;
```

Figure 4: The Heartbeat LED template (XSLT format). Such template files represent core implementation code that remains unchanged regardless of the user's requirements

A code generation tool-chain typically consists of a parser, analyser/transformer, and the code generator itself [Voelter, 2003]. Two types of XML parsers exist: the tree-constructing parser and the event-driven parser. Tree constructors parse the XML document in order to construct a tree, and then pass it on for further processing. Event-driven parsers notify the user as soon as desired grammatical constructs are recognised. Despite their large memory footprint, tree-constructing parsers are efficient and easier to use than event-driven parsers as their host code is much less complex than the corresponding host code for an event-driven parser [Herrington, 2003; XML.COM, 2004]. The constructed document tree holds in-memory representation of the parsed XML document. One popular tree-constructing parser

is the DOM (Document Object Model) parser. DOM is a platform- and language-independent interface, that provides a standard model of how the objects in an XML object are put together, and a standard interface for accessing and manipulating these objects and their inter-relationships [XML.COM, 2004; OMG, 2004]. As a result of its widespread use and the level of support it has among software developers, a DOM parser was an appropriate choice for our work.

6. The tool

This section describes the tool and its functionality and then illustrates its use in the creation of a simple example application.

6.1 Overview of the tool

The user interacts with the tool via a GUI-based "wizard". This wizard guides the user in the selection and fine-tuning (adaptation or instantiation) of patterns and at the same time also acts as an interface to a pattern repository on which a code generator is based. The code generator is "fed" by the pattern repository that contains meta-patterns in XML format and XSLT scripts incorporated with XPath facilities written to process the meta-patterns. Each one of these meta-patterns is a pattern from the PTTES collection that has been transformed from its book form to an XML format. Each meta-pattern consists of its description and its components which, to be precise, are implementations of the pattern. These implementations consist of code fragments that always need to be adapted for each implementation for instance global variables and header files. Applying the XSLT scripts to the object tree formed by the DOM parser generates the code; the XSLT scripts filter out the pattern implementations that match the XSLT specification. The XSLT templates also consist of code fragments embedded in them; this is code that will always remain unchanged in each pattern implementation; during the processing this code is combined with the code fragments filtered from the object tree to produce the generated code. Figure 5 provides an overview of the code generation process.

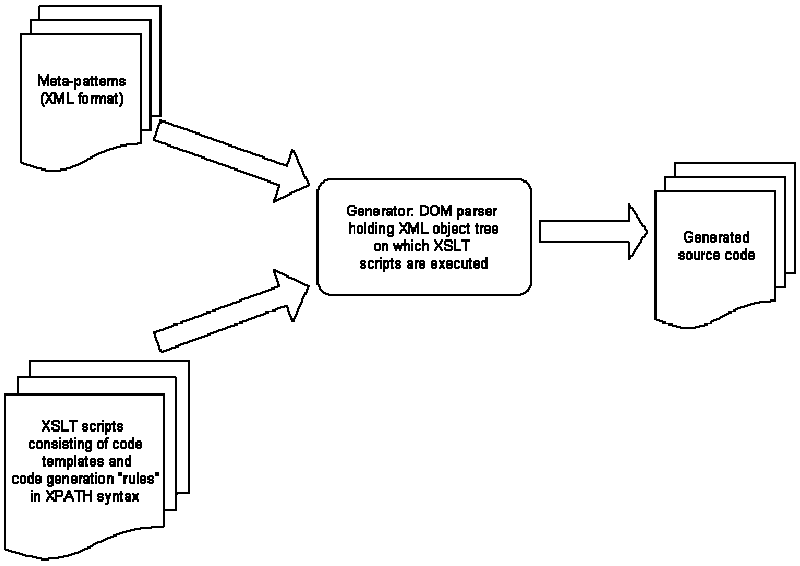


Figure 5: An overview of the code generation process within the tool

6.2 Example of tool application

As described in the previous section, the wizard guides the user through a step-by-step procedure in selecting and integrating patterns in order to generate code. In this section we illustrate how this is carried out.

Figure 6 – Figure 13 show how the Heartbeat LED pattern is integrated with the Co-operative Scheduler pattern in order to create a simple embedded application that flashes an LED on and off at a rate set by the developer.

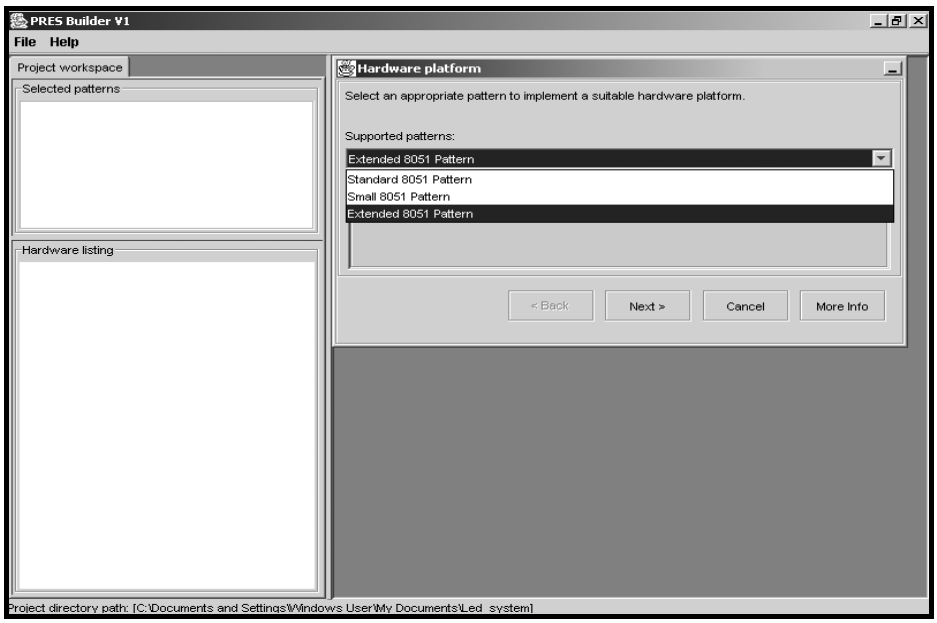


Figure 6: Using the tool: Selection of an appropriate microcontroller hardware platform. The Extended 8051 pattern is selected

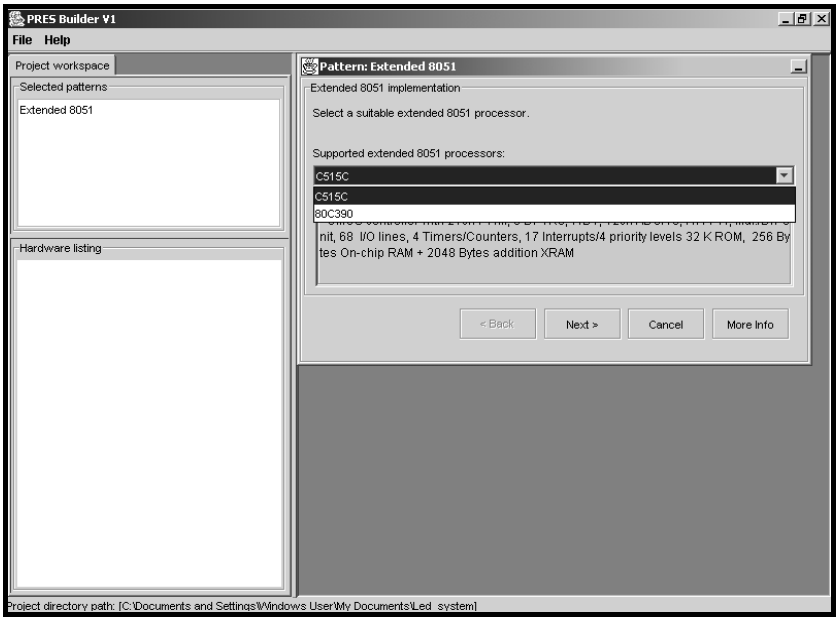


Figure 7: Using the tool: selection of an Extended 8051 microcontroller – this step and that in Figure 6 show the steps taken to build a systems hardware platform using the tool

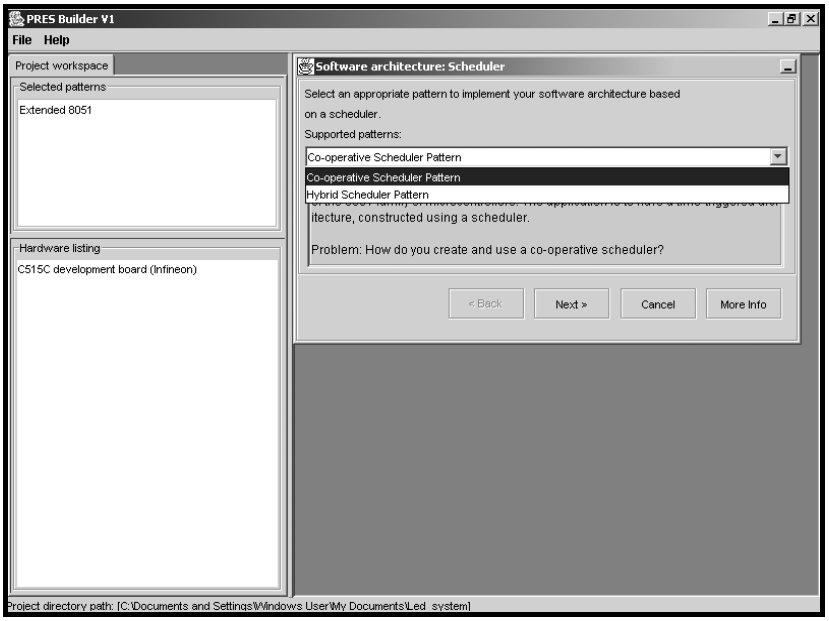


Figure 8: Using the tool: the Co-operative Scheduler pattern is selected as the software platform

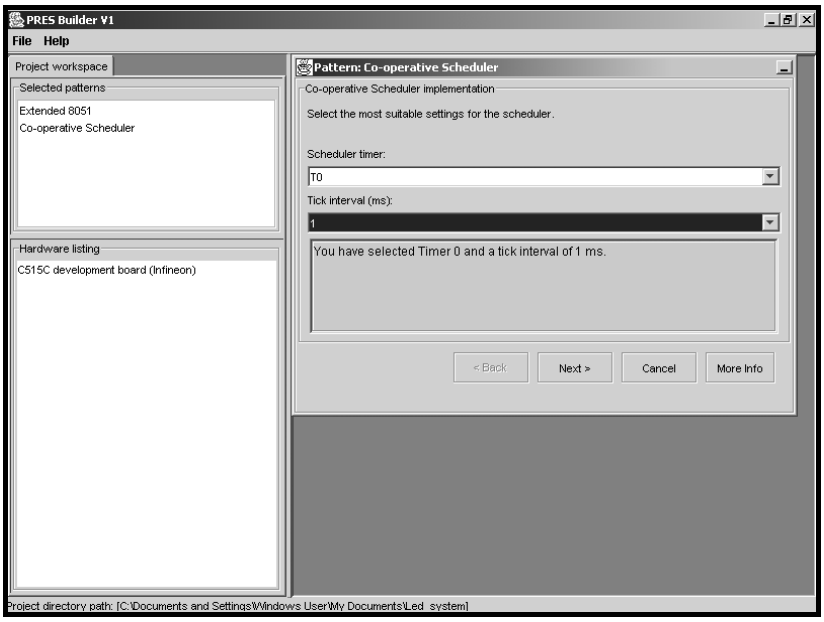


Figure 9: Using the tool: timer and tick interval of the Co-operative Scheduler are set

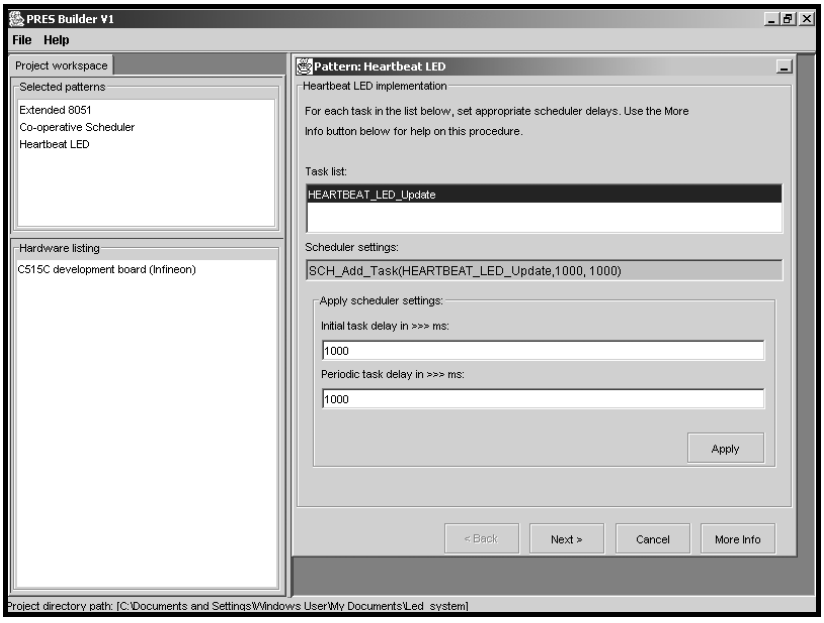


Figure 10: Using the tool: Heartbeat LED pattern is selected and its task interval is set

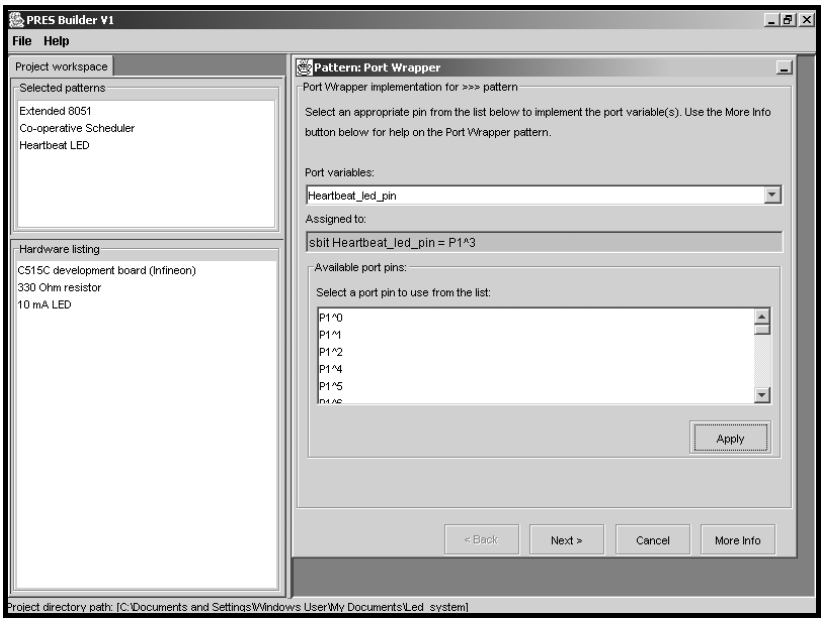


Figure 11: Using the tool: an appropriate hardware port pin on which the LED is to flash is selected

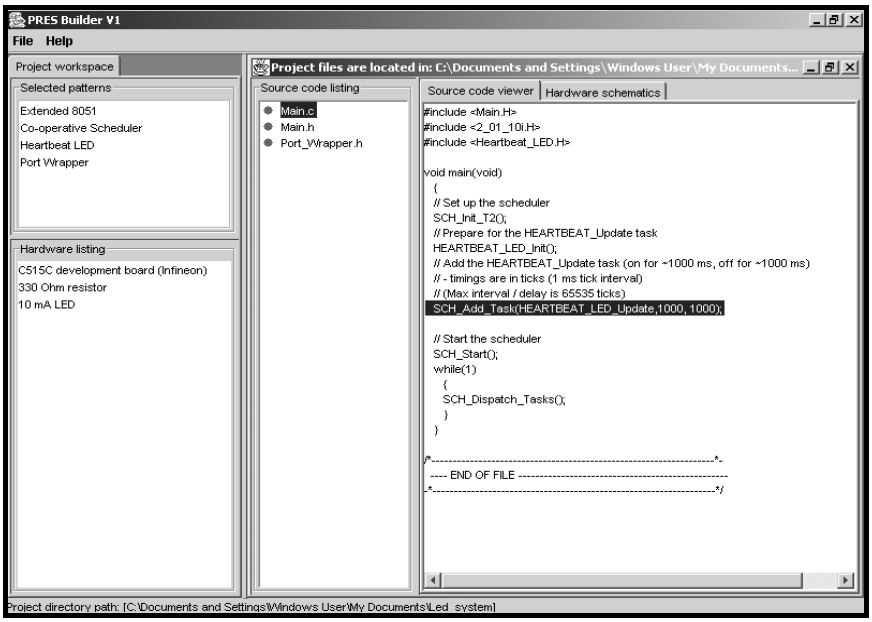


Figure 12: Using the tool: once the user has made all the implementation-specific selections, the code can be generated

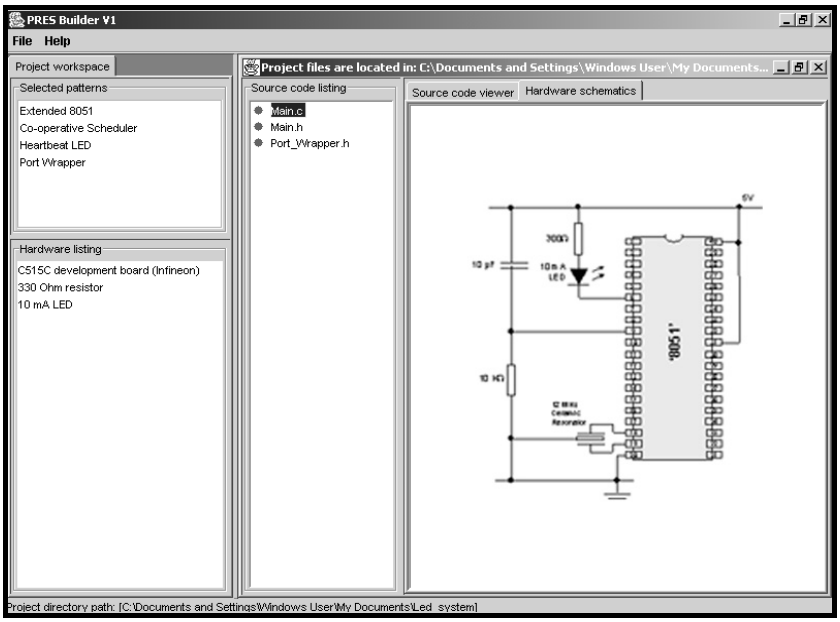


Figure 13: Using the tool: this screen shot shows the potential of the tool in aiding hardware construction in addition to software generation

7. Does the tool work?

At the present stage of the project, a prototype tool has been developed but it has not yet been tested against real-life case studies, a step which is planned for later in the project. However, one basic comparison is already possible. The tool contains a "data logger" that records the time taken to complete a project (Figure 14). This logger was used (in an informal study) to compare the time taken to complete the "Heartbeat LED" project manually, and using the tool. For this application it was found that the average tool user required less than 1 minute to complete this task, while the manual process took most users (at least) 7 minutes.

The "data logger" will allow for comparisons between the "manual" and "automatic" development process in later, more comprehensive, evaluation studies.

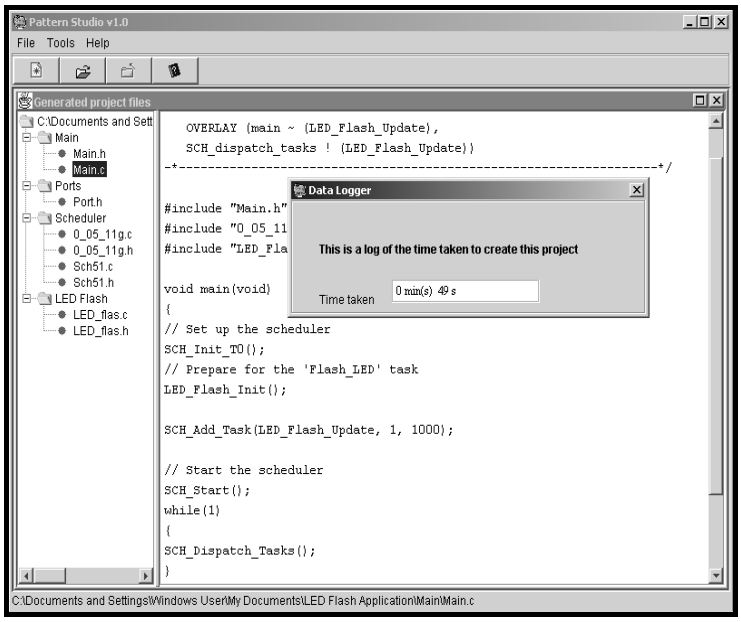


Figure 14: The "Data Logger" displays the time taken for a project to be completed

8. Discussion and conclusions

This paper has provided an overview of the first results from a long-term research project, which is considering the extent to which design patterns can be used to support code generation in embedded systems development. The initial results described here relate only to a prototype tool: nonetheless, the results obtained are encouraging, and suggest that this approach has potential.

The tool is currently being extended to support software maintenance. We are currently looking at how the tool can be used to remove and add different patterns to an application without disrupting the applications architecture. This would be a very powerful feature, as it would enable rapid prototyping and efficient software maintenance in large projects.

9. References

[Alexander, 1979] C. Alexander, "The Timeless Way of Building", *Oxford University Press,* New York, 1979.

[Alexander *et al.*, 1977] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fisksdahl-King and S. Angel, "A Pattern Language", *Oxford University Press,* New York, 1977.

[Audsley *et al.*, 2003] N. Audsley, I. Bate and S. Crook-Dawkins, "Automatic Code Generation for Airborne Systems", *Proceedings of IEEE Aerospace Conference 8th - 15th March 2003,* Big Sky Montana, 2003.

[Budinsky *et al.*, 1996] F. Budinsky, M. Finnie, J. Vlissides and P. Yu, "Automatic Code Generation from Design Patterns", *IBM Systems,* Vol. 35, (2), Pp. 151-171, 1996.

[Camposano and Wilberg, 1996] R. Camposano and J. Wilberg, "Embedded system design", *Design Automation for Embedded Systems,* Vol. 1, (1-2), Pp. 5, 1996.

[Cunningham and Beck, 1987] W. Cunningham and K. Beck, "Using Pattern Languages for Object-Oriented Programs", *OOPSLA'87,* Orlando FL., 1987.

[eSafety-Working-Group, 2003] eSafety-Working-Group, "Information and Communications Technologies for Safe and Intelligent Vehicles", *Commission of the European Communities,* 15th September 2003.

[Florijn *et al.*, 1997] G. Florijn, M. Meijers and P. v. Winsen, "Tool Support for Object-Oriented Patterns", *ECOOP'97,* Finland, 1997.

[Gamma *et al.*, 1995] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison-Wesley,* 1995.

[Heiner and Thurner, 1998] G. Heiner and T. Thurner, "Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems", *28th Annual Symposium on Fault Tolerant Computing,* IEEE Computer Society Press, Munich, Germany, 1998.

[Heister *et al.*, 1997] F. Heister, J. P. Riegel, M. Schütze, S. Schulz and G. Zimmermann, "Pattern-Based Code Generation for Well-Defined Application Domains", *EuroPLoP '97,* Kloster Irsee, Germany, 1997.

[Herrington, 2003] J. Herrington, "Code Generation in Action", *Manning Publications Co.,* Greenwich, USA, 2003.

[Kopetz, 1997] H. Kopetz, "Real-time Systems: Design Principles for Distributed Embedded Applications", *Kluwer Academic,* New York, 1997.

[MacDonald *et al.*, 2002] S. MacDonald, D. Szafron, J. Schaeffer, J. Anvik, S. Bromling and K. Tan, "Generative Design Patterns", *17th IEEE International Conference on Automated Software Engineering (ASE'02)*
September 23 - 27, 2002, IEEE Computer Society, Edinburgh, UK, 2002.

[Marsh, 2003] P. Marsh, "Models of Control", *The IEE: Electronics Systems and Software,* Vol. 1, (6), Pp. 16-19, 2003.

[McGinnity and Maguire, 2001] T. M. McGinnity and L. P. Maguire, "A CASE-Tool Oriented Approach for Embedded Systems Design", *Microprocessors and Microsystems,* Vol. 24, Pp. 493-499, 2001.

[Mullerburg, 1999] M. Mullerburg, "Software Intensive Embedded Systems", *Information and Software Technology,* Vol. 41, Pp. 979-984, 1999.

[Noble and Weir, 2001] J. Noble and C. Weir, "Small Memory Software", *Addison Wesley,* 2001.

[O'Halloran, 2000] C. O'Halloran, "Issues for the Automatic Generation of Safety Critical Software", *The Fifteenth IEEE International Conference on Automated Software Engineering, 11-15 September 2000,* Grenoble, France, 2000.

[OMG, 2004] OMG, "Object Management Group", http://www.omg.org, 2004.

[Pont, 2001] M. J. Pont, "Patterns for Time-Triggered Embedded Systems", *Addison-Wesley,* 2001.

[Pont and Banner, 2004] M. J. Pont and M. P. Banner, "Designing Embedded Systems Using Patterns: A Case Study", *Journal of Systems and Software,* Vol. 71, (3), Pp. 201-213, 2004.

[Rising, 2001] L. Rising, "Design Patterns in Communications Software", *Oxford University Press,* 2001.

[Sammet, 1981] J. E. Sammet, "History of IBM's Technical Contributions to High Level Programming Languages", *IBM Journal of Research and Development,* Vol. 25, (5), Pp. 520-534, 1981.

[Schatz *et al.*, 2003] B. Schatz, T. Hain, F. Houdek, W. Prenninger, M. Rappl, J. Romberg, O. Slotosch, M. Strecker and A. Wisspeintner, "CASE Tools for Embedded Systems", *Technical University of Munich,* July 2003.

[Storey, 1996] N. Storey, "Safety-Critical Systems", *Addison-Wesley,* 1996.

[Viljamaa, 2001] A. Viljamaa, "Pattern-Based Framework Annotation & Adaptation", *Department of Computer Science,* University of Helsinki, Helsinki, 2001.

[Viljamaa, 1997] J. Viljamaa, "Tools Supporting the Use of Design Patterns in Frameworks", *University of Helsinki,* March 1997.

[Voelter, 2003] M. Voelter, "A Catalog of Patterns for Program Generation", *Eighth European Conference on Pattern Languages of Programs,* Irsee, Germany, 2003.

[Whalen and Heimdahl, 1999] M. W. Whalen and M. P. E. Heimdahl, "On the Requirements of High-Integrity Code Generation", *Proceedings of the 4th High Assurance in Systems Engineering Workshop, November 1999,* Washington DC, 1999.

[XML.COM, 2004] XML.COM, "XML From the Inside Out", www.xml.com, 2004.

Using simulation to support the design of distributed embedded control systems: A case study

Devaraj Ayavoo[1], Michael J. Pont[1] and Stephen Parker[2]

[1]Embedded Systems Laboratory, University Of Leicester, University Road, Leicester LE1 7RH, UK.
Tel: 0116 2522873 Fax: 0116 252 2619
http://www.le.ac.uk/embedded/

[2]Pi Technology, Milton Hall, Ely Road, Milton, Cambridge CB4 6WZ, UK

Abstract

Distributed embedded control systems are becoming increasingly common and increasingly complex. For non-trivial designs, the number of possible system permutations is enormous and - given time-to-market constraints - the development of a comprehensive suite of prototypes is only rarely practical. As an alternative, this paper considers the effectiveness and practicality of employing a general-purpose simulator during the design process. Using a representative case study, the paper demonstrates that a simple simulation can provide a large amount of useful information at design time. However, the effort involved in generating the necessary simulations is significant. Future work will therefore focus on techniques to reduce the effort of developing simulation models.

Acknowledgements

This work is supported by an ORS award (to DA) from the UK Government (Department for Education and Skills), and by Pi Technology.

1. Introduction

Distributed embedded control systems are becoming increasingly common (Lewis, 2000), and increasingly complex (Montresor *et al*., 2003). 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, 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 (Tanenbaum, 1995) topology. 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, 2000). These individual processors may, for example, be C16x (Siemens, 1996), ARM (ARM, 2001), MPC555 (Bannatyne, 2003) or 8051 (Pont, 2001).

Our particular concern in this paper is with the development of embedded systems for automotive X-by-Wire applications. When designing such systems, timings delays (or variations) can have an impact on the system performance (Lonn and Axelsson, 1999). Some of the key timing factors that can cause degradations in the control systems have been described by Sandfridson (2000), where the effects of time variability in control delays and control task periods were discussed.

Overall, the number of possible system designs is enormous. If we want to understand the system timing behaviour – or identify the "best" system architecture - then prototyping even a small subset of the different systems is impractical: an alternative approach is therefore required (Thane, 2000).

In order to support the design of safe and reliable distributed embedded systems, a range of modelling and simulation tools have been developed. For example, El-khoury and Törngren (2001) described a toolset which integrates the modelling of schedulers and distributed systems with models of control system performance. Another tool (called RTSIM) was described by Palopoli *et al*. (2001) to help engineers to develop real-time distributed embedded control systems. Similarly 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). All of these tools are intended to help the developer of a distributed (control) system understand the product behaviour early in the design cycle.

The aim of the study described in this paper was to begin to understand the effectiveness of a simulation-based design approach when developing X-by-Wire applications. More specifically, we wanted to do two things. First, we wished to examine how effective simulation would be when comparing some of the different software architectures that might be used to implement distributed embedded control systems. Second, we wished to consider how much effort would be involved in creating such simulations.

To do this, a single representative case study was used in the project report here. This case study (described in detail in Section 2) represents a cruise control system (CCS), for use in a passenger car. In this case study, the focus was on the following aspects of the system performance:

- Control performance
  This is the measure of the effectiveness of the CCS in maintaining the vehicle speed at the desired value.
- Event response time
  In the CCS, the event response time is a measure of 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
  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.
- Control jitter
  This is a measure of the variations in start times of the periodic control task: for example, if the 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. An interference delay is the longest time that all higher priority tasks can be queued and executed before task x is finally executed.

It is to be expected that all of these measures will be affected to a greater or lesser extent, by the type of scheduling policies used (for example, time-triggered or event-triggered) as well as the choice of network protocols (for example, time-division multiple access, or fixed-priority). These different options were compared in a range of simulations.

The remainder of this paper is organized as follows. Section 2 provides a more detailed description of the case study. Section 3 discusses the design and implementation of the control system that is used in this case study. Section 4 goes on to compare the results obtained using the simulator with the results from a hardware testbed. Section 5 presents the conclusion and discusses some plans for future work in this area.

2. Overview of the case study

As noted in the introduction, the work described in the present paper is centred on a case study. This involved the design of a simple automotive cruise control system (CCS).

2.1 Specifications of the CCS

An automotive CCS is intended to provide the driver with an option of maintaining the vehicle at a desired speed without further intervention.

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 case study, the specification of the CCS was simplified such that the vehicle was assumed to be always in “cruise” mode. When cruising a ‘speed dial’ was provided to allow the driver to change the speed.

2.2 The car model

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

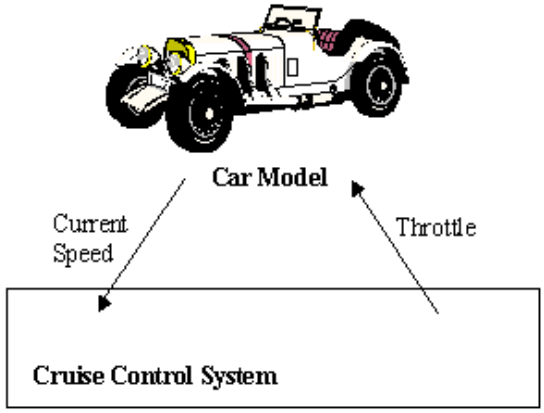


Figure 1 - A basic cruise control system

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

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

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

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

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. The engine torque is assumed to be constant over the speed range. These models can clearly be made more realistic, but - for the purpose of this study - this simplified model was sufficient.

2.3 The simulator

As briefly explained in Section 1, several modelling and simulation tools have been developed to help understand the behaviour and characteristics of distributed embedded systems early in the design cycle. In the study described in this paper, we employed a tool called "TrueTime" (Cervin *et al*., 2003).

The TrueTime simulator was developed by the Department of Automatic Control, Lund Institute of Technology. TrueTime is intended to facilitate the co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics (Cervin *et al*., 2003). 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 external and internal interrupts with multiple analogue-to-digital and digital-to-analogue channels. Multiple network communication blocks can be created, including CSMA/CA (Carrier Sense Multiple Access with

Collision Avoidance) which is similar in behaviour to the Controller Area Network (CAN) protocol (Kopetz, 1998), and TDMA (Time Division Multiple Access) which is similar to the Time-Triggered Protocol (Kopetz, 1998).

The TrueTime simulator was used in this study 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 Dorf and Bishop, 1998; Dutton *et al*., 1997; Sampson *et al*., 1999; TTTech, 2002). 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.

2.4 The hardware-in-the-loop testbed

In order to investigate the accuracy of the simulation process, a hardware-in-the-loop (HIL) testbed was used to implement each of the simulated systems, using appropriate processor hardware (in this case, a network of Infineon C167CS/CR microcontrollers). CAN was used as the network protocol to connect the distributed system. As with the software-only simulation, the HIL testbed made use of the environment model described in Section 2.2.

To allow measurements from this testbed, a monitoring device was constructed using an additional 8051 processor, with a serial (RS-232) link to a desktop PC.

3. Design options considered in this study

3.1. Distributed system design

The process of control can be divided into three main operations: sampling, control and actuation. In the first operation, data are sampled from the plant. In the second operation, these data are processed using an appropriate control algorithm. In the third operation, 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 El-khoury and Törngren, 2001; Lonn and Axelsson, 1999).

In the present case study, the CCS was designed to operate as a distributed system using two nodes: a sampler node and a controller & actuator (CA) node (Figure 2).

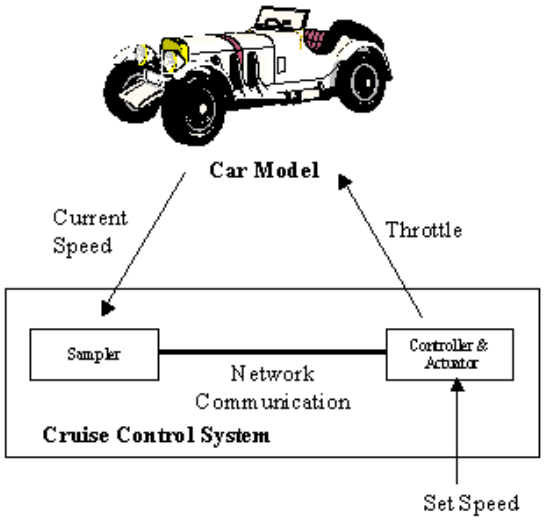


Figure 2 - A distributed cruise control system simulation

The sampler node was used to calculate the vehicle speed: this was 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 was also performed on this node.

The calculated car speed was then sent over a network to the CA node (Figure 1). On the CA node, a 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).

3.2. Error detection and management

The CCS is a safety-related system, and any software or hardware errors must therefore be detected and handled in an appropriate manner. For example, if a sensor failure occurs and no error management is performed, the result may be a cruise speed 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 should be detected, and that the vehicle would 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 could also be applied: for example, control could be returned to the driver, or a backup sensor could be deployed. However, the type of fail-safe mechanism used has no bearing on the present study.

3.3. Description of the tasks

Embedded software systems are often designed and implemented as a collection of communicating tasks (e.g. Nissanke, 1997; Shaw, 2001), and in this case each of the nodes was assigned a set of tasks to carry out. Table 1 below indicates all the tasks initial arrival, period and execution times on the sensor and the CA nodes.

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
			Total	***53***

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
			Total	***212***

Table 1 - Task initial arrival, period and execution times

3.4. Possible system designs

Choosing an appropriate design for the described CCS is not a trivial process, even for such a simple two-node system. As pointed out earlier, the software architecture for each node, and the network protocol, needs to be appropriate if the desired performance is to be obtained.

The various possible system architectures in this type of embedded design are often characterized in terms of the tasks that are to be performed. 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-preemptive). 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.

In the study described in this paper, four different systems were designed, implemented and compared (Table 2). In all cases, the nodes were linked using a CAN bus running at 333.3 kbits/s.

	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

Table 2 - Combinations of possible systems

In the "T-T-T" system, the scheduling on both nodes was time-triggered. 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.

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.

4. Results

In this section, the results from the software-only simulator and the HIL testbed are reported and compared.

4.1. Control performance

The control performance of all systems was found to be very similar. As an example, Figure 3 shows the results from the "E-E-E" system.

The two lines on the graph represents the performance obtained using the software-only simulator (solid line) and the HIL testbed (dotted line). 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 that the CCS was able to follow the desired (set) speed in both the simulator and the HIL testbed.

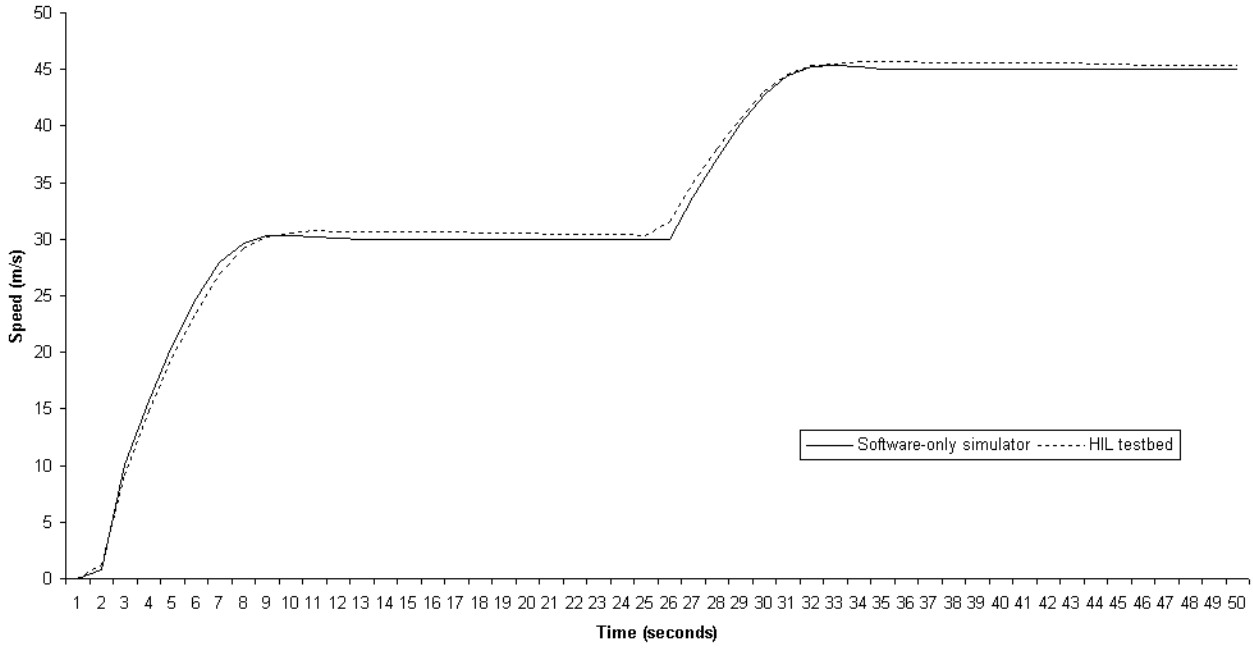


Figure 3 - Control performance using the software-only simulator and HIL testbed

4.2. Event response time

The event response times used in this study were defined in Section 1. Table 3 shows the recorded event response times for the simulation and HIL systems.

	Minimum (in µs)		Maximum (in µs)	
System	**HIL**	**Simulation**	**HIL**	**Simulation**
"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

Table 3 - Comparison of software-only simulation results and the HIL testbed for an event message response time

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 is due to the fact that the different boards (in the HIL design) have independent crystal oscillators, which move out of step. This possibility is not taken into account in the simulator used in this study.

To illustrate the underlying problem, Figure 4 shows the results recorded from a simple HIL implementation that sends a signal from the Sampler to the CA node every 2 seconds. On the nodes, the sampling and actuation tasks are (both) scheduled every 5ms.

In this case, the communication delay is approximately 390µs: this represents the minimum possible response time. The maximum response time is 390µs plus the 5ms (actuation) task period. The recorded results lie in this range of values, as illustrated in Figure 4.

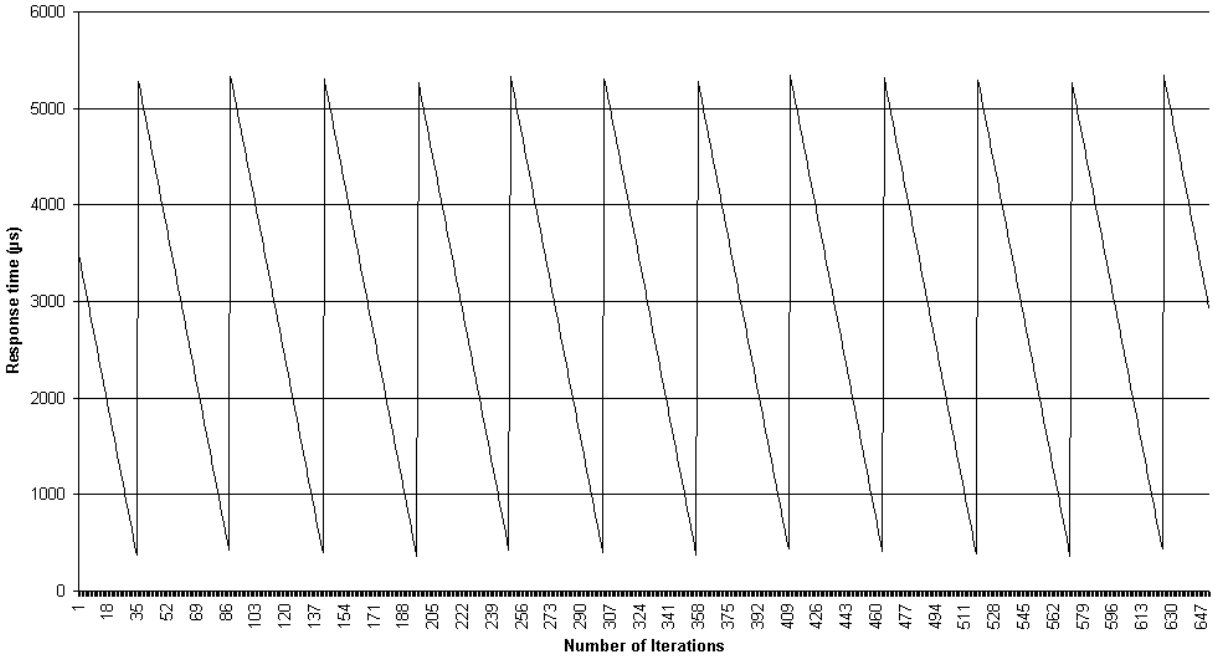


Figure 4 - Drift in the signal response time for "T-E-T" system using the HIL testbed

4.3. Control delay

Table 4 displays the control delay for the two systems that 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 Controller & Actuator node).

	Control Delay	
TTT System	**HIL**	**Simulation**
Minimum (µs)	1388	1373
Maximum (µs)	1398	1373
Average (µs)	1393	1373
Std Div	2.329284	0.00
ETE System		
Minimum (µs)	1390	1381
Maximum (µs)	1400	1381
Average (µs)	1395	1381
Std Div	2.256839	0.00

Table 4 - Comparison of control delay between the software-only simulator and the HIL testbed

Once again, the simulation results for the xTx systems follow those from the HIL testbed closely (with an average error of less than 2%).

The results obtained for the xEx system was similar to the characteristic as shown in Figure 4. This is – again - due to the fact that the two boards (in the HIL design) have independent crystal oscillators on the processors which move out of step.

4.4. Control jitter

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

	Percentage of average deviation between Simulation and HIL (%)	
	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

Table 5 – The percentage of deviation between the software-only simulator and the HIL testbed for the periodic control tasks

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

5. Discussion and conclusions

The results obtained in this paper show a close correspondence between the simulated behaviour of the CCS and the measurements obtained from the corresponding HIL testbed. Of course, the simulator is imperfect: for example, the presence of independent crystal oscillators in the two processor nodes was not taken into account. However, the simulation could however be extended to incorporate a realistic frequency drift (for example, see Törngren *et al*., 2001).

Overall, the results obtained in the case study suggest that simulation is an effective way of predicting the performance of distributed embedded control systems. This result is consistent with findings in previous studies: for example, Redell (1998) noted that modelling of distributed systems should be performed to understand the true behaviour of the implemented system.

However, it must also be noted that the effort involved in creating each simulation – while certainly less than that involved in the creation of the HIL prototypes – was nonetheless considerable. Specifically, even the simple CCS considered here required the creation of around 350 lines of code per system. A similar finding was reported recently by Castelpietra *et al*. (2002) where the authors described the development of simulation models as time consuming.

We are currently investigating techniques that can be used to reduce the effort involved in the creation of a wide range of simulations.

6. References

ARM, ARM7TDMI, (2001), Technical Reference Manual.

Bannatyne R., (2003), "Microcontrollers For Automobiles", Transportation Systems Group, Motorola Inc., Micro Control Journal.

Castelpietra P., Song Y. Q., Lion F. S., Attia M., (2002), Analysis and Simulation Methods for Performaance Evaluation of a Multiple Networks Embedded Architecture, IEEE Transaction on Industrial Electronics, Vol. 49, No. 6.

Cervin A., Henriksson D., Lincoln B., Eker J., Årzén K., (2003), How Does control Timing Affect Performance? – Analysis And Simulation Of Timing Using Jitterbug And TrueTime, IEEE Control Systems Journal (Vol. 23).

Dorf D., Bishop R., (1998), Modern Control Systems, Addison Wesley.

Dutton K., Thompson S., Barraclough B., (1997), The Art of Control Engineering, Addison Wesley.

Eker J., Cervin A., (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.

El-khoury J., Törngren M., (2001), Towards A Toolset For Architectural Design Of Distributed Real-Time Control Systems, IEEE Real-Time Symposium.

Hartwich F., Muller B., Fuhrer T., Hugel R., Bosh R. GmbH, (2000), CAN Networks with Time-Triggered Communication, 7th international CAN Conference.

Hartwich F., Muller B., Fuhrer T., Hugel R., Bosh R. GmbH, (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., (1998), A Comparison of CAN and TTP, Proceedings Of The IFAC Distributed Computer Systems Workshop.

Kopetz H., (2001), A Comparison of TTP/C and FlexRay, Research Report 10/2001, Vienna University of Technology.

Lewis D., (2001), Fundamentals of Embedded Software, Prentice Hall.

Lonn H., Axelsson J., (1999), A Comparison Of Fixed-Priority And Static Cyclic Scheduling For Distributed Automotive Control Application, EuroMicro Conference on Real-Time Systems.

Montresor A., Meling H., Babaoğlu Ö., (2003), Towards Self-Organizing, Self-Repairing and Resilient Distributed Systems.

Nissanke N., (1997), Realtime Systems, Prentice Hall.

Palopoli L., Lipari G., Abeni L., Natale M. D., Ancilotti P., Conticelli F., (2001), A Tool For Simulation And Fast Prototyping Of Embedded Control Systems, Proceedings of LCTES01.

Pont M. J., (2001), Patterns For Time Triggered Embedded Systems, Addison Wesley.

Rajnak A., Ramnerfors M., (2002), The Volcano Communication Concept, International Congress on Transportation Electronics.

Redell O., (1998), Modelling of Distributed Real-Time Control Systems An Approach for Design and Early Analysis, Licentiate Thesis, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Sweden.

Sampson D. J. M., McKevitt G., Cebon D., (1999), The Development of Active Roll Control for Heavy Vehicles, Proc. 16th IAVSD Symposium on the Dynamics of Vehicles on Roads and Tracks.

Sandfridson M., (2000), Timing Problems in Distributed Real-Time Computer Control Systems, Technical report, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Sweden.

Shaw A. C., (2001), Real-Time Systems And Software, Wiley.

Siemens AG, (1996), C167 Derivatives – User's manual Version 2.0.

Specks J. W., 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.

Thane H., (2000), Monitoring, Testing and Debugging of Distributed Real-Time Systems, Doctoral Thesis, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Sweden.

Törngren M., El-khoury J., Sandfridson M., Redell O., (2001), Modelling And Simulation Of Embedded Computer Control Systems: Problem Formulation, Technical Report, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Sweden.

TTTech, (2002), Utilizing TTP Tools in By-Wire Prototype Projects, TTTech Computertechnik AG.

Energy Efficient Functional Unit For A Compute-Intensive Asynchronous Parallel DSP

W. Suntiamorntut, L.E.M. Brackenbury
APT Group, Department of Computer Science, The University of Manchester,
Oxford Road, Manchester, M13 9PL, UK

Abstract

A functional unit (FU) that is well suited to high computational-intensive DSP systems, and combines several techniques to achieve high energy efficiency is presented. This FU consists of a datapath enabling parallel operations within it. A user configurable RAM memory specifies the operations required and gives the user and system programmer great flexibility in executing algorithms. Not only multiply accumulator operations can be performed by the FU, but other necessary instructions of a general DSP such as the hamming distance are also implemented. The operation flow is controlled using a self-timed circuit style to further reduce power and spread the circuit switching. Post-layout simulation of our full-custom FU implemented on 0.18μm operating at 1.8 V with an area 0.36 mm^2 shows a large energy improvement by a factor of 10 compared to the original design whilst performance results show that using four-way parallelism DSP yields a high throughout rate of equivalent to 800 MHz.

1. Introduction

As is well-known, the capabilities of computation in portable applications has been increasing exponentially. However, the intensive and continuous computing of hand-held computers and other portable devices are restricted by the source of power. There is no equivalent of Moore's Law in the case of battery technology and only a 20% improvement in capacity of battery technology is expected over the next 10 years [1]. Reiner Hartenstein reported a technology trend graph as shown in Figure1[2]. It is clear that the energy density of existing battery technologies are far from what is needed. Hence, an energy efficient design becomes vital.

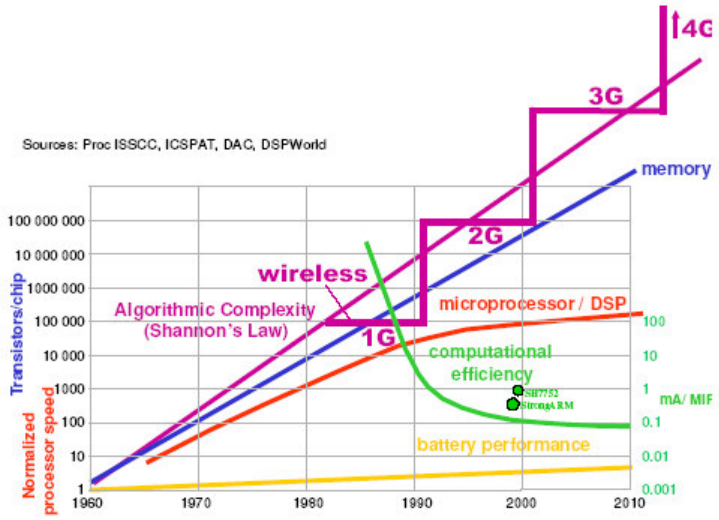


Figure1 Technology trends

Digital Signal Processors (DSPs) have been developed for wireless applications such as mobile handsets. Mostly, mobile phones are driven by cellular standards. In the first generation, many types of filter were expected to run on the DSPs. However, a modern mobile phone handles many more applications such as video decoding, data processing and speech recognition. Multiple standards are also needed in one device. Therefore, the trend for DSP architectures is parallelism; exploiting more than one processing unit gains high throughout. Whilst the area is increased by employing multiple computing units, the energy consumption is decreased by scaling down the supply voltage. However, there is a question as to whether large voltage scaling improves the energy efficiency since theoretically, the sub-threshold supply voltage leakage energy becomes dominant. For moderate voltage scaling, the logic depth, amount of switching and the workload factor of the circuit determine

the energy efficiency. In [3], it is reported that the operating voltage should be scaled to approximately 30% of the maximum supply voltage under typical workload requirements.

If the energy-delay product metric is used as the basis of comparison between designs, then the overall energy efficiency can be exponentially improved in a DSP with parallel FUs since each FU can be operated at lower throughput. This is because when the energy-delay product is taken into account, it means each component uses the smallest energy source per operation. When the speed has been slowed down, each component will dissipate less power. Meanwhile, the overall throughout of the system is still maintained because of the parallel architecture. Therefore, the system can gain an energy efficiency improvement merely by introducing parallelism.

An alternative approach to make an energy efficient system is to structure algorithms to fit the available hardware resource in a DSP. This usually has restrictions with the existing hardware architecture. Therefore, at this level, system may not gain a big improvement. When several energy efficient design techniques are combined, the system will give a large energy improvement when running on a powerful architecture.

Another requirement of a DSP is flexibility. In recent years, many research works [4-7] propose reconfigurable digital signal processors with designs implemented on Field Programmable Gate Array (FPGA) technology. However, the power dissipation is still relative high. New generation FPGA includes specific arithmetic units, accelerators, IP cores are these provides the speed and flexibility but still tend to be high power. The research challenge is to develop a design for both energy efficiency and flexibility. Portable systems need a processing unit that gives them the lowest energy consumption but at the same time must provide enough flexibility to the user and programmer. The functional unit presented here is an energy efficient and flexible component for an asynchronous parallel DSP performing computationally intense algorithms. A configuration memory has been attached to each energy efficient FU in our design for maximum. Internally, the design combines several energy efficient circuit/logic design techniques. The detail of our DSP architecture will be described briefly in the next section followed by the instruction set for our system and an example of a simple 2-D discrete cosine transform (DCT) to demonstrate the exploitation of parallelism in our DSP. Our FU architecture including arithmetic, logical unit and the configuration memory are described in section 3. The low power circuits used in the datapath are described in section 4. Simulation results from the full-custom layout are presented in section 5 demonstrating the improvements to energy achieved and also contains some concluding remarks.

2. Digital Signal Processor

A. Architecture Overview

Our FU has been designed and implemented for a parallel asynchronous DSP named CADRE [8]. CADRE was proposed to be a minimum power consumption DSP whilst meeting the performance requirements of next-generation cellular phones. However, the simulation results show that the power dissipation is still on the high side. In [8], the power dissipation of CADRE was analyzed and approximately 50% of the overall power consumption was found to be dissipated in the FU. Therefore, the new FU in this research will replace on the original design. The original philosophy of the architecture is described to help the reader understand our FU easily.

CADRE was implemented by exploiting four-way parallelism as this appears to be optimal for power reduction, Chandrakasan and Brodersen [9]. This is based on the premise that area can be traded for increased speed because silicon area is rapidly becoming less expensive. Most of the DSP activity can be characterized by frequent repetition of fixed instruction sequences. So, the instruction encoding which determines the selection and passage of data for each operation can be predetermined and stored in advance in a configurable memory which is located locally to each FU. These encodings can then be recalled with a compressed instruction. Because the configuration memories are RAMs, this allows reconfiguration at any point in execution. In addition, CADRE the encodings could be expanded within the FUs. This dramatically reduces the size and amount of information that needs to be fetched from main memory. CADRE used a dual Harvard architecture that has one program memory and two separate data memories. In addition, a large on-chip register file of 256 16-bit words was included to avoid traffic and power dissipation in the main memories. In this way, the operands required by the FUs were provided directly from a register file. As with other DSPs, a 32-entry

instruction buffer was also included to handle loop instructions and reduce traffic to or from the program memory. Finally, all standard hardware components in CADRE were operated using self-timed techniques.

This top-level architecture has been adapted for the current research work because time is too limited to fully implement the CADRE design. However, we still keep the major advanced feature such as four-way parallelism to give high throughput. Meanwhile, a new FU has been designed with its configuration memory. The new system consists of four FUs connected together with a global bus and a pair of FUs are connected locally. The input data of each FU will be directly provided from on-chip RAM blocks with the output data being kept in another RAM block. The encoded top-level instruction is stored in a program memory. It contains controllable to enable the FU and accumulator writeback plus a 5-bit address which accesses the configuration memory associated with each FU; the functional unit instructions are stored in advance into each configuration memory of the FUs.

B. Instruction Set

The instruction set of the proposed FUs has two sets of instructions: computation and data movement and these can occur concurrently.

1) *Computational instructions*: These instructions contain arithmetic and logical operations; the arithmetic instructions include distance, normalization, shift, ADD, SUB, MPY and MAC (multiply and accumulate). The output destination of the operation can be 1 of 4 accumulator registers (AccA to AccD) inside the FU.
2) *Data Movement Instructions:* These instructions process the data movement in or out of the FU. These instructions include an accumulator register to accumulator register transfer within the FU itself, an accumulator register to an accumulator register in another FU, or a movement of data to the output RAM.

C. Example Assembly Codes

The two-dimensional discrete cosine transform (DCT) is a member of the family of sinusoidal transforms and is often used in image compression. An 8x8 2-D DCT has been implemented for compression algorithms such as JPEG. An algorithm for computing the 2-D DCT on a single 8x8 block in high-level C code is given below:

```
for (k = 0; k < 8; k++)
    for (l = 0; l < 8; l++)
    {
        sum = 0;
        for (i = 0; i < 8; i++)
            for (j = 0; j < 8; j++)
                sum += x[i][j] * c[i][k] * c[j][l];      // x[i][j] is 16 bits, c[i][k] and c[j][l] are
                                                         //        12 bits, and sum is 32 bits
        y[k][l] = (sum + 2^15) >> 16;                    // y[k][l] is 16 bits
    }
```

An example of the single 8x8 block of 2-D DCT forming the output y[0][0] optimally mapped to the four-way parallelism of the FU architecture is shown in Table1. The sampling data, coefficients, the input and output occupy 16 bits whilst the sum occupies 32-bits. The primary datapath for this FU is a 40-bit datapath. In the first two cycles, only multiplication is performed. Thereafter, the accumulators hold the valid data so a multiply-accumulate (MAC) operation is performed. When all MAC operations are complete, the four products of the 8 sums reside in accumulator registers A and B of each FU. The sum of loop j can be produced by combining them. Finally, all 8 products for loop i are then combined to produce y[0][0]. However, the final result needs to be rounded before writing the result back to RAM. This is repeated to apply the loops for k and l generating the output y(k,l). The way that operations are mapped to the hardware dramatically reduces the amount of switching activity within the multiplier because the value on the data bus within each FU is held constant for two successive instructions. In addition, the frequency of coefficient reading from memory is reduced by factor of two.

3. Energy Efficient Functional Unit Architecture

The challenge here is to meet the requirements of future portable applications, such as mobile phones. These devices have a very small power budget but need high performance with a complexity approaching that of a

desktop computer. The throughput has been achieved by adopting a parallel architecture allowing up to 4 instructions to be processed in parallel. In addition, parallel arithmetic logic is also used within a FU. The next challenge is therefore to scale the supply voltage downwards whilst maintaining the performance with the use of parallelism at different levels. Therefore, our FU can support supply voltage scaling without sacrificing the overall performance.

Table 1: Mapping 2-D DCT (8x8) onto 4-FUs to find y[0][0]

FU0	FU1	FU2	FU3
AccA= x[0][1]*C[1][0]	AccA= x[1][0]*C[0][0]	AccA= x[2][0]*C[0][0]	AccA= x[3][0]*C[0][0]
AccB= x[4][1]*C[1][0]	AccB= x[5][0]*C[0][0]	AccB= x[6][0]*C[0][0]	AccB= x[7][0]*C[0][0]
AccA= x[0][2]*C[2][0]+AccA	AccA= x[1][2]*C[2][0]+AccA	AccA= x[2][1]*C[1][0]+AccA	AccA= x[3][1]*C[1][0]+AccA
AccB= x[4][2]*C[2][0]+AccB	AccB= x[5][2]*C[2][0]+AccB	AccB= x[6][1]*C[1][0]+AccB	AccB= x[7][1]*C[1][0]+AccB
AccA= x[0][3]*C[3][0]+AccA	AccA= x[1][3]*C[3][0]+AccA	AccA= x[2][3]*C[3][0]+AccA	AccA= x[3][2]*C[2][0]+AccA
AccB= x[4][3]*C[3][0]+AccB	AccB= x[5][3]*C[3][0]+AccB	AccB= x[6][3]*C[3][0]+AccB	AccB= x[7][2]*C[2][0]+AccB
AccA= x[0][4]*C[4][0]+AccA	AccA= x[1][4]*C[4][0]+AccA	AccA= x[2][4]*C[4][0]+AccA	AccA= x[3][4]*C[4][0]+AccA
AccB= x[4][4]*C[4][0]+AccB	AccB= x[5][4]*C[4][0]+AccB	AccB= x[6][4]*C[4][0]+AccB	AccB= x[7][4]*C[4][0]+AccB
AccA= x[0][5]*C[5][0]+AccA	AccA= x[1][5]*C[5][0]+AccA	AccA= x[2][5]*C[5][0]+AccA	AccA= x[3][5]*C[5][0]+AccA
AccB= x[4][5]*C[5][0]+AccB	AccB= x[5][5]*C[5][0]+AccB	AccB= x[6][5]*C[5][0]+AccB	AccB= x[7][5]*C[5][0]+AccB
AccA= x[0][6]*C[6][0]+AccA	AccA= x[1][6]*C[6][0]+AccA	AccA= x[2][6]*C[6][0]+AccA	AccA= x[3][6]*C[6][0]+AccA
AccB= x[4][6]*C[6][0]+AccB	AccB= x[5][6]*C[6][0]+AccB	AccB= x[6][6]*C[6][0]+AccB	AccB= x[7][6]*C[6][0]+AccB
AccA= x[0][7]*C[7][0]+AccA	AccA= x[1][7]*C[7][0]+AccA	AccA= x[2][7]*C[7][0]+AccA	AccA= x[3][7]*C[7][0]+AccA
AccB= x[4][7]*C[7][0]+AccB	AccB= x[5][7]*C[7][0]+AccB	AccB= x[6][7]*C[7][0]+AccB	AccB= x[7][7]*C[7][0]+AccB
AccA= x[0][0]*C[0][0]+AccA	AccA= x[1][1]*C[1][0]+AccA	AccA= x[2][2]*C[2][0]+AccA	AccA= x[3][3]*C[3][0]+AccA
AccB= x[4][0]*C[0][0]+AccB	AccB= x[5][1]*C[1][0]+AccB	AccB= x[6][2]*C[2][0]+AccB	AccB= x[7][3]*C[3][0]+AccB
AccA= AccA*C[0][0]	AccA= AccA*C[1][0]	AccA= AccA*C[2][0]	AccA= AccA*C[3][0]
AccB= AccB*C[4][0]	AccB= AccB*C[5][0]	AccB= AccB*C[6][0]	AccB= AccB*C[7][0]
AccB= AccB+AccA	AccB=AccB+AccA	AccB= AccB+AccA	AccB=AccB+AccA
AccB=AccB+AccB(FU1)	NOP	AccB=AccB+AccB(FU3)	NOP
AccB=AccB+AccB(FU2)	NOP	NOP	NOP
Y[0][0]=AccBh	NOP	NOP	NOP

3.1 Configuration Memory

The regularity of typical DSP code allows multiple FUs to be employed without the power and area expense of dynamic scheduling hardware. For example, most modern DSPs, such as the TMS320C55x from Texas Instruments, use very long instruction words (VLIWs). In this case, program memory is fetched at the full rate demanded by the FUs and a lot of power is dissipated. Although cache would be a possible method to ameliorate this, there is an energy overhead in searching for a hit in cache memory. In our proposed FU architecture, the VLIW encoded instructions are stored in advance of executing each algorithm in a configuration memory located to each FU. This is different from others commercial DSPs such as the Phillips REAL DSP [10] or the Infineon CARMEL DSP [11] which have a single global configurable memory which is only used for special instructions.

In our design, the VLIW instructions for a particular algorithm are ‘cached’ by software and looked up using a short-form instruction. Up to 64 instructions are produced in a configuration memory this is more than sufficient for an anticipated algorithm; for example the DCT algorithm in table 1 requires only 6 encoded instructions in a configuration memory. Furthermore, the configuration memory instructions are completely user definable. The simple look-up avoids any tag overhead associated with a cache. The configuration memory makes the configuration of the FU flexible, enabling optimization by users. This flexibility has led to additional

hardware costs, particularly in the implementation of the configuration memory. These costs can be justified by the flexibility and performance gained by users. In particular, the use of configurable memory embedded into the design is unique compared with the other DSPs mentioned above. This feature also allows each FU to operate on an independent instruction stream if required.

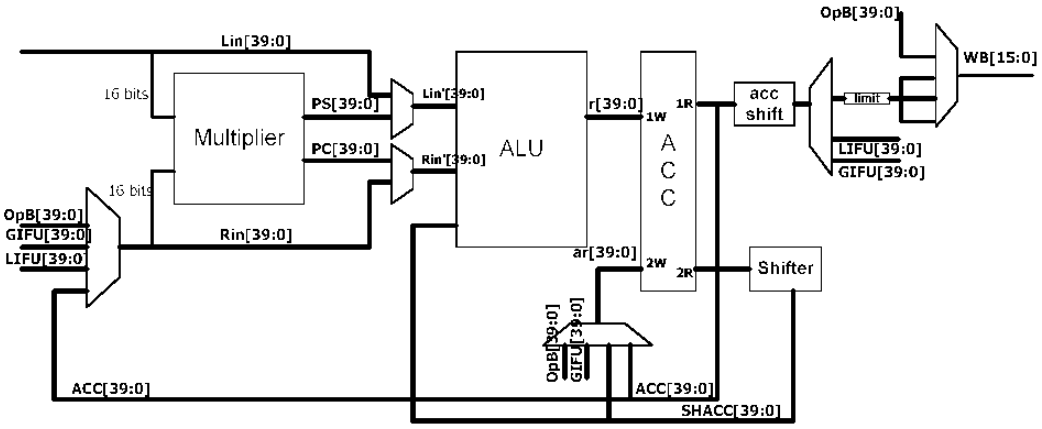


Figure 2 Functional unit datapath

3.2 Energy Efficient Arithmetic-Logical Unit

The FU datapath is shown in Figure 2. The number of available paths and input sources at different points mean that within the FU, many concurrent operations can occur within a single timeslot. For example, multiplication and addition with shifting can be performed in parallel with moving the contents of an accumulator result back to memory. Operations within the FU are termed major and minor. A major operation is usually performed in each timeslot and encompasses the arithmetic and logic operations. Data supplied to or from the FU is usually 16-bits. However written the FU the data is 40-bits wide as is usual in a DSP. Multiplication is 16x16 bits and produces a 40-bit partial sum (PS) and partial carry (PC) which are then forwarded to the adder in the ALU block for completion. Here, up to four operands may need to be added i.e. PS (or Lin) and PC (or Rin) together with an accumulator value (SHACC) which comes via the Shifter and a rounding constant (not shown). The ALU output is written to one of four accumulators AccA to AccD in the ACC.

Minor operations operate concurrently with a major operation. Here, in parallel with a major operation of addition and/or the multiplication, another Accumulator register can be written to either from another (shifted or unshifted) Accumulator register, or from the memory (OpB) or from the global bus GIFU. An Accumulator can also be written back to memory via the WB bus. To prevent unnecessary switching on the buses, transparent latches are inserted in front of multiplier, ALU and the second write port (2W) of the ACC.

The addition of the multiplier outputs PS and PC is performed by the adder in the ALU. This adder is therefore shared between the add/subtract and multiply/multiply-accumulate operations. This architecture is unusual in that most designs have a dedicated adder for the MAC operation and a separate adder provided for the addition. Having just one adder in the FU significantly reduces the amount of logic required in the datapath which in turn reduces power.

A. Asynchronous Circuit Design

The FU unit forms part of an asynchronous pipelined design. Asynchronous timing provides further power improvement as it eliminates clock generation, buffering and distribution. This asynchronous approach also gives a reduction of electromagnetic interference (EMI) as the switching of the logic is spread instead of being concentrated around the clock edge. The FUs are therefore based on the principle of micro-pipelines [12] where the data transfer between blocks uses local handshake signals rather than clocks. The principle is shown in Figure 3. The done signal allowing an output to propagate to the next micro-pipeline stage is generated either from the combinational logic for a data dependent operation or from a matched delay. Where the operation time is data dependent, as in the adder, the operation can be self-timed by using a *completion-detection* (CD) circuit. Many CD techniques are proposed [13-15]. The circuit in the FU has the additional requirement of

being low power. Therefore dual-rail coding having two wires per signal which always return to a '00' state, or duplicate logic producing the done signal which has the same delay as the combinational circuit is not suitable for such a low power application. Current-sensing CDs are also not suitable because they require an additional supply voltage and have higher quiescent current than synchronous circuits negating their advantage. Activity-Monitoring CD (AMCD) has an even higher energy-delay when there is a maximum ripple path. This is disadvantageous in a DSP since a maximum ripple path occurs in mamy arithmetic operations. Many designs use the *bounded delay* approach as it is the easiest CD method to implement. However, this is not a proper completion-detection method since the delay is fixed and needs to be longer than the maximum delay path. In [15], it is claimed that for combinational circuit having a critical path of less than ten gate-delays then a bounded delay approach works satisfactorily.

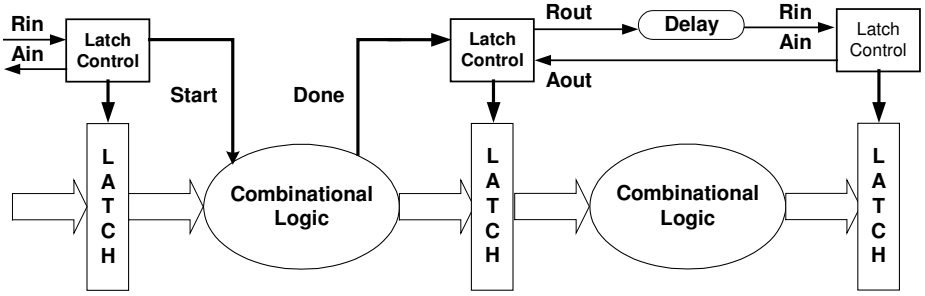


Figure 3 Asynchronous pipeline

Using different delay options for the same logic block has been proposed. For example, Nowick et al. proposed a method called '*speculative completion'* for use in a single rail asynchronous datapath [16]. This uses several different matched delays that allow each component to operate at a different speed. However, it involves extra circuit complexity, area and power dissipation. In our research work, we use our novel method to vary timing in an asynchronous control circuit by varying its supply voltage. It offers many advantages, such as allowing several different matched delays without incurring extra area on power overheads. Tuning the control delay to the datapath also improves performance and offers a flexibility not apparent in other asynchronous timing approaches. It therefore improves the likelihood of obtaining working asynchronous circuits at the first attempt and should lead to greater acceptability of asynchronous design techniques by easing the problem of designing timing and control.

B. **Multiplier Implementation**

The arithmetic unit has been implemented using two's complement rather than the sign and magnitude arithmetic in the initial design; this reduces design complexity, lowers dissipation and gives a better performance. The algorithms for GSM operations indicate a high proportion of multiplication operations. Generally, the multiplier consists of two main components: a partial product generation (PPG) and the addition of those partial products (PPs). The current 16x16 low-power multiplier is shown in Figure 4, has employed a modified Booth's algorithm [17] to reduce the number of PPs by dividing the multiplier bits into groups and selecting multiples of the multiplicand. Although larger groups of multiplier bits can give more PP reduction, a more complex selection table is required, so (as is usual) three bit groups are chosen to maintain speed while remaining low power.

The addition of the eight PPs requires a carry propagate adder (CPA) which has a long latency. To avoid this, a Wallace Tree [18] structure of 4-2 compressors is used which not only improves performance by reducing the latency but also reduces power by significantly reducing the amount of overall logic required. As a result of using tree structure topologies, the number of stages traversed by each input is approximately the same for all inputs. This leads to a *balanced delay tree* and results in less switching activity due to input skew. In addition, the eight PPs have been divided into two groups, PP1-4 and PP5-8 to produce partial carry and partial sum in parallel. Therefore, this multiply structure can achieve both performance and balance the delay in the tree structure. As the multiplier generates 40-bit outputs, all PPs have to be sign-extended. In order to minimize the logic within the tree structure, a pre-calculated sign-extension is applied. Furthermore, eight signed bits from the modified Booth's logic are forwarded to the adder; this reduces the depth of the Wallace tree logic required by one stage.

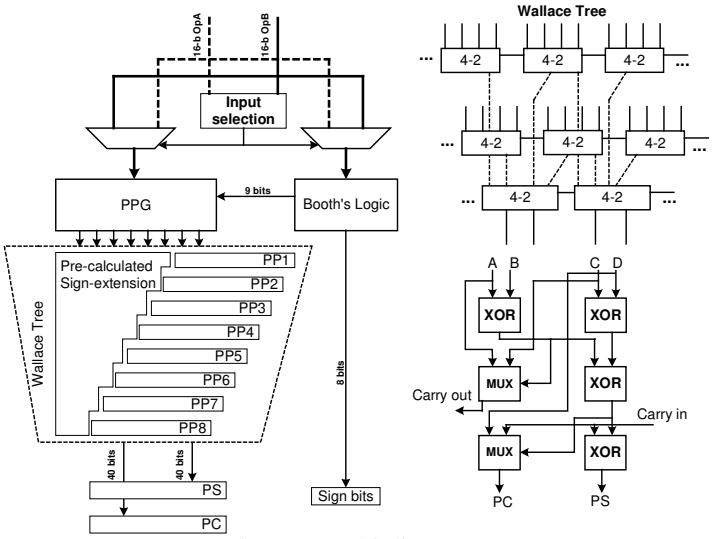


Figure 4 Multiplier Structure

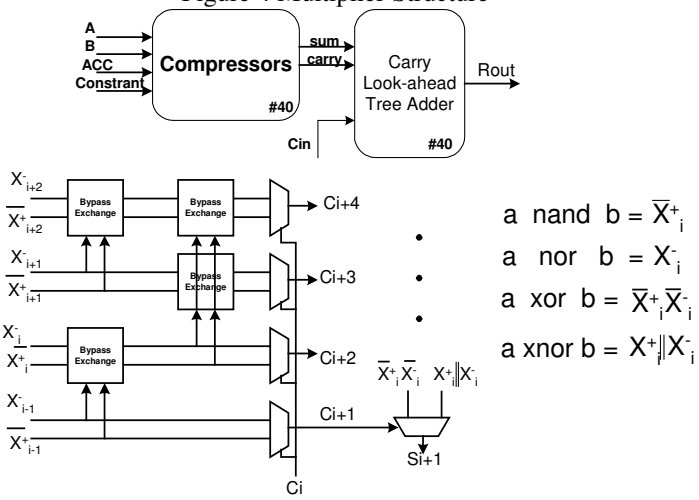


Figure 5 Structure of 4 input 40 bit adder and carry-look-ahead tree.

C. Adder & Associated Implementations

Operation analysis reveals that addition is the most frequent function performed in speech code. Thus, whilst the power dissipation of addition is approximately 1/5 that of multiplication; the very high proportion of additions means that it is essential to minimise the power required for this operation. In the FU, addition involves adding up to four variables as shown in Figure 5. The four inputs are first compressed to two by using 4-2 compressors. The two compressor outputs are then added in a carry-look-ahead tree. The carry-look-ahead equation, given in [19], has been modified when used in our design to enable an easy efficient mapping onto pass-transistor circuits. The carry is generated across blocks of four bits because this offers the best speed-power product. The adder in [19] was implemented using VHDL whilst a synthesis tool then used to generate the circuit and layout; therefore, its energy efficiency was relatively poor. In our design, the logic has been optimised and organised to map efficiently onto low energy pass transistor circuits and the circuits have then been laid out by hand to give an area efficient implementation. Thus the design achieved here is power efficient.

The normalized and Hamming distance instruction are normally included in the instruction set since they are useful for most DSP algorithms. In our design, we combine these functions into only one logic circuit. This not only reduces the number of logic gates but gives lower power dissipation than other designs. This novel logic also gives a performance improvement when compared with the conventional logic of normalized and distance designs. This is because carry-save techniques have been used in preference to the more usual carry ripple in the adder.

4. Energy Efficient Circuits and Layout Implementation

It is important to understand how energy is consumed in a circuit. An energy efficient design can be realised by either minimizing the energy consumption subject to a throughput constrain, or by maximizing the amount of computation for a given amount of energy. The optimal design can be made if the trade-off between the energy and delay can be met since it is possible to determine the lowest energy for a given level of performance. One approach in the system is to incorporate parallelism or pipelining. In addition, use of energy efficient circuits to implement the sub-components in the system result in good energy efficiency. Therefore, this section will discuss and show the energy-delay related to the output loads and transistor sizing. The multiplexer cell as shown in Figure6 implemented by pass-transmission gate is the main cell applied everywhere within the FU datapath. If S is low, the top CMOS pass gate is on passing B to the output, whilst if S is high, the bottom CMOS pass gate is on and A passes to the output.

Gate sizing can be used to trade-off energy and delay. The minimal energy delay point of a circuit is not only affected by its basic design but also by its output load capacitance. The logic gate capacitance and load capacitance including wiring both increase linearly with transistor size. Gate delay can be calculated as $t_d = \tau d$, where τ is a process-dependent constant, and d is a unitless delay of the gate. The unitless delay is determined as $d = h_{eff} + \rho$, where h_{eff} is the product of logical effort of the driving gate and electrical fan-out; a logical effort of 1 results from a minimum size inverter driving a similar inverter. The electrical fan out is the equivalent number of minimum size inputs being driven. The self-loading delay ρ is the product of logical effort (g) and the ratio of the equivalent driven gate width to the equivalent driving gate width, $\rho = gW_{par}/W_{in}$. Because there are no publications relating to the logic effort of a pass-transistor logic as yet, we regard the multiplexer as a logic gate. In our work to find the minimal point of energy delay of multiplexer, the transistor size and loading capacitance for the gate in figure 6 are varied as shown in Table3. The circuits were simulated on SPICE assuming a geometry of 0.18μm and an operating voltage of 1.8V.

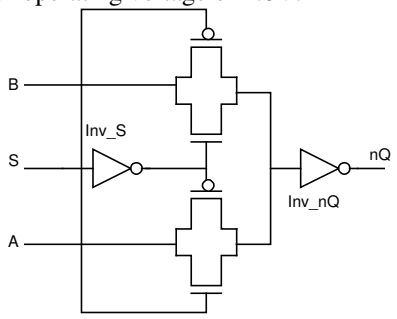


Figure 6 A pass-transmission gate multiplexer

Table 2: Energy delay product simulation results of pass-transmission gate multiplexer using transistor sizing with various load capacitances.

*0.18um geometry operating at 1.8V. CASE	Load Capacitance = 10 ff (equal to 4 Inverters)		Load Capacitance = 20 ff (equal to 8 Inverters)		Load Capacitance = 30 ff (equal to 12 Inverters)	
	Et (pJ) (worse case)	Et (pJ) (average case)	Et (pJ) (worse case)	Et (pJ) (average case)	Et (pJ) (worse case)	Et (pJ) (average case)
1	6.049	5.691	18.298	14.220	33.778	23.542
2	6.909	6.831	19.989	16.112	37.726	26.848
3	8.302	8.118	22.879	18.265	41.784	29.696
4	8.422	6.258	14.409	11.119	21.769	17.390
5	8.802	6.563	14.895	11.530	22.160	17.795
6	7.791	5.669	14.029	10.585	21.698	16.924
7	6.054	6.054	17.680	11.676	29.061	19.178
8	9.719	6.638	18.110	12.256	29.280	19.703
9	8.460	6.421	16.467	12.282	27.310	20.079
10	7.319	5.159	16.624	11.267	28.157	18.946

CASE	Pass Gates (-e06 m)		Inv_S (-e06 m)		Inv_nQ (-e06 m)	
	NMOS	PMOS	NMOS	PMOS	NMOS	PMOS
1	0.28	0.28	0.28	0.28	0.28	0.28
2	0.28	0.28	0.70	1.24	0.28	0.28
3	0.80	0.80	0.70	1.24	0.28	0.28
4	0.80	0.80	0.70	1.24	0.70	1.24
5	1.00	1.00	0.70	1.24	0.70	1.24
6	0.28	0.28	0.70	1.24	0.70	1.24
7	0.28	0.28	0.70	1.24	0.36	1.46
8	0.80	0.80	0.70	1.24	0.36	1.46
9	0.80	0.80	0.28	0.28	0.36	1.46
10	0.28	0.28	0.28	0.28	0.36	1.46

Table 2 shows the energy delay product resulting from the use of different transistor sizes in the multiplexer and different load capacitances. We have analyzed the energy delay product of the circuit with both average rise and fall times and worse case edge times. Clearly, transistor sizing can help the circuits to have a low energy delay product when load capacitance is increased. In practical, load capacitance is very important and has a large effect both the performance and power of the system, especially in a large design. In addition, using the minimal transistor size cannot achieve optimum energy efficiency. It can be concluded from these results, that the CMOS pass gate transistors that for minimum energy-delay should be minimum size, the inv_S should be a drive 1 gate (PMOS/NMOS width = 1.24e-06m/0.70e-06m) and be capable of driving the select signals on 2 or 3 multipliexer gates, and that the inv_nQ should be matched to the driven load (PMOS/NMOS width = 1.24e-06m/0.70e-06m). These figures do not include leakage power which is small for this process and provided the transistor sizing above is adopted.

Finally, a full custom layout is required to build an energy efficient system. This is especially important when the pass-transistor circuit topology is used since we can achieve less area and a non-predictable load capacitance. In addition, the physical capacitance and the length of wires can be minimised. Therefore, full custom design combined with transistor sizing will give a big energy saving compared to the automatic place and route tools which have developed to reduce the energy consumption and which yield only an 18% reduction[20].

5. Discussion and Conclusions

The results from remaining simulations on the full custom layout for the datapath shown in Figure2 indicate that our FU dissipates about 13.68mW@200MHz. Projecting this to four-way parallelism yields 800MHz with a power consumption of only 54.72mW. This indicates a factor of 10 improvement in the dissipation on the original FU design assuming a similar geometry and operating voltage.

Energy efficiency is a significant design constraint for battery powered DSPs requiring a coherent low energy technique applied at all levels and particularly at logic and circuit levels. The proposed FU architecture has been designed to be flexible, have low power and high throughput particularly in performing arithmetic operations involving the multiplier and adder. In addition, asynchronous data dependent operation and a tunable delay mechanism are incorporated to gain even better energy efficiency. At the circuit level, pass transistor logic is used which is low power without sacrificing performance. Transistor sizing has been applied to this pass transistor logic to identify the optimum trade-off between energy and delay.

In the future, the need for energy efficiency design should result in a move from low-level energy-efficient techniques to higher levels as the automatic tools which currently do not support coherent lower energy strategies.

Acknowledgement

This work was funded by EPSRC grant GR/S61270/01 and the authors are grateful for this support. The authors are also grateful to Jim Garside for discussion and valuable suggestions, and to Dave Clark, Jeff Pepper and Steve Temple for encouragement in this work.

References

[1] Sheng S., Chandrakasan A. and Brodersen R.W., "A portable multimedia terminal.", *IEEE Communications Magazine*, pp. 64-75, vol.30, no.12, December, 1992.
[2] Reiner H., "Reconfigurable Computing: A New Business Model and its Impact on SoC Design.", Keynote speeches in *EUROMICRO Symposium on Digital System Design, Architectures, Methods and Tools*, September, 2001.
[3] Bo Z., David B., Dennis S. and Krisztian F., "Theoretical and practical limits of dynamic voltage scaling.", *ACM/IEEE Design automation conference (DAC)*, under review, June, 2004.
[4] Li-Hsun C., Chen O.T.-C. and Ruey-Ling M., "A high-efficiency reconfigurable digital signal processor for multimedia computing.", *Proceedings of the 2003 International Symposium on Circuits and Systems*, vol.2, pp. 768-771, May, 2003.
[5] Sangjin H., Shu-Shin C. and Connaway C., "Variable-rate pipelined multiplier design for reconfigurable DSP applications.", *45th Midwest Symposium on Circuits and Systems*, vol.1, pp. 587-590, August, 2002.
[6] Martina M., Masera G., Piccinini G., Vacca F. and Zamboni M., "Reconfigurable DSP IP for multimedia applications.", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol.4, pp. 4179, May, 2002.
[7] I. Verbauwhede, P. Schaumont, C. Piguest and B. Kienhuis, "Architectures and design techniques for energy efficient embedded DSP and multimedia processing.",
[8] M. Lewis and L. Brackenbury. CADRE: An Asynchronous Embedded DSP for Mobile Phone Applications. *Design Automation for Embedded Systems*, vol. 6, No. 4, pp. 451- 475, 2002.
[9]A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.
[10]P. Kievits, E. Lambers, C. Moerman and R. Woudsma, "R.E.A.L. DSP Technology for Telecom Basedband Processing", *Proceeding 9th International Conference on Signal Processing Applications and Technology*, Miller Freeman Inc., 1998.
[11]*Carmel DSP Core Technical Overview Handbook*, Infineon Technologies, 2000.
[12] I.E. Sutherland, "Micropipelines.", *Communications of the ACM*, vol.32, no.6, pp.720-738, June, 1980.
[13] M.E. Dean, T.E. Williams and D.L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (LEDR)." *MIT Conference on Advanced Research in VLSI,* March 1991.
[14] V. Varshavsky, V. Marakhovsy and M. Tsukisaka, "Data-controlled delays in the asynchronous design," *Proceeding Of the 1996 IEEE International. Symposium Circuits and Systems (ISCAS'96),* Atlanta (USA), vol. 4, pp. 153-155, May 1996.
[15] E. Grass, Viv Bartlett and Izzet Kale, "Completion-Detection Techniques for Asynchronous Circuits." *IEICE Transaction Information and System,* vol.E80-D, no. 3, March 1997.
[16] S.M.Nowick, K.Y.Yun, P.A.Beerel and A.E.Dooply, "Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders.", *Proceedings of Async97*, pp. 210-223, April, 1997.
[17] A. D. Booth, "A Signed Binary Multiplication Technique.", *Quarter Journal Mech. Applicantion Math.*, vol. 4, pp. 236-240, 1951.
[18] C. S. Wallace, "A Suggestion for Fast Multipliers.", *IEEE Transactions Electronic Computer*, vol. EC-13. pp. 14-17, February, 1964.
[19] Keshab K. Parhi, "Low-Energy CSMT Carry Generators and Binary Adder.", *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol.7, no.4, pp.450-462, December, 1999.
[20] Chao K. and Wong D., "Low power considerations in floorplan design.", *International workshop on low power design*, Napa Valley, CA, pp.45-50, April, 1994.

Implementing PID control systems using resource-limited embedded processors

Simon Key and Michael J. Pont

Embedded Systems Laboratory, University of Leicester,
University Road, Leicester LE1 7RH, UK.

www.le.ac.uk/eg/embedded

Abstract

This paper is concerned with the development of software for PID control systems implemented in an embedded form. We seek to do two things. First, we consider how the choice of implementation method affects memory and CPU requirements (and, hence, system cost). Second, we consider how the same implementation decisions affect control performance.

Acknowledgements

The authors would like to thank Dr John Twiddle (University of Leicester) for his assistance and advice, and Perkins Engines Company Ltd for providing the diesel generator used in this study. This work is funded by the UK Government (EPSRC / DTA award).

1. Introduction

Modern control systems are almost invariably implemented using some form of digital computer system (Kilian, 2001). The dominance of digital systems in this field is due to the cost, flexibility, ease of use, and performance of digital control algorithms when compared with equivalent analogue implementations (Åström, 1997; Virk 1991).

However, despite many advantages, successful implementation of a digital embedded control system requires knowledge of both software engineering and control engineering: at present, there is a gap between these two disciplines (Törngren, 1998). One consequence is that current control theory tends makes assumptions about the timing, data acquisition, and resource availability in embedded environments. For example, with sampling and actuation it is assumed that accurate timing behaviour is inherent in embedded control systems (Åström, 1997), with no further consideration of the effects of timing errors. In addition, the possible connection between control algorithm, implementation method, and processor requirements are not usually considered.

Many modern control designs are developed and tested with the use of Simulink® and dSPACE or Real-time workshop® on a Desktop PC, then - using automatic code generation - transferred to the embedded controller (Matlab 1,2; dSPACE 1,2). This raises two questions: [1] From the control standpoint - are we using the most cost-effective hardware? [2] From the embedded standpoint - are we using an appropriate implementation of the control algorithm?

In this paper we explore the implementation of a PID algorithm in a resource limited embedded controller. Whilst the effects of numerical accuracy and quantisation are well known from a theoretical background (Katz, 1981; Williamson, 1991; Widrow, 1956; Slaughter, 1964); the effects in a practical controller have not been extensively explored; similarly the resource usage of control algorithms has not previously been explored in academic papers.

The particular focus of the paper is on the speed control of a diesel engine generator. Various experimental types of speed regulation have been applied to diesel engines, including sliding mode control (Goh, 2003), H_∞ (Xiros, 2001), and fuzzy logic (Wijetunge, 2001) but PID control continues to dominate in the industrial sector. For power generation the selection of hardware is typically an ad hoc process with the typically assumption that power generation requires the use of either 16- or 32-bit microcontrollers (Stobart, 1999).

This paper is organised as follows. Section 2 describes the diesel engine generator and speed control system used in this paper. Section 3 describes the PID algorithm and program implementation. The theoretical background to limited word length processes is covered in Section 4. The performance of the controller in use on the diesel generator is demonstrated in Section 5, and experimental results in terms of program size and execution times are in Section 6. Section 7 discusses the results of this paper and presents the conclusions.

2. The diesel engine generator

The diesel engine “Gen-Set” used in the studies presented in this paper is a turbo-charged, four cylinder, four-litre diesel powered electrical generator set, (Perkins model 1004TG2). The basic model has one control loop to control engine speed; this consists of a GAC ADC100 electrical actuator on the fuel injection system controlled by a GAC ESD5500E speed control unit, which in turn receives the engines rotational speed from a GAC MSP6730 magnetic pickup. The ESD5500E implements a two-stage PID controller; the first stage controls engine start up and

controls the increase in engine speed from stop to running speed, the second stage PID controls engine speed in the run state. Engine speed can be varied between 1500 RPM where 65kW of electrical power is generated at 50Hz or 1800RPM where 75kW of electrical power is generated at 60Hz. The hardware configuration is shown in Figure 1 with the set point used to determine whether 50Hz or 60Hz electrical power is generated.

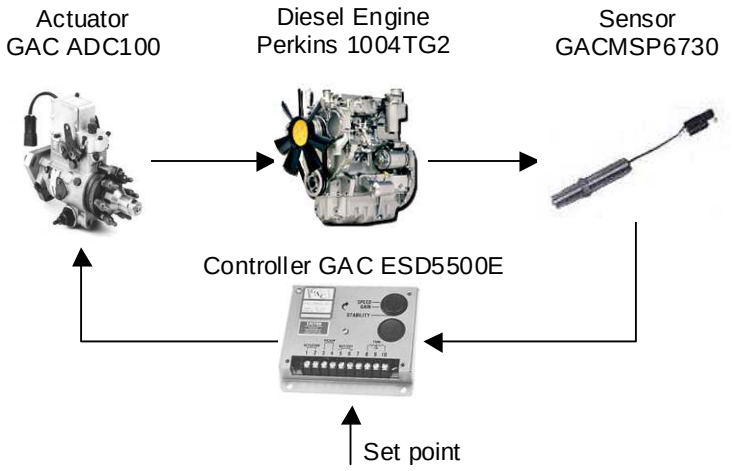


Figure 1 Original speed-control configuration

For the purposes of the studies in this paper, the ESD5500E controller was replaced by a Phytec kitCON-515 development board, incorporating an Infineon C515C microcontroller. The C515C (Infineon, 1998) is a low-cost 8-bit processor based on an "extended" 8051 architecture. It has several on-chip components which are well matched to the needs of the current study, including hardware support for the generation of pulse-width modulated (PWM) signals, and an 8-Channel (10-bit) analogue-to-digital converter (ADC).

This study also used a Perkins EX.1018CC Dual Frequency-Voltage card converter, a passive single-pole 50 Hz Butterworth low-pass filter Figure 2, and a 10dB gain pre-amplifier/buffer for the C515C's internal A/D converter, using a MAX492 (Maxim, 1996) Op Amp Figure 3. As well as amplification of the output PWM signal from 5V p-p to 12V p-p and current deliver of 20mA increased to 3A (Figure 4).

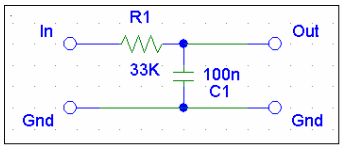


Figure 2 Input Filter

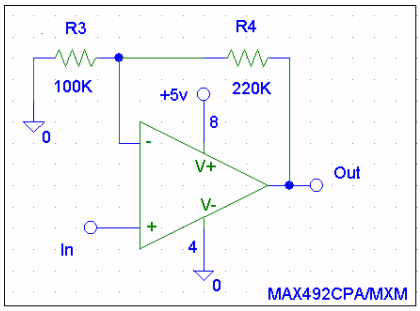


Figure 3 Pre- Amplifier

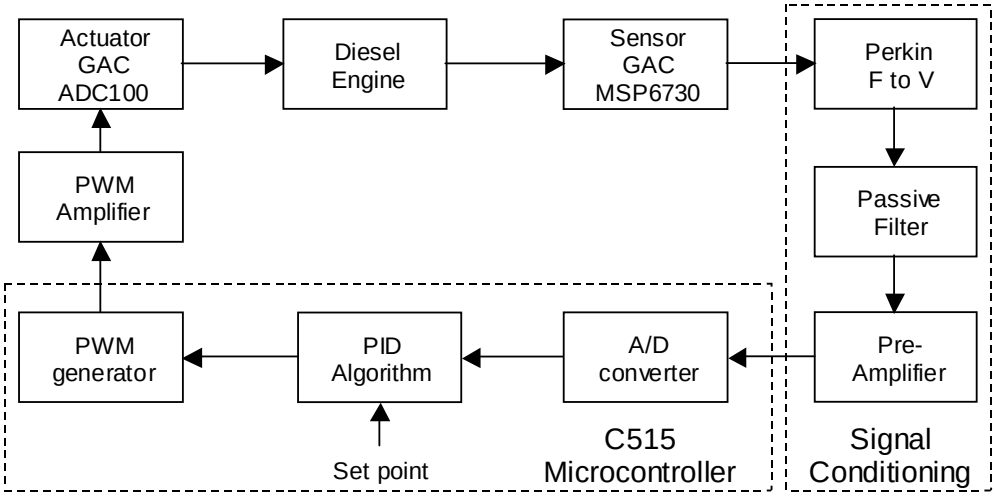


Figure 4 Experimental speed controller configuration

3. PID implementation

The PID control algorithm developed by Callender and Stephenson (Callender, 1939) remains in widespread use, and it is estimated that between 50% (Ogata, 1995) and 90% (Franklin, 1995) of all control systems are based on this technique.

There are four basic implementation methods for a PID controller: direct form 1 (series); direct form 2 (canonical); cascade; and parallel (Katz, 1981). In addition, there are positional and velocity formats (Katz, 1981). In this paper, we consider the positional, parallel, form of the PID controller as – although this method loses some of the simplicity of the equivalent direct form – it generates smaller errors (Katz, 1981).

A block diagram of the parallel PID controller is shown in Figure 5. The associated continuous time representation is given in Equation 1, with the discrete time equivalent in Equation 2 (Franklin, 1994). This leads in turn to the difference equation given in Equation 3 and hence, the program code in Listing 1 (adapted from Pont, 2001).

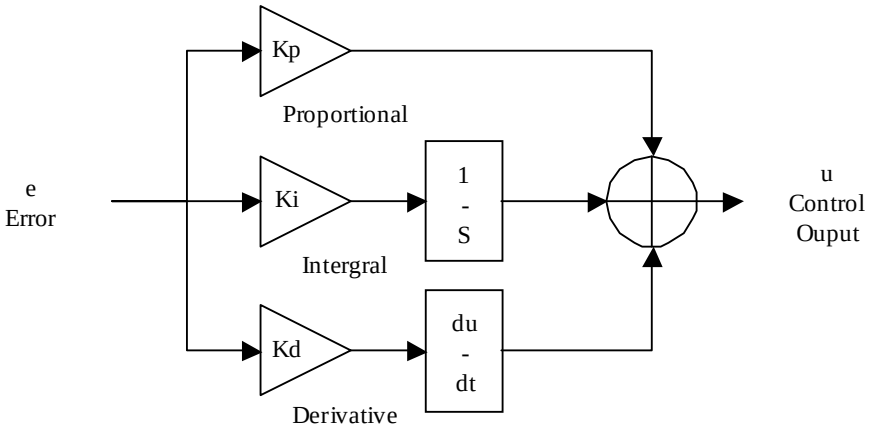


Figure 5 Time domain PID controller

$$u_C(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de}{dt} \quad \textbf{(Equation 1)}$$

$$U_C(s) = k_p E(s) + \frac{k_i}{s} E(s) + k_d s E(s) \quad \textbf{(Equation 2)}$$

$$u_C(k) = k_p e(k) + k_i T e(k) + k_d \frac{(e(k) - e(k-1))}{T} \quad \textbf{(Equation 3)}$$

```
/* Proportional term */
Control_new = (PID_KP * Error);

/* Integral term */
Control_new += PID_KI * SAMPLE_RATE * Error;

/* Differential term */
Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));
```

Listing 1 Simple C code for parallel PID controller. Adapted from Pont (2001).

The material so far has described the basic PID algorithm. This algorithm needs to be implemented in a manner that allows sampling and actuation to be carried out at regular intervals. In this paper, we achieve this using a Time-Triggered Co-operative Scheduler (TTCS) (Pont, 2001). While many other architectures are possible (and some may be more resource efficient: see Locke, 1992), a TTCS provides accurate timing and a well-understood system architecture (Pont, 2001).

As a first step in the development of the TTCS implementation, it was important that an appropriate sample period was chosen for the control loop. The rise time of the Gen-Set speed signal at start up is between 2 and 2.5 seconds (Figure 6). Common estimates suggest that the sample rate can be determined by dividing this figure by 4x to 10x (Åström, 1997), 5x to 20x (Houpis, 1992) or 20x to 40x (Franklin, 1994)[1]. In this case a sampling period of 10ms is used to minimise the overall effect of sampling time upon any particular implementation.

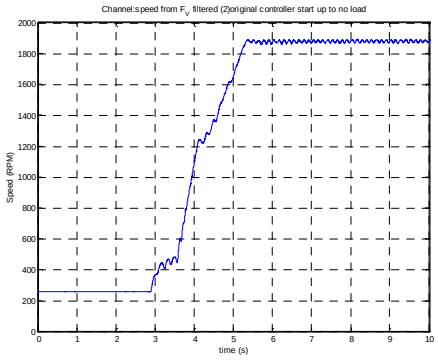


Figure 6 Rise time for the original controller

[1] As the variation in these values makes clear, the selection of sampling period is a rather ad hoc affair from the control engineer's point of view: by contrast, the sampling period plays a very significant role from the software engineer's perspective. Sampling at a shorter sample period than necessary will – for example – add significantly to the system CPU requirements (and, hence, the system cost) the CPU resource usage of the control algorithm.

In the study described here, all the tasks were executed at 10 ms intervals, using a scheduler with a tick interval of 1 ms. The A/D sampling task (AD_Sample) had an offset of 0 ms followed by the PID calculation task (PID_Update, PID_Control) with an offset of 1ms, finally, with an offset of 4 ms, the control signal is sent to the actuator (task PWM_Update). The task arrangement is shown schematically in Figure 7.

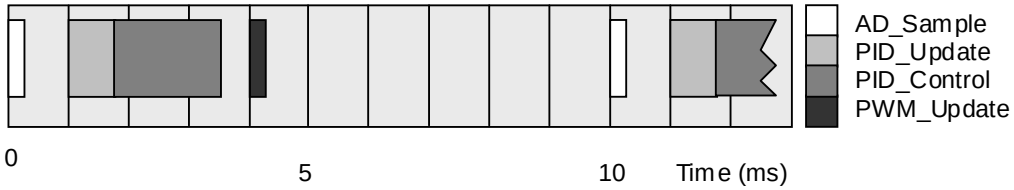


Figure 7 PID controller task management

4. Finite word effects

In any real digital control system, the process variable will have to be quantised for use in the digital controller. The effect of quantisation is to introduce an error into the system. Depending on the analogue signal and the type of quantiser used, the error can been modelled as additive white noise (Bennett, 1948) with studies on the statistical composition of the quantisation noise and limits of this model been conducted later (Widrow, 1956; Bertram, 1958; Slaughter, 1964).

Quantisation errors can occur in three sections of a digital controller: A/D conversion, arithmetic operations (e.g. multiplication and division), and coefficient storage (Katz, 1982). A/D conversion is limited by the resolution of the A/D converters. Mathematical operations are limited by the implementation methods used in the arithmetic code libraries, with support for integers and floating point on 8-bit microcontrollers typically having a maximum resolution of 32 bit (cosmic, 2004) (Keil, 12004) (Raisonance, 2004). Similarly, the word length of the processor and compiler implementation limits coefficient storage and the arithmetic operations.

There are essentially two different types of quantisation process errors: rounding and truncation. Rounding errors occur where the input value is rounded to the nearest discrete quanta (Figure 8). The error is given in Equation 4.

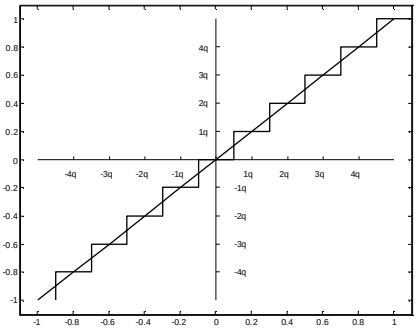


Figure 8 Round off of a continuous value between -1 and 1 with 11 discrete quanta (from Katz, 1982)

$$e_R = Q_R[x] - x \qquad \text{(Equation 4)}$$

Truncation errors occur where the input variable is rounded down to the nearest quanta (Figure 9). The error is given in Equation 5.

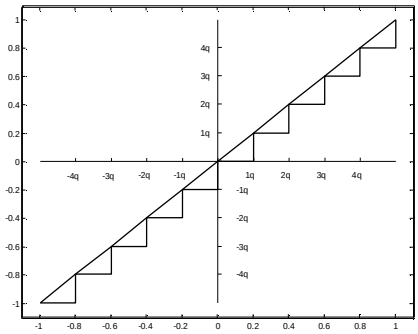


Figure 9 Truncation of a continuous value between -1 and 1 with 11 discrete quanta (from Katz, 1982)

$$e_T = Q_T[x] - x \qquad \text{(Equation 5)}$$

In terms of arithmetic, there are two basic choices: integer (fixed point) or floating point. Floating point gives a better dynamic range and is less likely to encounter a limit overflow (Williamson, 1992). However, these advantages are likely to occur at the expense of longer computational time (Morton, 2001), unless floating-point hardware is available. Fixed-point arithmetic is implemented using 8-, 16-, and 32-bit representations, with longer word lengths providing increased resolution. With the limited range of the fixed point arithmetic greater care is required in the construction of the control algorithm to prevent overflows and reduce signal quantisation by providing appropriate scaling of the control variables (Hanselmann, 1987; Houpis, 1992). With insufficient resolution the performance of the digital controller decreases and phenomena - such as limit cycles and oscillatory output – can become apparent (Houpis, 1992).

5. Control performance

We began our comparisons by comparing the (control) performance of three implementations of the speed-control system. The three implementations used floating-point arithmetic, 32-bit integer arithmetic, and 16-bit integer arithmetic. In this case a P-only (proportional only) controller was implemented.

If we compare the transfer functions of the PID algorithms, this will give a predictor of the differences between the implementations. The open loop response transfer functions (shown in Figure 10) demonstrate that for proportional control the mathematical implementation of the controller is not significant.

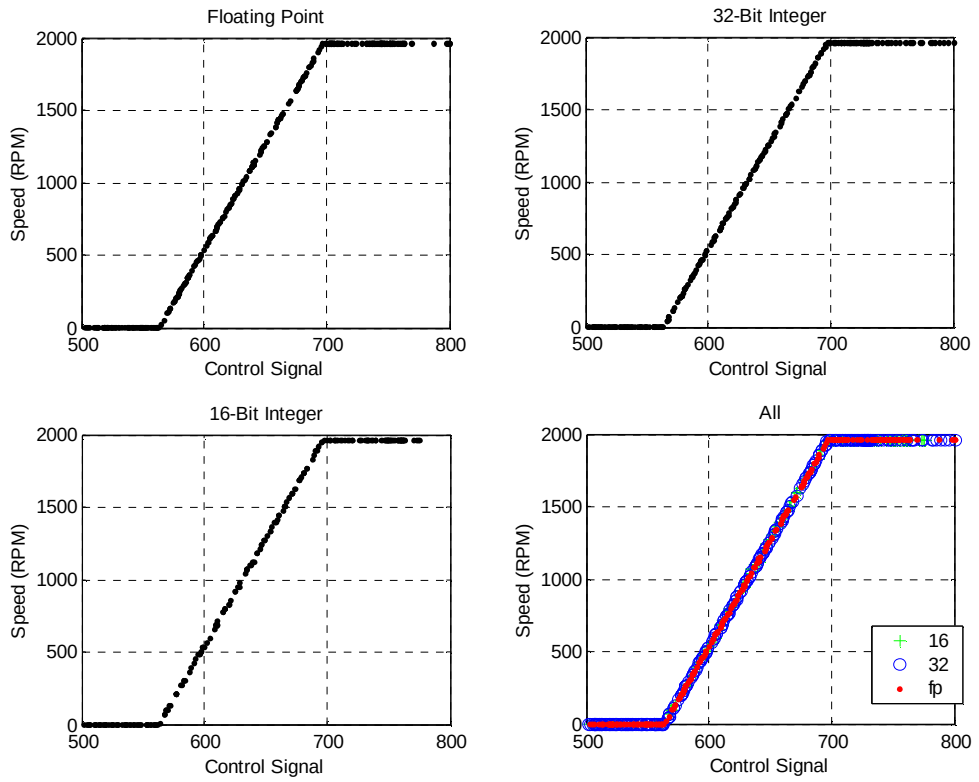


Figure 10 Open loop Response for the proportional-only controller

Figure 11 Figure 12 and Figure 13 show the floating point, 32-bit integer, and 16-bit integer speed regulation respectively. Each of the implementations was tested by applying a load to the Gen-set. The arrows close to the bottom of the graphs show the times when a load change occurred. The load is increased in steps of 10kW from zero load to 60kW and then similarly decreased back to zero load. Figure 14 Figure 15, and Figure 16 show the response in close-up around the set point with no load applied to the controller.

Overall, while the controller is only loosely tuned (with some oscillatory behaviour apparent), the differences between the implementations, under normal conditions, are small.

As a side issue, the effective resolution of the A/D converter was reduced by a programming technique known as left shifting (using the C >> operator). The original resolution of 10 bits was reduced in 1-bit decrements until the controller could no longer maintain a running state; the plots of speed versus time are shown in Figure 18, Figure 19, Figure 20,Figure 21, and Figure 22, for 10-, 8-, 6-, 4- and 3-bit resolution, respectively. It can be seen that the range of speed values is relatively constant for the change of resolution until the lower resolutions of 4 and 3 bit are reached. Note that the vertical range for the 4-bit graph is 200 RPM, the 3-bit graph is 300RPM, and all others are 100RPM.

Figure 17 shows the speed regulation of the original controller. The range of speed values is of a similar order to that of the speed controllers described in this paper.

6. Resource usage

As we have seen, for this case study, the control performance of the various implementations are similar. The choice of implementation can then be made based on the resource requirements of the algorithm. In this section, we discuss the resource requirements.

The control algorithm is (as discussed earlier) a parallel PID system, the implementation of which can – here - take three forms: floating point, 32-bit integer, and 16-bit integer. As we will see, these implementations have different resource requirements.

6.1. Code size

Initially we examined the code size for each implementation. The complexity of the mathematical libraries has a significant effect upon the size of the code, as does the hardware upon which it is implemented. To reduce the bias which might result from the use of a single compiler (since the results would depend upon the implementation of the mathematical libraries), two compilers were used, the Keil C51 Compiler, v7.10), and the Raisonance RC51 compiler, v1.1.2 (BN732).

The pattern of decreasing code size for floating point, 32 bit integers, and 16 bit integers is observed in both compiler implementations (Figure 23). This trend is expected (Morton, 2001; Lewis, 2002), as use of floating-point arithmetic requires a considerable amount of code, usually in the form of libraries, whereas 16-bit integers can be implemented almost entirely using native machine instructions.

6.2. Memory requirements

As with code size the RAM requirements of the program vary according to the compiler used and the implementation mathematics (Figure 24). For both compilers the trend is similar with (unexpectedly) the 32-bit integer program having the largest requirements for data memory, and 16-bit having the lowest requirements. A 32 bit integer requires the same storage space as a floating point variable this coupled with the extra manipulation in the form of scaling, would account for the large than expected RAM requirements for the 32 bit integer implementation. The differences in size between implementations are less pronounced than with the code sizes.

6.3. Timing constraints

The timing characteristics of the system are of major importance in any real-time design. In particular, for a control system, missing deadlines will result in degradation of the system performance.

Figure 25 shows average module execution times, Figure 26 shows worst-case times, and Figure 27 shows best-case times. The modules chosen for examination are the essential modules of the PID algorithm the A/D conversion (AD_Sample), the actuator control (PWM_Update), and the PID algorithm (PID_Control, and PID_Update) PID_Total is the total time taken for the PID algorithm.

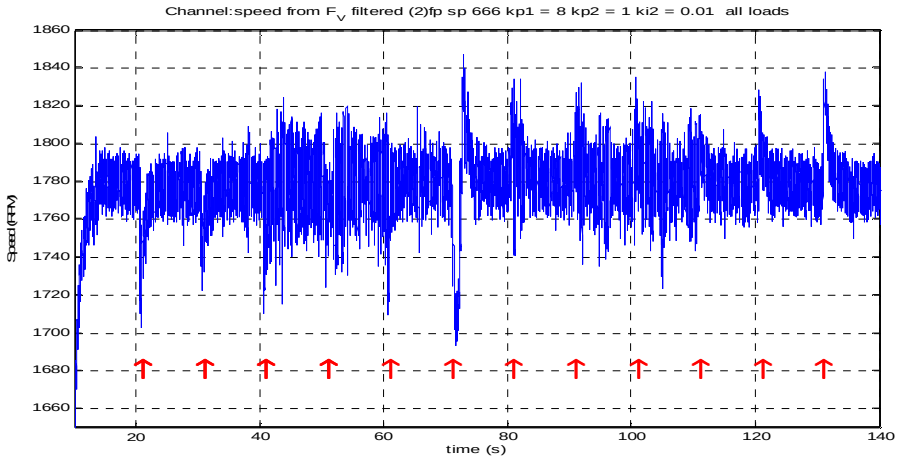


Figure 11 Speed regulation for experimental floating-point PID controller

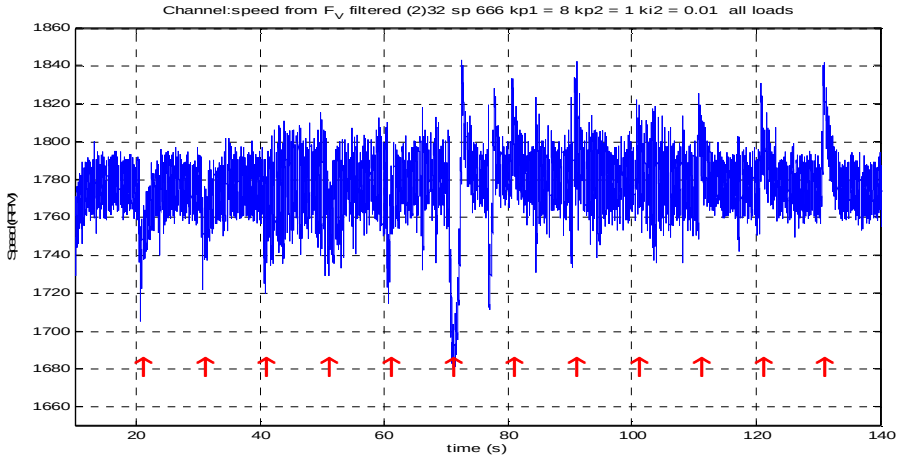


Figure 12 Speed regulation for experimental 32-bit integer PID controller

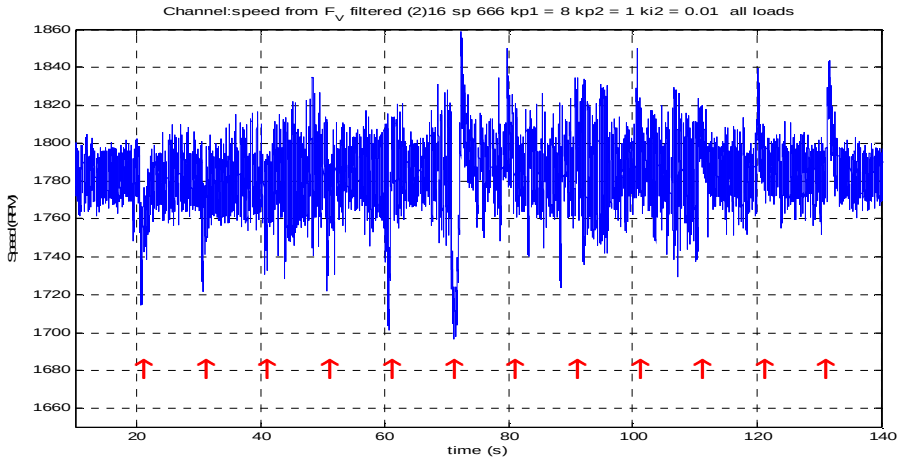


Figure 13 Speed regulation for experimental 16-bit integer PID controller

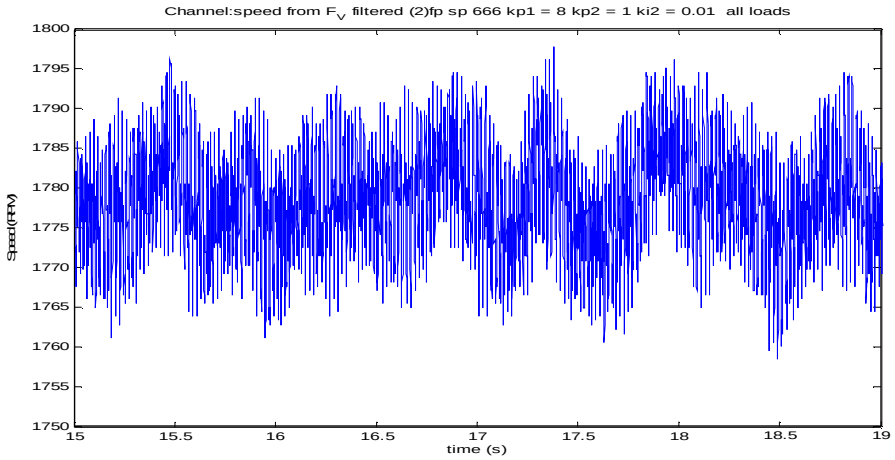


Figure 14 Speed regulation for experimental floating-point PID controller (no load)

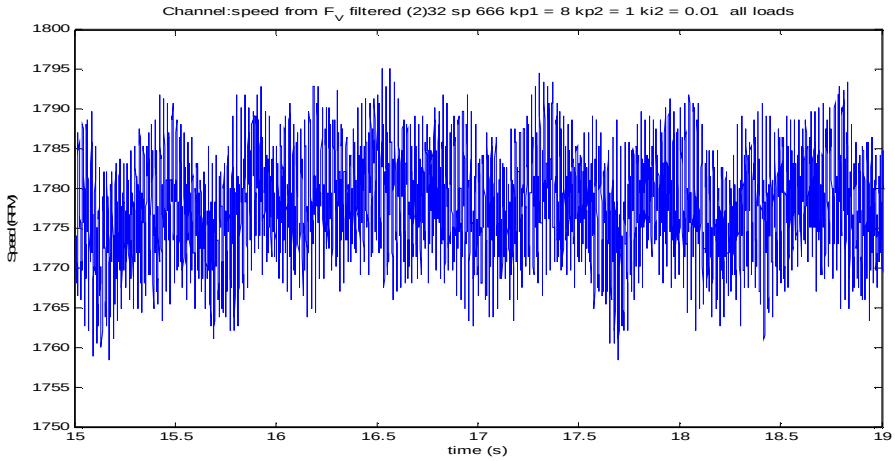


Figure 15 Speed regulation for experimental 32-bit integer PID controller (no load)

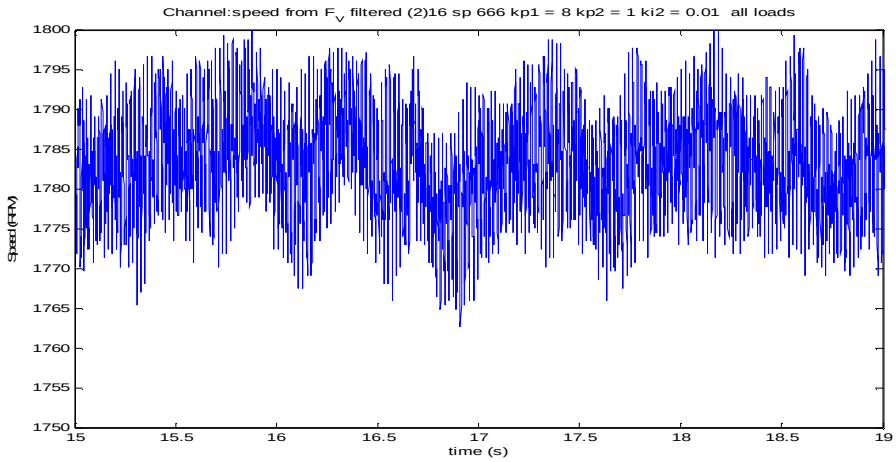


Figure 16 Speed regulation for experimental 16-bit integer PID controller (no load)

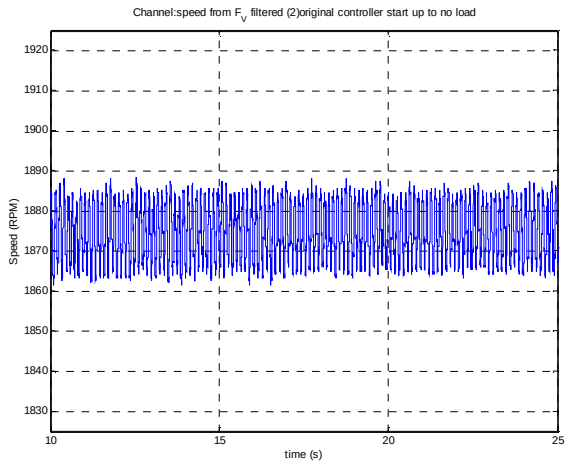


Figure 17 Speed regulation for original controller

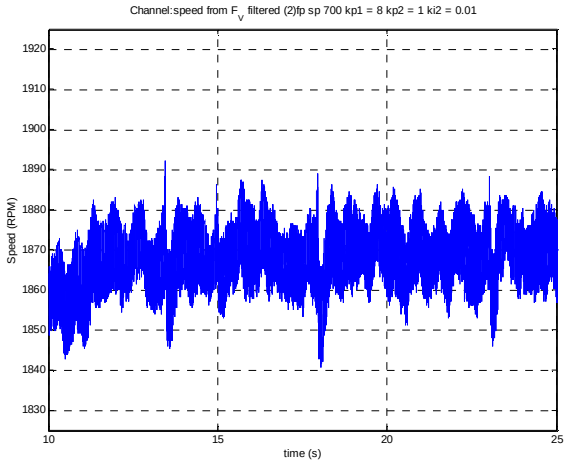


Figure 18 Speed regulation for 10-bit A/D conversion

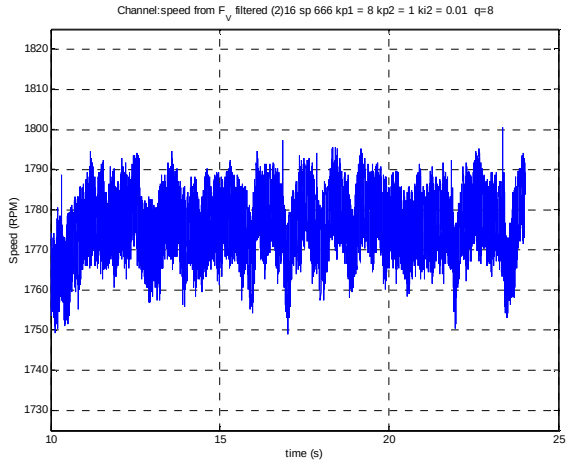


Figure 19 Speed regulation for 8-bit A/D conversion

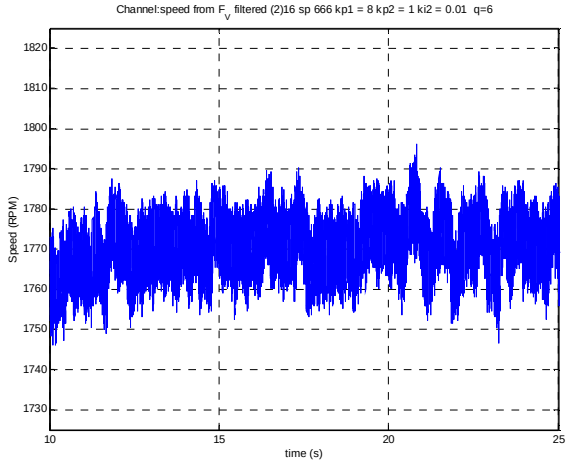


Figure 20 Speed regulation for 6-bit A/D conversion

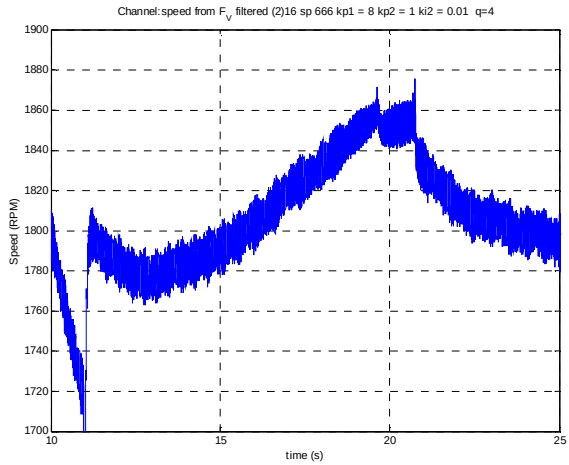


Figure 21 Speed regulation for 4-bit A/D conversion

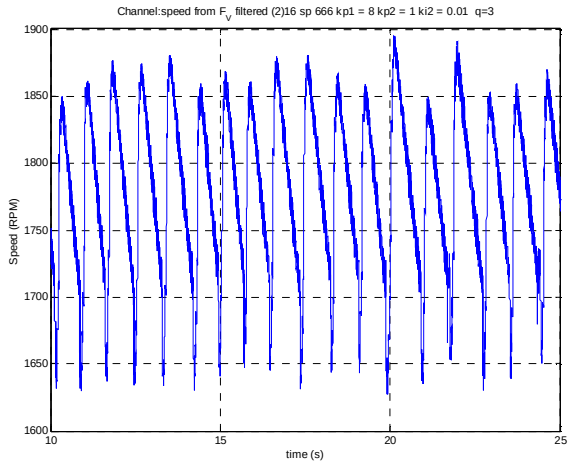


Figure 22 Speed regulation for 3-bit A/D conversion

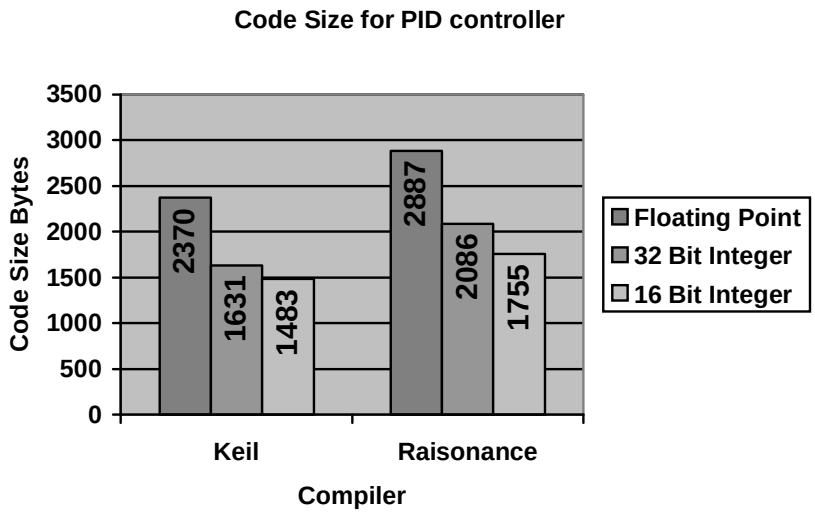


Figure 23 Code size for various implementations of PID controller

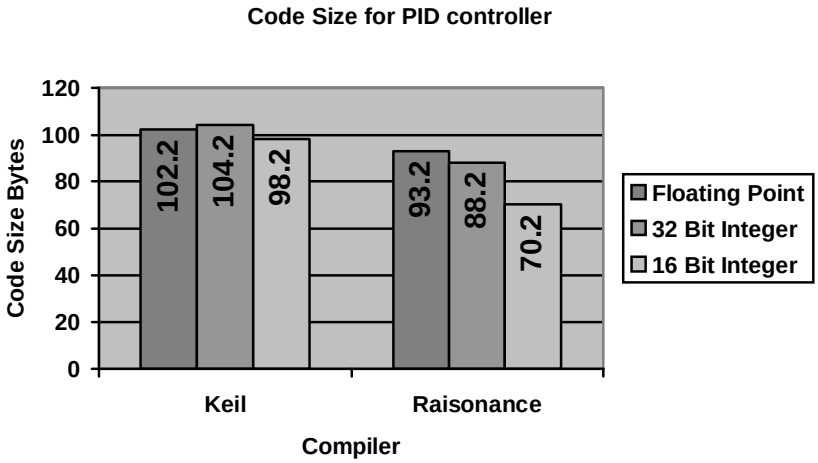


Figure 24 Data size for various implementations of PID controller

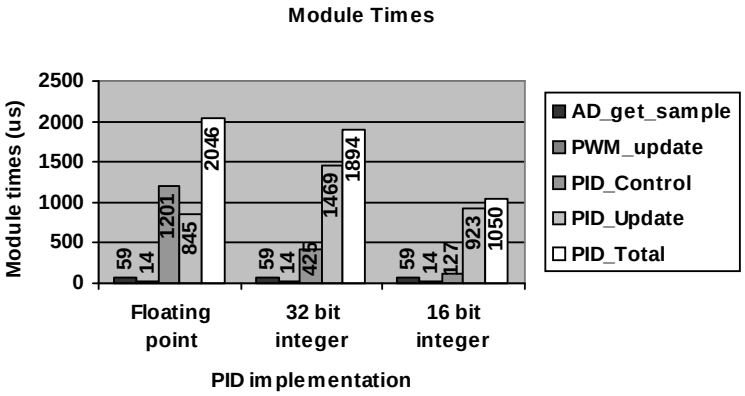


Figure 25 Execution times for program modules (average case)

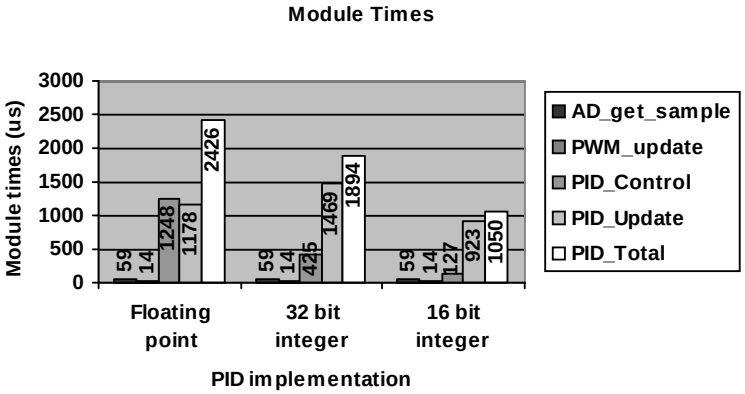


Figure 26 Execution times for program modules (worst case)

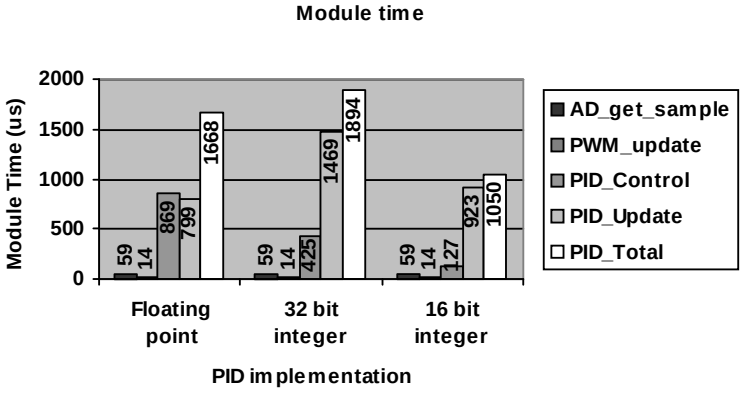


Figure 27 Execution times for program modules (best case)

As we would expect the A/D conversion and control actuation times have a constant value across all implementations. Similarly, the average and worst case execution times for the PID algorithm are related to mathematical complexity. Of interest are the best-case execution times for the floating-point calculations, which are better than the 32-bit implementations. However, these times for the floating point cannot be guaranteed and to ensure correct operation of the controller the worst case scenario has to be assumed..

Of note is the reduction in overall execution time for the 32- and 16-bit integer implementations compared to the floating-point case. The interval of 4 ms (first considered in Figure 7) between sampling (AD_Sample) and actuation (PWM_Update) can be considerably shortened with a 16-bit PID calculation taking only 1.050 ms as opposed to the worst case floating point scenario of 2.462ms. This reduced time between sampling and actuation could be used to improve control performance (Åström, 1997).

7. Discussion and conclusion

For resource usage, the most significant savings occur in the worst-case execution time of the PID algorithm, with the 16-bit integer algorithm implementation requiring only 43% of the execution time of the floating-point version. Reductions in memory are also possible, since a 16-bit integer implementation requires 60% of the code size and 70-80% of the data memory of a floating-point implementation.

Whilst there are reductions in resource usage to be made this is only worthwhile if the performance is not adversely affected. The results in Section 5 demonstrated that control performance is broadly similar in all cases and is not significantly dependant upon PID implementation.

Overall, in the case study described in this paper, we have demonstrated that the critical factor in implementation of a PID controller is not the mathematical implementation of the PID algorithm and that low A/D resolutions have a more significant impact. The mathematical implementation does, however, play a significant part in the microcontroller resource requirements.

8. References

Åström K.J. and Wittenmark B., (1997), “Computer-Controlled Systems. Theory and Design.” Third edition, Prentice Hall, London

Callender A, and Stevenson A B. (1939), “Automatic Control of Variable Physical Characteristics.”, U.S. patent 2,175,985. Filed February 17, 1936 in the United States. Filed February 13, 1935 in Great Britain. Issued October 10, 1939 in the United States

Cosmic, (2004), “C Cross Compiler User’s Guide for Motorola MC68HC08” [Available from http://www.cosmic-software.com/downloads/docs/cx6808.zip]

DSPACE 1 “DaimlerChrysler Relies on TargetLink for Engine Controls” [Available from http://www.dspace.ltd.uk/shared/data/pdf/applex/a4_daimlerchrysler_relies_on_targetlink_for_engine_controls(daimlerchrysler_ag).PDF]

DSPACE 2 “Getting There Faster: TargetLink at Conti Temic“[Available from http://www.dspace.ltd.uk/shared/data/pdf/applex/a5_getting_there_faster_targetlink_at_conti_temic(conti_temic_microelectronic_gmbh).PDF

Franklin G F, Powell J W, Emami-Naeini A, (1994), “Feedback control of dynamic systems” Prentice Hall, London

Franklin G E, Powell J D, and Workman M L, (1995), “Digital control of dynamic systems”, Menlo Park, California, Harlow, Addison Wesley.

Hanselmann H, (1987), “Implementation of Digital Controllers – A Survey” Automatica, Vol. 23 No, 1 pp 7-32 1987, Pergamon Journals

Houpis C.H., Lamont G.B., (1992), “Digital Control Systems”, second Edition, McGraw-Hill

Infineon, (1998), “C515C Users Manual 8-Bit CMOS Microcontroller 11/97”, [Available at http://www.infineon.com/cmc_upload/migrated_files/document_files/Users_Manual/m515c.pdf]

Katz P, (1981), “Digital control using Microprocessors”, Prentice Hall, London

Keil, (2004), “C51/CX51 User’s Guide”, [Available from http://www.keil.com/support/man/docs/c51/]

Kilian C T, (2001), “Modern Control Technology: Components and Systems, Second Edition.”, Delmar Thomson Learning, New York

Lewis D W, (2002), “fundamentals of embedded software: Where C and Assembly Meet”, Prentice Hall, London

Locke C D, (1992), “Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives”, Real-Time Systems, 4(1), 37--53

Matlab 1 “Real-Time Workshop Embedded Coder 3.2: Jaguar Reduces Development Costs with MathWorks Rapid Prototyping and Code Generation Tools” [Available from http://www.mathworks.com/products/rtwembedded/userstories.html?file=5118]

Matlab 2 “Real-Time Workshop Embedded Coder 3.2: Motorola Creates Electric Vehicle Battery Management Controller with Real-Time Workshop Embedded Coder [Available from “http://www.mathworks.com/products/rtwembedded/userstories.html?file=5278]

Maxim (1996), “MAX492, MAX494, MAX495 Single/Dual/Quad, Micropower, Single-Supply, Rail-to-Rail Op Amps” [Available from http://pdfserv.maxim-ic.com/en/ds/MAX492-MAX495.pdf]

Morton T D, (2001), “Embedded Microcontrollers”, Prentice Hall, London

Ogata K, (1995), “Discrete-Time Control Systems”, Prentice Hall, London

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

Raisonance, (2004), “RC51, ANSI-C compiler for 8051”

Slaughter J E, (1964), “Quantization Errors in Digital Control Systems", IEEE Trans. Automat. Contr., vol. 9, pp. 7074, 1964

Stobart R K, (1999), “Modern Control in Diesel engine Management” in “Diesel engine Reference Book, second edition” B Challen and R Baranescu editors, Butterworth-Heinemann, Oxford

Törngren M, (1998), “Fundamentals of implementing Real-time Control applications in Distributed Computer Systems”. Journal of Real-time Systems, 14, p. 219-250. Kluwer Academic Publishers. [Available from http://www.md.kth.se/~martin/Documents/PAPERS/JRTS-paper.ps]

Widrow B, "A study of rough amplitude quantization by means of Nyquist sampling theorem", IRE Trans. Circuit Theory, vol. CT-3, pp. 266-276, 1956

Wijetunge R S, Brace C J, Hawley J G, and Vaughan N D, (2000), "Fuzzy Logic Control of Diesel Engine Turbo charging and Exhaust Gas Recirculation ", CONTROL 2000 UKACC International Conference on Control University of Cambridge 4-7 September 2000 Mini-Symposium on Engine Control Systems (6 September 2000) organised by UKACC member organisations (IEE, IMechE, InstMC) [Available from http://www.bath.ac.uk/%7Eenscjb/FuzzyLogicControl.pdf]

Williamson D, (1991), "Digital Control and Implementation: Finite Wordlength Considerations", Prentice Hall, London

Woodward, (2003) "SG 2D Digital Speed Governor:" [available at http://www.woodward.com/pdf/ic/03258.pdf]

Xiros N I, Kyrtatos N P, (2001), "Diesel Ship Propulsion State-feedback regulation for meeting Hinf-norm requirements", 9th IEEE Mediterranean Conference on Control and Automation (MED'01), Dubrovnik, Croatia, June 27-29, 2001. [Available from http://www.lme.naval.ntua.gr/publications/papers/downloads/2001/med01-162.pdf]

Transaction-Level Modelling and Graphical Simulation of Novel Display Controllers

David Antonio-Torres, Paul F. Lister and Paul Newbury

Centre for VLSI and Computer Graphics, Department of Informatics,
University of Sussex, Brighton BN1 9QH, UK
P.F.Lister@sussex.ac.uk

Abstract. The system-level modelling of a novel display controller is realised using the SystemC environment. Transaction-level (TL) modelling is used as it affords high-speed simulation, easy coding and maintainability. The operation of a basic matrix-addressed display controller is analysed in terms of TL models. The operation of this basic display controller is extended in order to provide compensation for degradation in OLED displays. The compensation scheme is explained and the extended instruction set of the compensating display controller is presented in terms of SystemC transactions. The display controller model is embedded in a graphical user interface (GUI) to produce graphical simulations of testing scenarios and enable visual assessment of the compensation scheme.

Keywords: System-Level Design, Transaction-Level Modelling, SystemC, Display Controllers.

1 Introduction

Transaction-level (TL) modelling is one of the abstraction levels supported by SystemC [1] that enables System-On-Chip (SoC) design. TL has been initially targeted to analyse bus traffic, explore different bus architectures and to enable parallel development of software and hardware in the design flow [2]. Among the main aspects that make TL attractive for system-level modelling are high-speed simulation, fast coding and code maintainability.

Despite being targeted to analyse and simulate complex SoC designs, TL can also be used to model the communication scheme of algorithm-dominated systems. Previous research has shown that display controllers can be described and simulated in terms of behavioural and algorithmic models using SystemC [10]. While these abstraction levels can faithfully represent the processing the data goes through, the simulation speed is drastically reduced because of the low-level details that accompany the description.

It has been shown that high-speed simulation is a must when simulating display systems, where the display controller is the simulation engine, in order to assess the presence of visual artefacts and display behaviour over time [11]. SystemC and TL offer a combined solution for display controller modelling and high-speed simulation. Indeed, behavioural models such as those described in [10] for display controllers can be presented in terms of TL while producing high-speed simulation by raising the abstraction level of the communication architecture of the system.

This paper reports the use of SystemC and TL to model the behaviour of a matrix-addressed display controller with support to compensate for degradation in OLED displays, an emerging display technology. A basic display controller is described as composed of three sub-systems: input section, display memory and output section. The input section offers an interface to receive data and instructions, the display memory is built around a dual-port memory, and the output section refreshes the display with the image stored in the display memory. The behaviour of the basic display controller is enhanced by the inclusion of a compensation scheme that modifies the driving condition of the display based on its estimated degradation.

The paper is organised as follows: section 2 explains OLED degradation, the main problem addressed by the compensation scheme in the display controller, section 3 gives an overview of modelling guidelines and modelling blocks in the TL approach, section 4 presents the operation of a basic display controller in terms of TL models, section 5 shows the TL modelling of a compensation scheme for OLED degradation, section 6 presents the display controller with the embedded compensation scheme as a programmable unit with an instruction set, which is modelled as a set of transactions, section 7 discusses issues related to graphical simulations having TL models as the simulation engine and shows some simulations of the instruction set and section 8 draws some conclusions and outlines future work.

2 OLED Degradation and Compensation

Organic Light-Emitting Diode (OLED) is an emerging display technology aimed at producing light-weight, self emissive, low-power consumption displays, which have been initially targeted at mobile applications. OLED technology initially entered the market with low-content display applications mainly because of the unsolved technology flaws. Gradually the technology has matured until the production of acceptable high-content displays is now possible; however, pixel degradation remains as the main limitation that restrains the technology from being a real challenger to mature technologies, such as liquid crystal, in the mobile applications arena.

OLED degradation is characterised by the gradual and inevitable loss of luminance in the display over time [3, 4] (long-term degradation). Degradation is described in terms of a profile that is determined by the number of hours of display operation. As different materials combine to produce the three primary colours in a full-colour display, three different degradation profiles (and degradation rates) are involved in the loss of luminance of the display, giving rise to a loss of colour balance (colour shift) in the display. The degradation rate is also affected by the level of brightness each pixel is driven to reproduce, thus on a pixel-by-pixel basis different degradation rates (differential aging) can be found.

Extending the lifetime of the materials that produce the primary colours is the ultimate solution to degradation. Red and green materials have already reached a performance adequate for applications such as digital cameras but technology improvements have yet to deal with the lifetime of blue OLED devices, which have the poorest stability over time [5]. Whereas extrapolation measurements can predict good performance of blue devices, further improvements have to take place in order to reach a performance comparable to the other two primary colours and thus challenging mature technologies.

While technology improvements are still attempting to increase lifetimes, solutions to the OLED luminance degradation based on the display controller have been proposed in order to speed up the presence of OLED displays in the display market; however, because of the complexity of the degradation mechanisms they only address partial compensations. A compensation scheme that only deals with long-term degradation by storing different bias current magnitudes with which to drive the display has been proposed [6]; these different current magnitudes can be reprogrammed at any time during the operating life of the display. Related research [7, 8, 9] has effectively reduced differential aging based on optical monitoring of some pixels on the display in order to provide accurate driving that eliminates the effects of differential aging between pixels.

Previous research [11] has proposed a compensation scheme for long-term degradation and differential aging of blue subpixels by modifying their driving conditions; blue devices are targeted because it is expected that in the near future red and green devices will have reached adequate performance for consumer products while blue devices will lag behind. The compensation scheme takes into account the estimated aging of a set of blue subpixels and tries to reduce the degradation effects during their lifetime. The compensation scheme is built around a lookup table (LUT) and a display monitoring unit (DMU). The compensation scheme is embedded in the display controller in order to receive refreshing information from the display driver and thus estimate the aging of the monitored blue subpixels. A typical OLED display controller is modelled using the TL modelling approach and then the compensation scheme is analysed and modelled from its position around the display controller.

3 Transaction-Level Modelling with SystemC

3.1 Modelling Guidelines

The main objective when writing TL models is to produce high-speed simulations while keeping the main TL attributes: communication between sub-systems modelled as transactions and separating functionality from communication architecture.

Among the most important modelling guidelines for high-performance TL modelling are:

- Communication protocol encapsulated in a transaction. Bi-directional data transfer and control are accomplished through arguments of a function call.
- High-level (C++ native) data types are used instead of hardware-oriented ones.
- Passing-by-reference and pointers to data are used within transactions to enable fast and efficient data and control transfers.

- Distinction between master and slave processing elements (PE) can reduce the number of processes that have to be activated in every clock cycle, thus improving simulation performance.

3.2 Modelling Blocks

From a SystemC basis, channels implement communication while modules implement functionality. This concept is sometimes blurred when it comes to talking about hierarchical channels (as opposite to SystemC native channels such as FIFOs, buffers and signals), and modules because both entities are actually the same from the SystemC semantics. However, keeping this specialisation along the modelling flow can help reduce the number of clock-activated processes.

Communication between modules is handled by channels in terms of transactions. The number of transactions a hierarchical channel supports is listed in its interface declaration. A SystemC interface is an abstract class that contains the declaration of the transactions supported by a channel without providing an implementation. A channel has to be derived from a supported interface and provide implementation to all the listed transactions. A hierarchical channel can also contain other channels, SystemC processes and ports.

Modules implement the functionality of the system through SystemC processes and use ports to transfer data and control to other modules. Each port is associated with an interface and invokes methods the interface provides (transactions). The actual execution of the transaction takes place in the channel that provides implementation to the interface.

4 Matrix-Addressed Display Controller

The TL modelling approach has been used to develop a model of a typical OLED display controller. As a prerequisite to this task a behavioural modelling of the components of a display controller has been undertaken to analyse their communication scheme and make the TL models account for it. The display controller model serves as a basis for a compensation scheme for OLED degradation.

4.1 Operation

The refreshing operation of OLED displays is built around a column and a row driver. The refreshing operation takes place on a row-by-row basis, which is known as a matrix-addressed scheme. The row driver selects the row that is to be updated while the column provides the new contents of that row. Usually the two drivers are packaged along with a controller in a single chip. The tasks of the display controller are synchronising the drivers, providing interface and control to the embedded display memory and interfacing the whole system to a microcontroller. Figure 1 shows an algorithmic view of the typical components of an OLED display controller [6, 10].

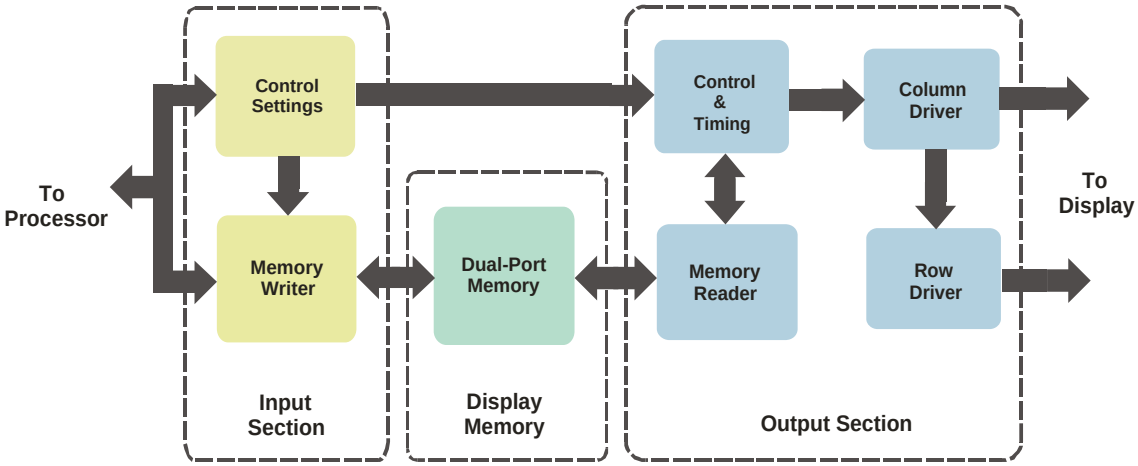


Figure 1: OLED Display Controller

The input section provides an interface for a microcontroller to send data and instructions. Data are forwarded to the display memory via the memory writer. Instructions are translated into registers and flags updates, which govern the operation of the whole display controller (control settings and timing).

The output section embraces the two display drivers, a unit control that is in charge of providing synchronisation between the drivers and an access path to the display memory via the memory reader. The display memory is where the image with which to refresh the display is stored; it can be seen as a dual-port memory that allocates one port to each section of the display controller.

The memory writer and reader can be seen as abstract finite-state machines (FSM) that implement the communication protocol implied in a write and read operation. Each FSM makes use of an address generator, which is a programmable counter that produces valid column and row addresses with which to access the display memory.

4.2 Transaction-Level View

From a TL (and timed functional modelling) view, clock cycles are used to synchronise the data flow between the PEs of the system and annotate the processing time. Thus under this view, processing stages as well as data transfer can execute in one clock cycle without allocating them to any intended architecture implementation.

In the TL model of the display controller, the timing and control units can be implied by clock cycles while annotated processing times are modelled using wait statements. Processing tasks are allocated between wait statements, with which they are executed during a clock cycle.

Distinction between master and slave PEs can be determined by establishing those tasks that trigger the transactions and those that serve them. In Figure 1 the memory reader and the control and timing unit are master PEs because they need to continuously refresh the display regardless of the status of the rest of the system; the input section is a master PE when accessing the memory but is a slave PE when responding to an instruction or data from the microprocessor. As stated before, the control settings and timing and control units can be implied by a global clock and some configuration transactions; hence the memory writer and reader are the only master PEs in the display controller from a TL view.

In order to speed up simulation, slave PEs are not clocked and are made channels that implement transactions via their interfaces. One example of a slave PE is the display memory; the display memory is modelled as a double array where a set of transactions represent the reading and writing operations invoked by the writer and reader. The transactions also imply which port is being accessed.

The display drivers in Figure 1 can be seen as registers that give format to the data that has been read from the display memory. They can be represented by transactions, the arguments of which determine the format the data must have when reaching the display.

Finally, the address generation needed when writing to and reading from the display memory can be modelled as channels that implement the address generation as transactions. Figure 2 shows the operation of the display controller viewed from a TL modelling following the abstraction guidelines explained above.

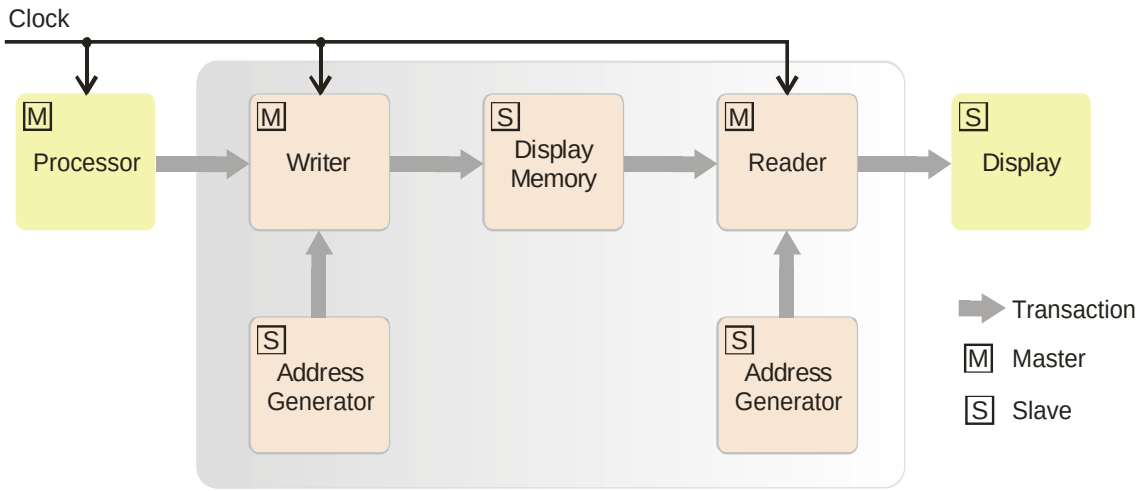


Figure 2: Display Controller at Transaction Level

The processor component in Figure 2 is a master PE that sends data and instructions to the display controller while the display, a slave PE, groups the data the reader sends. The grey box in Figure 2

encloses the functionality of the display controller. As will be shown in section 6, the display controller can be seen as a hierarchical channel whose implemented transactions model the instructions invoked by the processor.

4.3 Transaction-Level Modelling

The transactions that each slave PE implements give shape to the intended functionality of a system from a communication perspective. Each slave PE is now analysed from the transactions they implement.

4.3.1 Display Memory

The display memory is described as a dual-port memory that supports write and read operations in both ports; it is modelled as a port-less hierarchical channel with no SystemC-registered processes. Thus it only executes when its implemented transactions are invoked. These transactions are enclosed by the interface shown in Figure 3.

Each access port is modelled as a set of transactions that enable memory accesses. A memory access requires two sequenced transactions: the operation itself, read or write, and an unlocking operation. The unlocking operation allows the other port to know that the just accessed column and row addresses are now free to be used; partitioning memory accesses into two transactions allow memory contentions to be properly modelled.

```
class dual_port_memory_if : virtual public sc_interface
{
public:
    virtual void unlock_address_1() = 0;

    virtual bool read_1(int & Data, const int & ColumnAddress, const int & RowAddress) = 0;

    virtual bool write_1(const int & Data, const int & ColumnAddress, const int & RowAddress) = 0;

    virtual void unlock_address_2() = 0;

    virtual bool read_2(int & Data, const int & ColumnAddress, const int & RowAddress) = 0;

    virtual bool write_2(const int & Data, const int & ColumnAddress, const int & RowAddress) = 0;
};
```

Figure 3: Implemented transactions of the Display Memory

4.3.2 Address Generator

The address generator returns a valid column and row address to a calling PE, according to its window memory settings: the initial and final column address and its generation direction, i.e. starting with the initial address and counting up to the final or vice versa, and the initial and final row and its generation direction. The window memory settings are handled by functions that are not part of the interface but included as member functions of the address generator channel. As will be shown later, these functions are used to model the instruction set of the display controller. Figure 4 shows the transaction of the address generator.

```
class address_generator_if : virtual public sc_interface
{
public:
    virtual void generate_address(int & ColumnAddress, int & RowAddress) = 0;
};
```

Figure 4: Implemented transactions of the Address Generator

4.3.3 Display

The display is modelled as a channel that implements the transaction that models the operation of the display drivers. This transaction is invoked by the reader when it has finished filling in a row. Capabilities such as image mirroring are modelled in the reader and determine the order that the row is filled and the sequence with which to access the rows of the display. Figure 5 shows the transaction of the display, which carries the selection of the row to be updated and its new contents.

```
class display_if : virtual public sc_interface
{
public:
      virtual void update_row(const int & RowIndex, int * Row) = 0;
};
```

Figure 5: Implemented transaction of the Display

5 Display Controller with Compensation

The compensation scheme implements a compensation model that is based on a degradation model reported in [12]; the degradation model predicts the loss of brightness in an OLED device under certain driving conditions. The compensation model involves monitoring the activity of a number of blue subpixels to keep track of their driving conditions history and then restore the predicted loss of luminance. For its implementation, the compensation model requires a mechanism that keeps track of the activity of the blue subpixels and another mechanism that holds the compensation model. A compensation scheme that fulfils these requirements is presented below.

5.1 Compensation Scheme

Figure 6 shows a compensation scheme for OLED degradation that has been reported in [11]. The DMU is set up to record and use the activity of a number of blue subpixels on the display to estimate their aging and tailor their individual compensation. The aging is estimated based on the refreshing frequency of the display, which is obtained by monitoring the row driver, and the brightness levels that the monitored blue subpixels have been driven to resolve, which are known via the column driver. The lookup table (LUT) holds the compensation model quantised by the levels of the LUT, where the estimated aging is its index. The compensation coefficient of a subpixel retrieved from the LUT is translated into driving conditions by the column driver. Compensating every blue subpixel is accomplished by spreading the aging estimation to non-monitored blue subpixels using interpolation techniques embedded in the DMU.

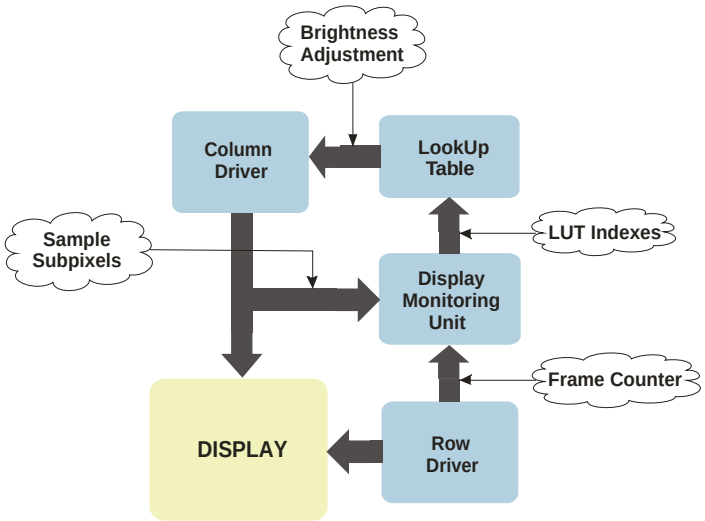


Figure 6: Compensation Scheme for OLED Degradation [11]

5.2 Transaction-Level View

The compensation scheme is driven by the estimated aging of a number of blue subpixels, which in turn is driven by the display contents; hence for its modelling, the compensation scheme is mixed up with the operation of the display and the degradation model. Figure 7 shows the compensation scheme from TL modelling. From a TL modelling perspective, the compensation scheme is represented as a sequence of transactions triggered by the reader when refreshing the display. This sequence gathers all the necessary information to evaluate the compensation model implemented in the LUT.

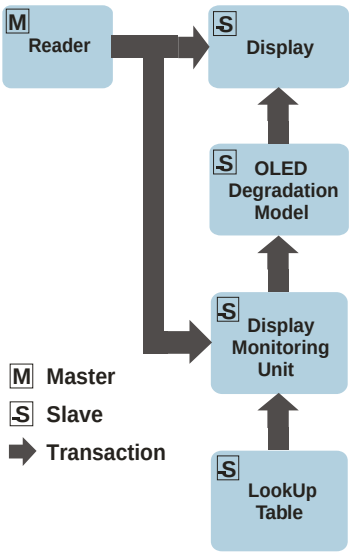


Figure 7: TL view of Compensation Scheme and Degradation Model

Once the reader PE sends the new row information, the display PE requests the degradation model PE, via a transaction, to provide the degradation model to be applied to the current row. To that purpose the display provides the number of times the display has been refreshed as an argument, the age of the display, in the transaction to help estimate the aging level of the blue subpixels monitored by the DMU PE. In turn, the degradation model PE requests the current display row to be compensated by invoking a transaction implemented by the DMU. The DMU returns a compensated row based on the compensation model implemented in the LUT. Communication between the DMU and the LUT is handled by a transaction.

5.3 Transaction Level Modelling

The execution of the compensation scheme and the degradation model are triggered by the display drivers' transaction invoked by the reader, see Figure 5. It is the transaction implemented by the display that activates the sequence of transactions represented in Figure 7. The sequence of transactions that models the compensation scheme is analysed below in terms of the involved PEs.

5.3.1 OLED Degradation Model

Upon invocation of the display drivers' transaction implemented by the display, shown in Figure 5, the display requests the degradation model to be applied to the current display row. The transaction of the degradation model is shown in Figure 8.

```
class degradation_model_if : virtual public sc_interface
{
public:
    virtual void apply_aging(const int & DisplayAge, const int & RowIndex, int * Row) = 0;
};
```

Figure 8: Implemented transaction of the Degradation model

The display provides the simulated age of the display as part of the transaction as well as the current row and its index. The simulated age of the display is needed to calculate the resulting degradation level according to the degradation model. The row index is needed to access a matrix within the degradation model channel that holds the modelled differential aging for each pixel. The degradation is directly applied to the display row provided in the transaction.

5.3.2 Display Monitoring Unit

The execution of the DMU is triggered by the degradation model via a transaction. The degradation model requests the compensation on a pixel-by-pixel basis for the current display row. The DMU returns a coefficient with which to weight the degradation profile in order to increase the brightness of the display [11]. Figure 9 shows the transaction, get_compensation, implemented by the DMU.

```
class compensation_if : virtual public sc_interface
{
public:
    virtual void update_row(const int & RowIndex, int * Row) = 0;

    virtual void get_compensation(const int & RowIndex, const int & ColumnIndex, double & Weight) = 0;
};
```

Figure 9: Implemented transaction of the DMU

Figure 9 also shows the transaction update_row, which allows the DMU to receive the contents of each row in order to tailor the compensation scheme. This transaction is invoked by the reader, as shown in Figure 7.

5.3.3 LookUp Table

The LUT implements the compensation model as a relation between the operating time of a subpixel (index), which involves degradation, and the adequate compensation coefficient (output) to restore the estimated loss of luminance. Both elements of the compensation model are provided in the transaction invoked by the DMU and implemented in the LUT channel. This transaction is shown in Figure 10.

```
class lut_for_blue_if : virtual public sc_interface
{
public:
        virtual void get_output(const int & OperatingTime, double & Output) = 0;
};
```

Figure 10: Implemented transaction of the LUT

6 Display Controller Instruction Set

In the declaration of each of the PEs involved in the TL modelling of the display controller, Figures 2 and 7, public methods have also been included, which play the role of settings for their operation. These methods are not part of the transactions described above because they are not involved in the communication protocol of the display controller or the compensation scheme, but rather provide mechanisms that:

- Determine the possible scenarios under which the display controller may operate.
- Individually set up the operation of the different PEs of the display controller such as settings for the DMU and LUT.

By enclosing the different PEs of the display controller in a large top hierarchical channel, the display controller can be presented as a programmable unit with an instruction set created with the public methods in each PE. The public methods provide the specific functionality of the display controller from a degradation, compensation and operation point of view. The instruction set is enclosed in a SystemC interface, thus each instruction is invoked as a transaction by the processor and forwarded to its corresponding PE to be executed. Below, the instruction set is analysed in terms of degradation, compensation and operation programming

6.1.1 Degradation Programming

Among the settings of the display controller that can modify the degradation model embedded in the compensation scheme is the definition of the lifetimes of the three primary colours, enabling differential aging to be modelled, etc. Instructions to retrieve the current definition of the degradation model are also provided. Figure 11 shows some of the degradation-related instructions.

```
class display_controller_if : virtual public sc_interface
{
public:
    //other instructions not shown//

    virtual void set_longterm_aging(const double & RedLifetime,
                                    const double & RedAlpha,
                                    const double & GreenLifetime,
                                    const double & GreenAlpha,
                                    const double & BlueLifetime,
                                    const double & BlueAlpha) = 0;

    virtual void set_differential_aging(const int & BrightLevels,
                                        const int & MaxAge) = 0;

    virtual void get_differential_aging(int & BrightLevels,
                                        int & MaxAge) = 0;

    //other instructions not shown//
};
```

Figure 11: Degradation-related instructions

6.1.2 Compensation Programming

Different settings are involved in the definition of the compensation scheme: number of monitored blue subpixels, definition of the compensation model, size and settings of the LUT, etc. As there are a number of instructions available, different testing scenarios can be built by the appropriate combination of instructions and settings. Figure 12 shows some of the compensation-related instruction.

```
class display_controller_if : virtual public sc_interface
{
public:
    //other instructions not shown//

    virtual void set_blue_compensation_mode(const int & Mode,
                                            const int & Setting) = 0;

    virtual void get_blue_compensation_mode(int & Mode,
                                            int & Setting) = 0;

    virtual void set_lut_for_blue_operation(const int & Levels,
                                            const int & UpperLifetime,
                                            const int & LowerLifetime) = 0;

    virtual void get_lut_for_blue_operation(int & Levels,
                                            int & UpperLifetime,
                                            int & LowerLifetime) = 0;

    virtual void set_lut_for_blue_location(const int & OperatingTime,
                                           const double & Output) = 0;

    virtual void get_lut_for_blue_location(const int & OperatingTime,
                                           double & Output) = 0;
    //other instructions not shown//
};
```

Figure 12: Compensation-related instructions

6.1.3 Operation Programming

The instruction set provides instructions that modify the operation of the display controller from a hardware simulation perspective. Stopping the operation of a master PE, updating internal flags and registers and stopping refreshing the display are among the operations supported. Communication

instructions such as sending data to the display controller or retrieving its current operating status are also part of this instruction sub-set. Figure 13 shows some of the operation-related instructions.

```
class display_controller_if : virtual public sc_interface
{
public:
    //other instructions not shown//

    virtual bool accept_data() = 0;

    virtual bool send_data(const int & Data) = 0;

    virtual void pause_writer_operation(const bool & Enable) = 0;

    virtual void reset_writer_operation() = 0;

    virtual void set_mirroring_operation(const bool & Horizontal,
                                         const bool & Vertical) = 0;

    virtual void set_reading_window(const int & MinColumn,
                                    const int & MaxColumn,
                                    const bool & ColumnDirection,
                                    const int & MinRow,
                                    const int & MaxRow,
                                    const bool & RowDirection) = 0;

    //other instructions not shown//
};
```

Figure 13: Operation-related instructions

7 Simulation and Results

The simulation of the display controller and the embedded compensation scheme has been further enhanced with a graphical user interface (GUI). The interaction of the GUI with the display controller model is shown in Figure 14 [10]. The GUI provides the instructions and input images and is driven by the simulation of the display controller model to periodically refresh a graphical control that models the operation of the display. As part of this display modelling, the GUI also formats the image it receives from the simulation with some typical subpixel architectures [13-16].

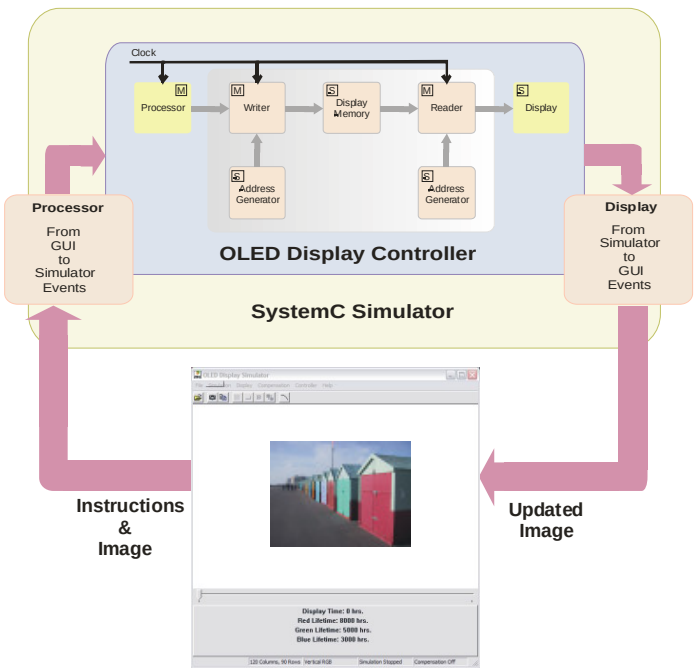
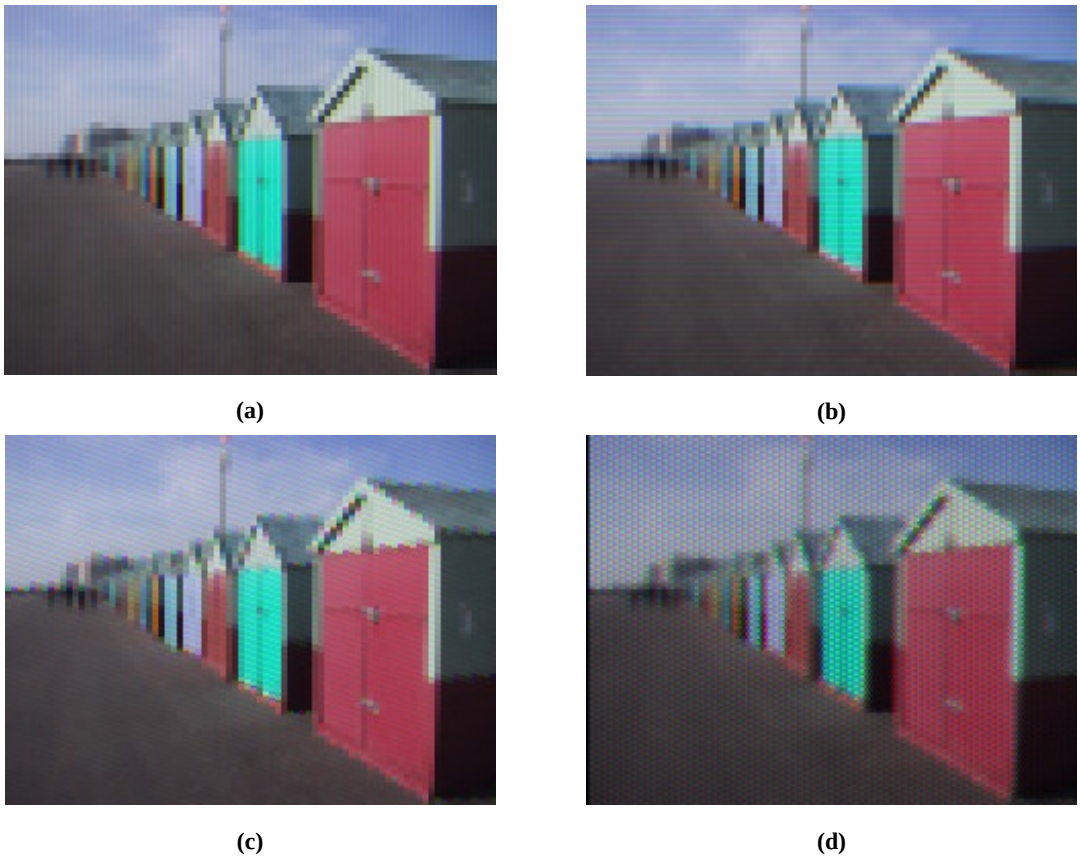


Figure 14: GUI interfacing for the simulation of the display controller and compensation scheme

The GUI allows the creation of testing scenarios for the compensation scheme by: programming different degradation models in the degradation model PE, programming adequate compensation models in the LUT, injecting different sets of test images, etc. Whereas testing scenarios can be implemented using the instruction set of the display controller model, graphical simulations are not supported in system-level design environments. Hence interfacing the simulation with a GUI facilitates the simulation of different display conditions that help evaluate the performance of the compensation scheme as well as enabling an interactive visual assessment.

7.1 Subpixel Architectures

Subpixel architectures are simulated by manipulating the image returned by the display model with predetermined patterns that build a pixel with a certain number of subpixels. Subpixel architecture simulation is an example of the collaborative execution of the SystemC simulator and the GUI. Figure 15 shows four simulated subpixel architectures using a test image.



(a) **(b)**

(c) **(d)**

Figure 15: A test image simulated using the subpixel architecture a) vertical stripe b) horizontal stripe c) diagonal stripe and d) delta triad

7.2 Degradation and Compensation

Figure 16 shows the simulation of degradation and compensation of two test images rendered with the vertical stripe pattern, which is shown in Figure 15a. The degradation and compensation models in both simulations are the same; however, these simulations show how the compensation scheme is dynamically driven by the image. It can be seen from Figure 16 that the degradation effects are essentially removed by the compensation scheme in both test images.

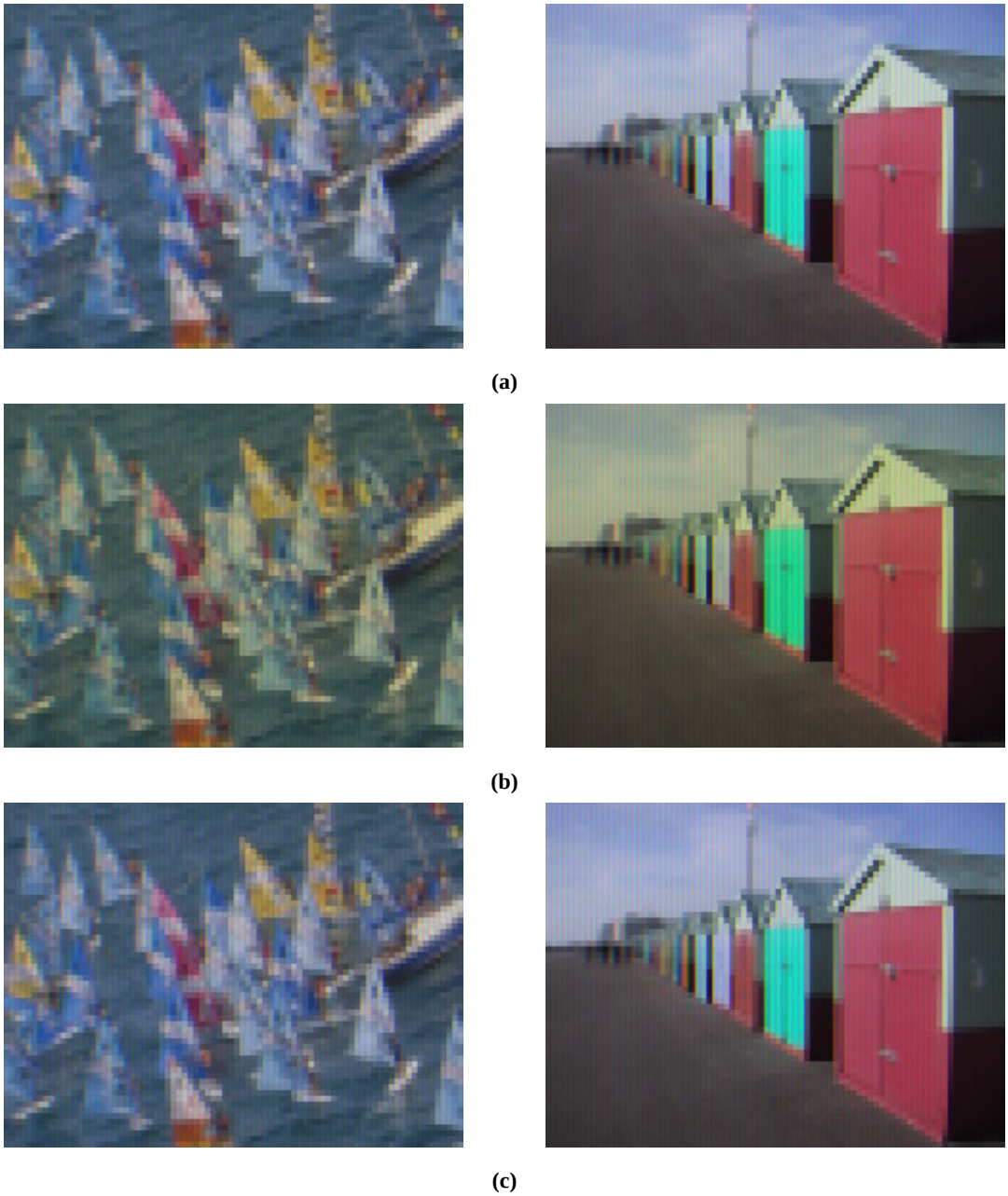


Figure 16: Simulation of degradation of a) two test images at b) 1000 hours with no compensation and c) 1000 hours with compensation

8 Conclusions and Future Work

The modelling of a display controller that provides compensation for degradation in OLED displays has been shown in terms of TL models. It has been shown how a TL modelling approach can be used to translate a low-level description of an OLED display controller into high-speed simulation TL models (up to four QVGA frames/second for PC simulations). Likewise, the compensation scheme reported in [11] has been analysed and modelled using a transaction-oriented approach.

From a TL perspective the processing elements (PE) involved in the display controller modelling were classified as master and slave PEs in order to reduce the number of processes activated in each clock cycle and thus speeding up the simulation. This led to a model of the display controller with two master PEs, the reader and writer, and four slave PEs: two address generators, the display memory and the

display, while the compensation scheme was modelled as a sequence of transactions implemented by slave PEs.

The operation of the whole display controller has been characterised by its instruction set. The instruction set is modelled as a set of transactions, where a transaction invocation is forwarded to one of the PEs for the involved operation to take place. The instruction set is focused on three aspects of the display controller modelling: the degradation model, the compensation scheme and its hardware functionality.

Simulation of the display controller and the compensation scheme has been leveraged by the interfacing of a GUI. The GUI allows the graphical assessment of testing scenarios, which are built by combinations of test images and the definition of degradation and compensation models using the display controller instructions set.

Some graphical simulations have been shown, where the collaborative execution of SystemC and the GUI have been proven by simulations of subpixel architectures and the performance of the compensation scheme have been verified using two test images. Several simulations of testing scenarios are expected to be run to 1) reach a better understanding of the relation between degradation and display usage and 2) evaluate the performance and limits of the compensation scheme.

It can be concluded that a system-level modelling approach, using SystemC, has played a key role in analysing OLED degradation and developing a compensation scheme embedded in the display controller. Another benefit of using SystemC is that the models can be executed to verify the correct operation of the display controller and assess the performance of the compensation scheme.

9 References

[1] SystemC website: http://www.systemc.org.

[2] Thorsten Grotker, Stan Liao, Grant Martin and Stuart Swan, *System Design with SystemC*, Kluiwer Academic Publisher, 2002.

[3] Gopalan Rajeswaran and Kathleen M. Vaeth, *Fundamentals of OLED Displays*, Society for Information Display, Short Course S-4, June 2001.

[4] Gu Xu, “Fighting OLED Degradation”, *Information Display*, vol. 19, no. 6, pp. 18-21, June 2003.

[5] M. Leadbeater et al., “Blue LEP-OLED with 30, 000 hours Lifetime for Flat-Panel Display Applications”, SID 04 Proceedings, pp. 162-165.

[6] *MXED 301: 128-Column by 80-Row OLED Driver with Controller*, Clare Micronix Inc., June 2002.

[7] P. Salam, “OLED and LED Displays with Autonomous Pixel Matching”, SID 01 Digest, pp. 67-69.

[8] E. T. Lisuwandi, “Feedback Circuit for Organic LED Active-Matrix Display Drivers”, Massachusetts Institute of Technology Master Thesis, May 2002.

[9] D. Fish et al., “Improved Optical Feedback for AMOLED Display Differential Aging Compensation”SID 04 Proceedings SID, pp. 1120-1123.

[10] D. Antonio-Torres P. F. Lister and P. Newbury, “System-Level Verification of an OLED Display System Using SystemC” IEE Seminar on SoC Design, Test and Technology, pp. 13-18, 2003.

[11] D. Antonio-Torres P. F. Lister and P. Newbury, “Modelling of a Compensation Scheme for OLED Degradation” SID 04 Proceedings, pp. 1124-1127.

[12] P. Kobrin et al., “Time Dependance of OLED Luminance”, SID 03 Proceedings, pp. 539-541.

[13] R. Samadani, J. Lanham, D. Loomis, L. D. Silverstein, J. Larimer, “Color Tilings for Flat-Panel Displays”, SID 93 Digest, pp. 83-86.

[14] P. G. J. Barten, “Resolution of Liquid-Crystal Displays”, SID ’91 Digest, pp. 772-775.

[15] K. G. Freeman, “Simulation of Liquid Crystal Colour TV Displays”, Proc. EuroDisplay ’90, pp. 116-119.

[16] L. D. Silverstein, et al., “Effects of Spatial Sampling and Luminance Quantization on the Image Quality of Color Matrix Displays”, J. Opt. Soc. Am. A, vol. 7, no. 10, pp. 1955-1968, October 1990.

A test bed for evaluating and comparing designs for embedded control systems

Tim Edwards[1], Michael J. Pont[1], Pete Scotson[2] and Steve Crumpler[2]

[1]Embedded Systems Laboratory, University Of Leicester, University Road, Leicester LE1 7RH, UK. Tel: 0116 2522873 Fax: 0116 252 2619
http://www.le.ac.uk/embedded/

[2]TRW Conekt, Stratford Road, SOLIHULL, B90 4GW, UK
http://www.conekt.net/

Abstract

In this paper, we discuss the design and evaluation of a desktop test bed, intended to allow for the rapid comparison of a range of possible designs for distributed embedded control systems. The test bed is based around an inverted pendulum. Such a system is inherently unstable and provides a demanding control task, with a simple – clearly visible – indication of system performance. One significant feature of the present test bed is the support for system redundancy, in the form of duplicated sensors and motor drives. The design and construction details for the pendulum hardware are presented in this paper, followed by test results for both single-processor and multi-processor embedded control systems.

1. Introduction

The first fly-by-wire test flights took place over 30 years ago (NASA, 1999). The systems were developed for use in military aircraft; however, the benefits of implementing similar systems in commercial aircraft were considerable. Overall weight could be reduced considerably due to replacement of heavy mechanical linkages, re-configuration became much easier which helped reduce development costs and passenger comfort could be improved through automation of processes such as turbulence suppression (Pratt, 2000). Fly-by-wire technology is now widely used in military and civil aircraft as well as space shuttles (Ladkin, 1995; Krahe, 1999)

The automotive industry is based around mass-production and therefore future drive-by-wire systems should involve the minimum of hardware to keep the cost per unit low. However, there are regulations (SAE, 1993) that state that no single component failure, in a safety-critical system, should cause a loss of service at the application level. This means that some hardware redundancy will be a requirement for commercial drive-by-wire systems.

As part of joint project with the University of Leicester and TRW Conekt the effect of software and network architectures on fault-tolerance capabilities is being investigated. The findings of this work are intended to aid designers of future drive-by-wire applications.

All experimentation for this project is being done through the use of desktop demonstration systems, suitable for use on a laboratory workbench. The aim is to tackle issues such as the detection of faulty buses, nodes, sensors and actuators, and to continue operating our system without performance degradation.

The first test bed to be used in this project is based around an inverted pendulum. Such a system is inherently unstable and provides a demanding control task, with a simple – clearly visible – indication of system performance. One significant feature of this test bed is the support for system redundancy, in the form of duplicated sensors and motor drives.

In this paper we describe the design and development of the test bed. Section 2 introduces the inverted pendulum problem, and describes the design and implementation of the test bed in terms of both hardware and software. Section 3 focuses on the results from single-processor control of the test bed. In Section 4 a networked controller using 3 processors is introduced along with

experimental results using the rig. Section 5 concludes the paper, with a summary of the test bed and the results obtained to date.

2. Design and implementation of the test bed

To study fault-tolerance in X-by-wire systems, a test bed is to be used that will allow different architectures to be evaluated and compared in a laboratory environment. In this section, the theory of the inverted pendulum system, that the test bed is based on, is presented. Details of the design and implementation of the physical rig and the software controller are also included here.

Figure 1 shows two representations of an inverted pendulum restricted to one plane of movement, and mounted on a cart that can only move in the same plane. In Fig 1**a** the pendulum is falling to the right due to gravity, and in Fig1**b** a force applied to the cart is correcting the angle of the pendulum.

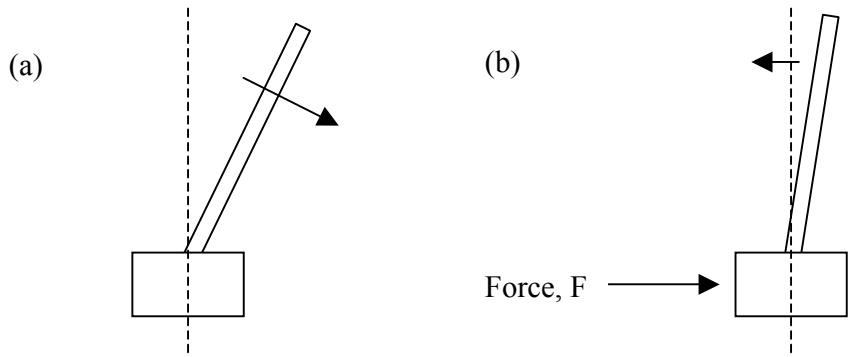


Figure 1 – *Two illustrations: (a) An inverted pendulum falling over; (b) An inverted pendulum being stabilised by applying a force to the cart at its base.*

The inverted pendulum can be considered to be a single-input, single-output system, where the only measured output will be the angle of the pendulum, and the only input through an actuator motor that will drive the cart.

2.1 Underlying theory

One reason for selecting the inverted pendulum as a test bed is that this is a well-studied control problem (e.g. Dorf and Bishop, 2001; Franklin *et al.*, 2002; Ogata, 1995). In this section a linear model of the system is discussed.

Figure 6 is a detailed figure of an inverted pendulum and cart system. Where, θ, denotes the angle, from vertical, of the pendulum and u, a correcting force on the cart. The total mass of the cart, $M = M_1 + M_2$.

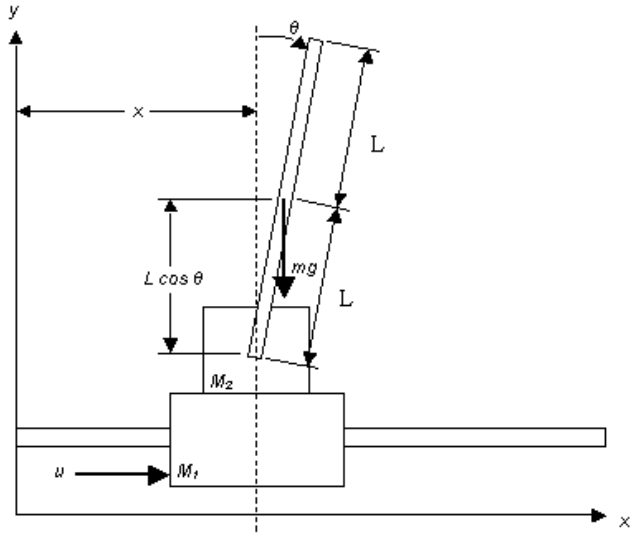


Figure 6 - *Model of inverted pendulum and cart*

A downward force is shown acting on the pendulum at its centre of gravity, the (x, y) coordinates at this point are (x_G, y_G). Therefore:

$$x_G = x + L \sin \theta$$
$$y_G = L \cos \theta$$

The rotational motion of the pendulum rod about its centre of gravity can be described by

$$I \cdot \frac{d^2 \theta}{dt^2} = V L \sin \theta - H L \cos \theta \qquad \textbf{(1)}$$

where, I, is the moment of inertia of the rod about its centre of gravity. H, is the horizontal motion of the pendulum rod's centre of gravity and is described by

$$m \cdot \frac{d^2}{dt^2} (x + L \sin \theta) = H \qquad \textbf{(2)}$$

V is the vertical motion of the pendulum rod's centre of gravity and is described by

$$m \cdot \frac{d^2}{dt^2} (L \cos \theta) = V - mg \qquad \textbf{(3)}$$

The horizontal motion of the cart is described by

$$M \cdot \frac{d^2x}{dt^2} = u - H \qquad \textbf{(4)}$$

Equations (1) through (4) describe the motion of the inverted-pendulum-on-the-cart system. The sin θ and cos θ terms, mean that these are non-linear equations.

If we assume the angle θ is small, then equations (1) through (4) may be linearised as follows:

$$I \cdot \frac{d^2\theta}{dt^2} = VL\theta - HL \qquad \textbf{(5)}$$

$$m \cdot \left(\frac{d^2x}{dt^2} + L \cdot \frac{d^2\theta}{dt^2} \right) = H \qquad \textbf{(6)}$$

$$0 = V - mg \qquad \textbf{(7)}$$

$$M \cdot \frac{d^2x}{dt^2} = u - H \qquad \textbf{(8)}$$

From Equations (6) and (8), we obtain

$$(M + m) \cdot \frac{d^2x}{dt^2} + mL \cdot \frac{d^2\theta}{dt^2} = u \qquad \textbf{(9)}$$

From Equations (5) and (7), we have

$$I \cdot \frac{d^2\theta}{dt^2} = mgL\theta - HL$$

$$= mgL\theta - L \cdot \left(m \cdot \frac{d^2x}{dt^2} + mL \cdot \frac{d^2\theta}{dt^2} \right)$$

or

$$(I + mL) \cdot \frac{d^2\theta}{dt^2} + mL \cdot \frac{d^2x}{dt^2} = mgL\theta \qquad \textbf{(10)}$$

Equations (9) and (10) describe the motion of the inverted-pendulum-on-the-cart system.

Together these equations constitute a mathematical model of the system.

2.2 Design and construction of the pendulum

To begin the design of the pendulum, we considered designs for similar systems produced at MIT (Stimac, 1999), the Technical University of Denmark (Jantzen, 1998), and the University of Leicester (Lim, 2001; Wong, 2002). None of these designs could be used directly as they did not incorporate hardware redundancy, a key requirement for this project. In the end, we opted for a simple mechanical design, loosely based around the existing rig from the University of Leicester.

Figure 1 demonstrated how movement of the cart on which an inverted pendulum is mounted, can correct the angle of the pendulum. The cart for this test bed is made from Perspex and has two sets of linear bearings mounted through it. The main base for the test bed is a 1m track made from tubular steel rails. The linear bearings in the cart allow it to move smoothly along these rails. **Figure 2** is a photograph of the complete test bed.

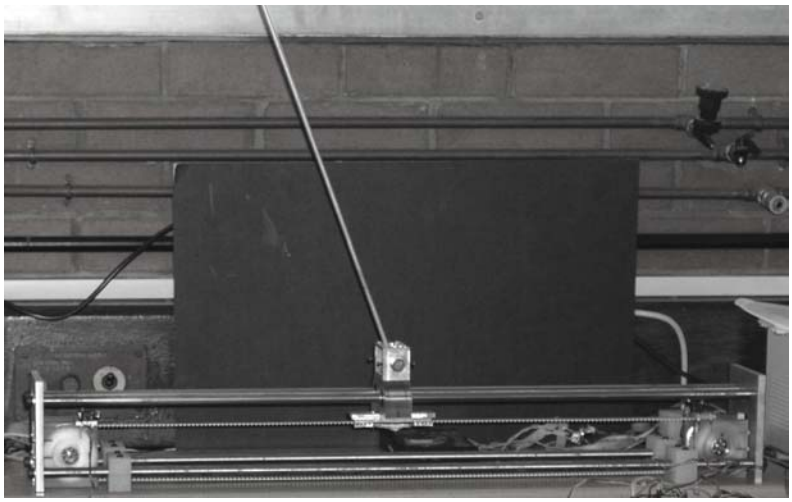


Figure 2 - *The inverted pendulum rig*

There are two main requirements for the actual pendulum, (1) it must be able to move freely and (2) its angle needs to be monitored. Rotary potentiometers are variable resistors that are adjusted by angular motion. Our pendulum is mounted across the shafts of two such potentiometers (see **Figure 3**). As the resistance of the potentiometers change, the voltages across them vary according to the potential divider rule (see **Figure 4**). Reading the voltage, V_{ref}, across the potentiometers through an analogue-to-digital converter allows an approximation to the pendulum angle to be made.

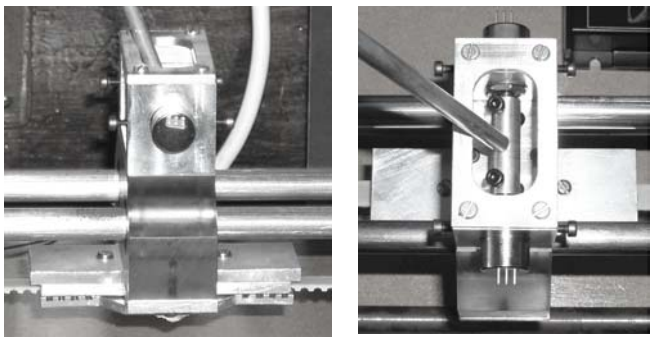


Figure 3 – *Photos of the cart illustrating potentiometer placement. (left) front view, (right) top view.*

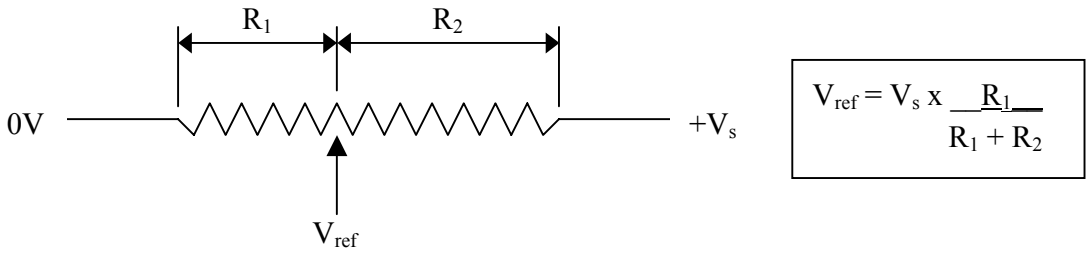


Figure 4 – *Circuit diagram of a potentiometer and the potential divider equation*

A pulley is mounted at each end of the track, supporting a drive-belt that is secured to the base of the cart. A micro-switch is fixed at each end of the track to disable the system if the cart travels too far in either direction. 50W brushless DC motors were chosen to drive the pulleys as they are capable of switching direction rapidly and have low friction when not operational (this is an important feature for later experiments where duplicate actuator motors will be used). The nature of the motors does mean that some dedicated feedback control is needed to maintain set speeds. The motors have integrated hall-effect sensors and an optical encoder to allow for this control process.

To maintain a focus on pendulum rather than motor control, dedicated amplifiers (Maxon Motors DES 50/5) have been integrated into the current version of the test bed. The amplifiers respond to speed commands sent over an RS-232 or CAN link and then maintain that speed without further interaction.

There were two problems with the original test bed design. The first was due to a rigid connection between the pulleys and motors. After some sustained use, the high frequency direction changes had caused to the motors to work loose, eventually rendering the rig inoperable.

The test bed now has the track and motors mounted separately to a baseboard, and the connection between motors and pulleys is done with flexible couplings (see **Figure 5**). The test bed is now quieter and considerably more robust.

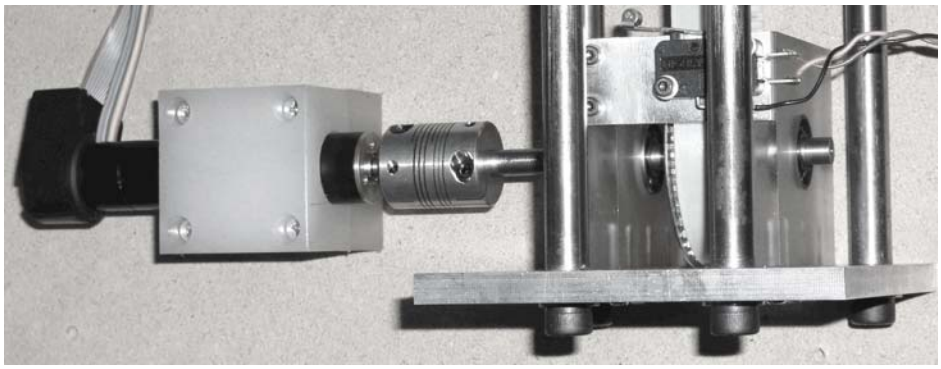


Figure 5 – *Photos of an actuator at one end of the track.*

The second problem was due to the considerable friction in the two wire-wound potentiometers at the base of the pendulum, at low speeds the pendulum could actual become stuck. This has been overcome by replacing them with a pair of non-contacting magnetic potentiometers. These act in the same way but the friction is now minimal.

2.3 Implementation of the control system

Any computer-based control system will be operating at discrete intervals of time, called sample intervals (Ogata, 1995). To regulate this sampling frequency a time-triggered co-operative scheduler (Pont, 2001) was used.

Time-triggered co-operative schedulers (TTCS) can be ideal for digital control applications. Time-triggered means that tasks are executed only in response to the progression of time and not any other external events which may prove unpredictable. Further, the co-operative nature of the scheduler means tasks cannot interrupt (or pre-empt) each other; together these properties make the timing of control actions reliable and predictable.

All the controller implementations described in this paper split the control process into three tasks that are scheduled individually. **Figure 7** illustrates a simple scheduler cycle to control the inverted pendulum test bed, where four task intervals are allowed to each sample interval.

Task 1 – Takes the average of three readings of the sensor position.

Task 2 – Uses the latest sensor data to calculate the next control step using a PID algorithm.

Task 3 – Transmits a new actuator command to the amplifier over a CAN bus.

Task 4 – Transmits system data over RS-232 for diagnostics and performance evaluation.

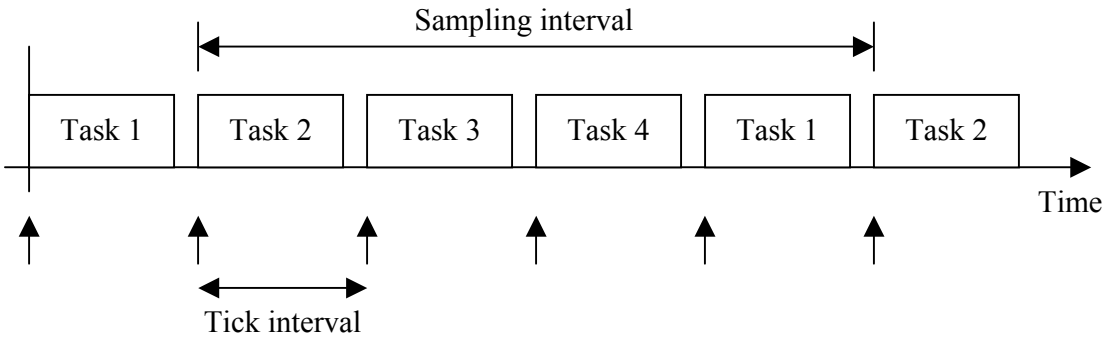


Figure 7 – *Illustration of the task scheduling for basic PID pendulum control*

Task 1 – Sampling

The inverted pendulum is physically limited to move only +/- 20 degrees from vertical, to avoid it getting caught up in the track and/or drive system. The potentiometers at the base of the pendulum are sampled through an analogue-to-digital converter, the 40 degrees of movement equates to range of ~120 steps through the ADC. This means a change of 3 in the ADC reading equates roughly to 1 degree of movement. The sampled data can be susceptible to noise though, with the pendulum fixed in position the ADC readings can fluctuate by +/- 0.66 degrees.

To reduce noise in the sampled data, each time the sample task is called it actually samples three times and takes an average value. The sampling task also has limits set which the data should be unable to exceed, if it appears that data is outside this range it can be ignored or in the current implementation an error is flagged and the system shut down. It will be possible to add further input checks later such as a maximum rate of change between samples, again if this was exceeded it would indicate that an error had occurred.

Task 2 - Calculation

A Proportional Integral Derivative (PID) control algorithm was used in this test bed. The PID control algorithm developed by Callender and Stephenson (1939) and remains in widespread use.

Indeed, it is estimated that between 50% (Ogata, 1995) and 90% (Franklin *et al.*, 1995) of all control systems are based on this technique.

Computationally, PID is a simple algorithm. There are number of forms it can take: the main body of the calculation task used in this implementation is shown in **Code Listing 1**.

```
// Proportional term
signed float Control_new = (PID_KP * Error);

// Integral term
Sum_G += Error;
Control_new += PID_KI * Sum_G;

// Differential term
Control_new += (PID_KD * SAMPLE_RATE * (Error - Old_error_G));
```

Code Listing 1 – *A simple PID algorithm (Pont, 2001)*

Varying the three gains, the proportional gain, PID_KP, the integral gain, PID_KI, and the derivative gain, PID_KD, allows the system performance to be tuned to meet the system requirements.

Task 3 – Actuation

The main responsibility of this task is to relay the latest control command to the actuator(s). This is done in a separate task from the control calculations to make the timing more deterministic. In the practical implementation, dedicated amplifiers are used to control motor speed and so this task will transmit a new speed command to amplifier over a CAN bus.

3. Using the test bed: Single-processor implementation

This section describes a single-processor solution for controlling the inverted pendulum rig. Details of the hardware and software architectures are provided and logged data is used to illustrate system performance.

3.1 System Overview

Figure 8 shows a block diagram schematic of the control loop for a single-processor implementation.

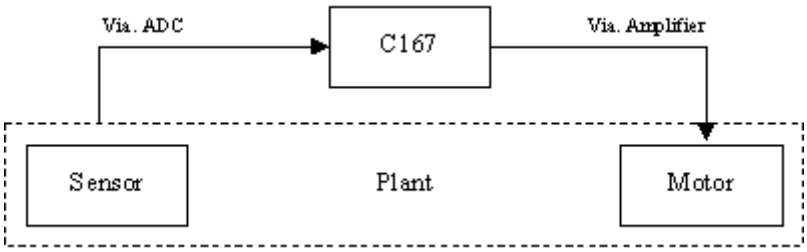


Figure 8 – *Simple illustration of the flow of data with a single C167 controller.*

The C167 is a 16-bit micro-controller, widely used in the automotive industry, and we have chosen to use them throughout these tests. A time-triggered co-operative scheduler is implemented as illustrated in **Figure 7**. An internal timer regulates the tick interval, by overflowing and generating an interrupt at pre-defined intervals. Initially the tick is set at 1ms, meaning the sample interval will be 4ms, but this can be adjusted later, by changing the timer settings.

3.2 Tuning System Performance

The performance of the inverted pendulum is dependent on a range of variables; most important are the sampling frequency and the PID gain values. **Figure 9** illustrates some of the consequences of varying the sampling frequency.

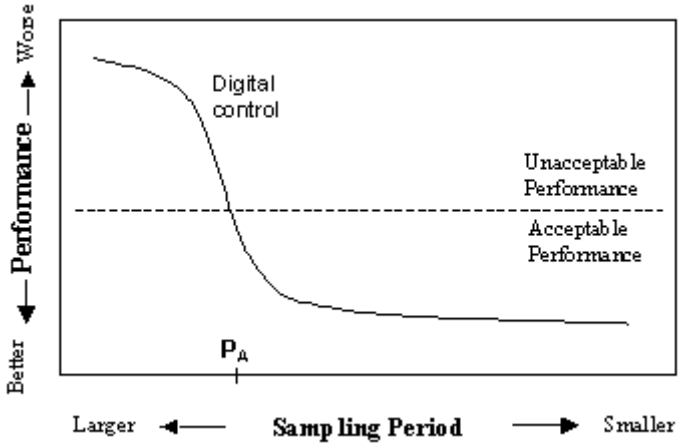


Figure 9 - *Performance of digital control against sampling frequency (Adapted from Lian et al., 2002).*

The worst, unacceptable, acceptable, and best regions can be defined based on control system specifications such as overshoot, steady-state error, and/or phase margins.

In the majority of control applications the highest possible sampling frequency, f_s, is preferable. However, in many cases the sampling period can be lengthened up to a certain bound within which stability is guaranteed in spite of performance degradation. This certain bound is called a maximum allowable delay bound (MADB) and corresponds to a period $P_{A,}$ as shown in **Figure 9** (see Park 2002 for details on calculating the MADB).

After experimenting with a variety of sampling frequencies both in simulation and on the rig, 4ms has been kept as the sample frequency. This value lies in the region of acceptable performance but is relatively close to the MADB.

To illustrate control performance it has been decided to start the pendulum from rest (i.e. at the maximum of +/- 20 degrees from vertical), this effectively shows the controllers response to a disturbance. To begin tuning the controller parameters only the proportional gain will be used. However, the error at start-up will be very large meaning only a small proportional gain is required initially. After starting up the error will tend to be much smaller, requiring a larger proportional gain. **Figure 10** shows the pendulum angle and actuator commands when only Kp is used, set at 350. This figure clearly shows that although the pendulum is righted initially, control is quickly lost.

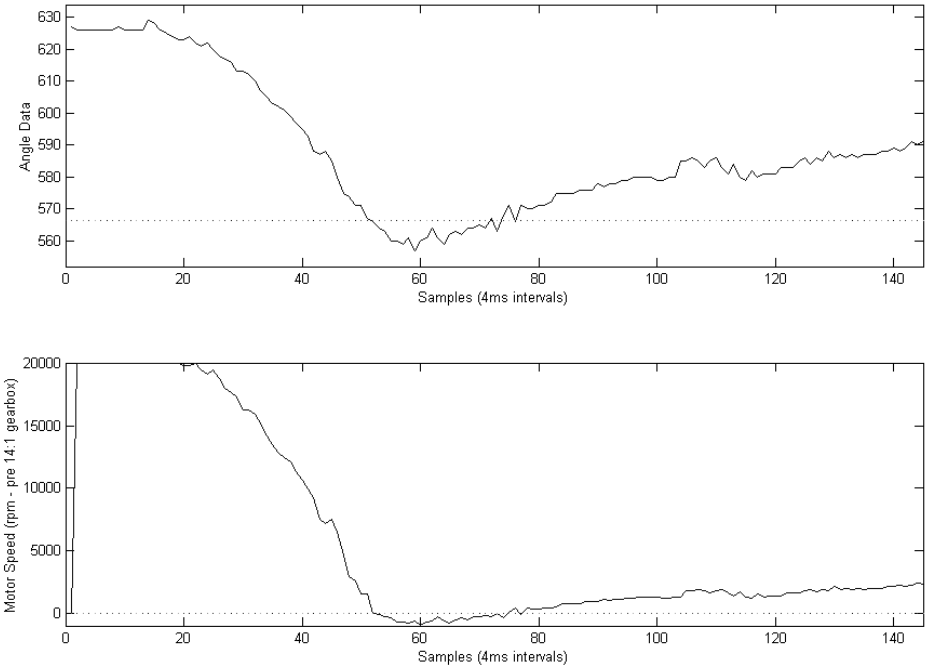


Figure 10 - *Plots of angle error and actuator commands. Loss of control after start-up, when Kp = 350.*

A simple way to solve this problem is using one set of gains at start-up until a threshold level is reached (in this case when the pendulum first becomes vertical, i.e. error equal to zero), then switching to another set of gains for better steady-state control. As Kp = 350 has been shown to be sufficient to start the system it will be kept the same for all further tests described in this paper. Setting a second proportional gain much greater then the first can keep the pendulum from falling back over completely but instead it causes the pendulum to oscillate. **Figure 11** shows plots of the pendulum angle and the actuator command in this situation.

Reducing the proportional gain and introducing small integral and differential gains can stabilise the pendulum. **Figure 12** shows logged data recorded with Kp = 350, Ki = 3, and Kd= 20.

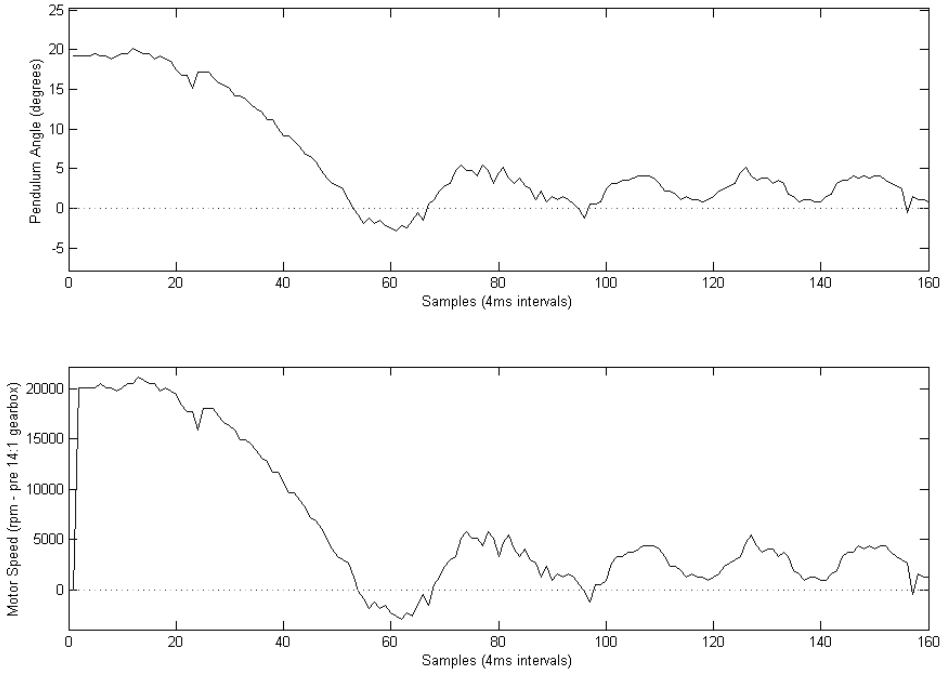


Figure 11 – *Plot of angle error and actuator commands. Oscillations in response to single-processor P-only control, where Kp = 2000.*

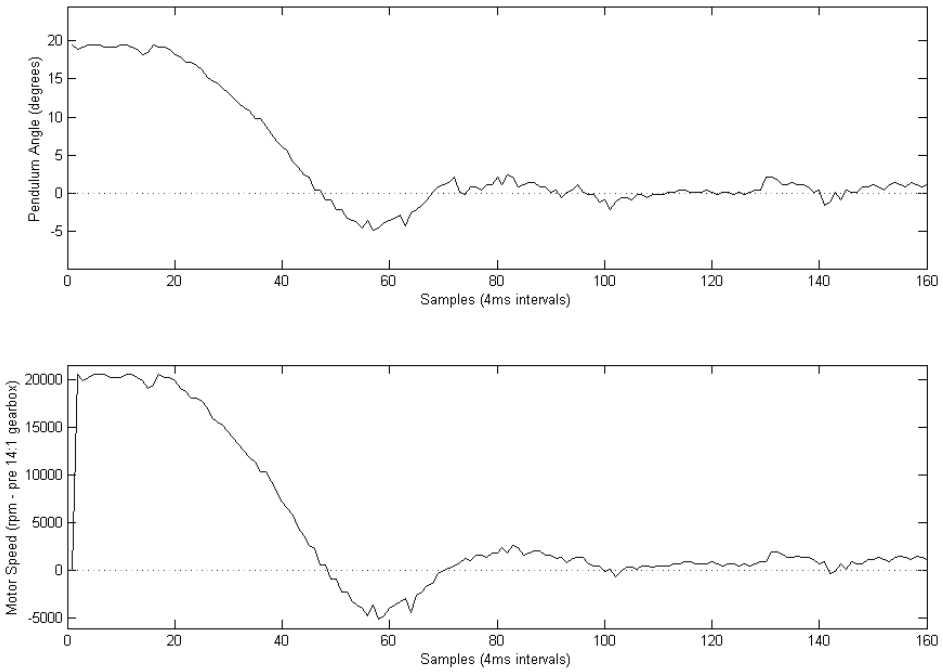


Figure 12 - *Plots of pendulum angle and actuator commands. Inverted pendulum under single-processor PID control, where Kp = 350, Ki = 3 and Kd = 20.*

3.3 Summary

A "single-processor" controller for the inverted pendulum test bed has been described. A C167CS running a time-triggered scheduler is responsible for the timing of all control tasks, and a simple PID algorithm is used to calculate a suitable control input.

At a sample frequency of 4ms it is possible to obtain adequate system performance from the test bed. Data has been logged and video clips recorded to act as a record a base level of performance.

Later stages of this project will be focused on distributed control systems more similar to those found in real X-by-wire applications, as such the following section will demonstrate basic control using such an arrangement. This will be the true measure of base level performance.

4. Using the test bed: Multi-processor implementation

Single-processor control of the inverted pendulum test bed has already been demonstrated. However, the X-by-wire systems that this project aims to address do not use this style of centralized control, instead, control is distributed over a number of processors.

Each processor will still run a time-triggered co-operative scheduler similar to that described in the Section 2. To keep the nodes synchronized all timing will be done using one clock that is connected to the master node. This architecture is known as Shared-Clock Scheduler (Pont, 2001).

Every scheduler tick on the master node triggers a "tick" message to be sent out over the CAN bus. When the "slave" nodes receive this message, they in turn generate an interrupt that drives their own scheduler. The current implementation of the shared-clock scheduler involves 5 data bytes being sent with every message. The first of these bytes represents the ID of one slave node, the named slave will then reply to the master with an "ack" (acknowledgement) message. This mechanism allows the master to track whether all slaves are operational and to relay data between nodes.

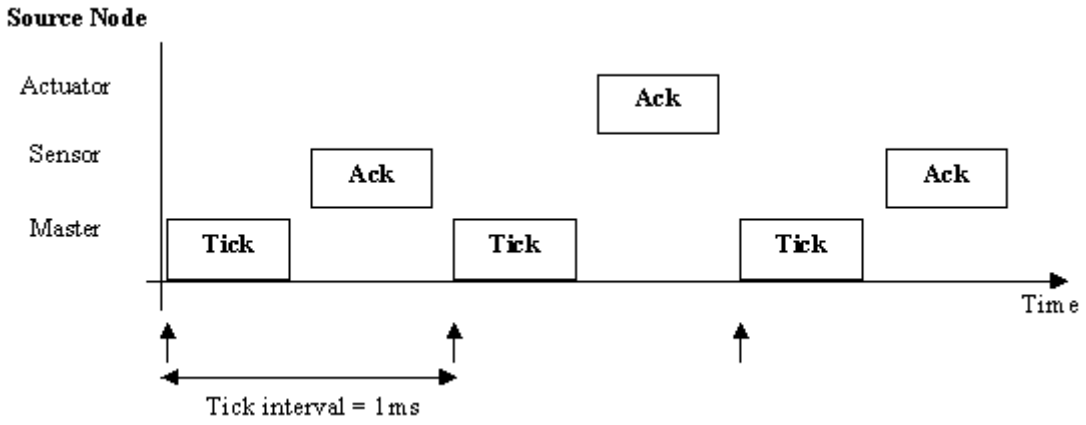


Figure 13 – *Message schedule illustration for simple distributed control of the inverted pendulum rig.*

Figure 13 illustrates one and a half cycles of this message schedule. Every acknowledgement message from the sensor node contains the latest input data while actuator commands are included in the tick messages for the actuator node. The remaining data bytes are currently unused to allow further information to be added later during more complex fault-insertion tests.

The basic network structure for a three-node distributed design is illustrated in **Figure 14**. In this design, the Master node runs the control algorithm and periodically communicate with the Slaves; the sensor node takes on the responsibilities for sampling and averaging the sensor data, and the actuator node relays the commands from the Master to the amplifier. Additionally the sensor node sends the status of the micro-switches at either end of the track back to the Master.

All communications between nodes is carried out using the CAN protocol, currently used extensively in automotive applications.

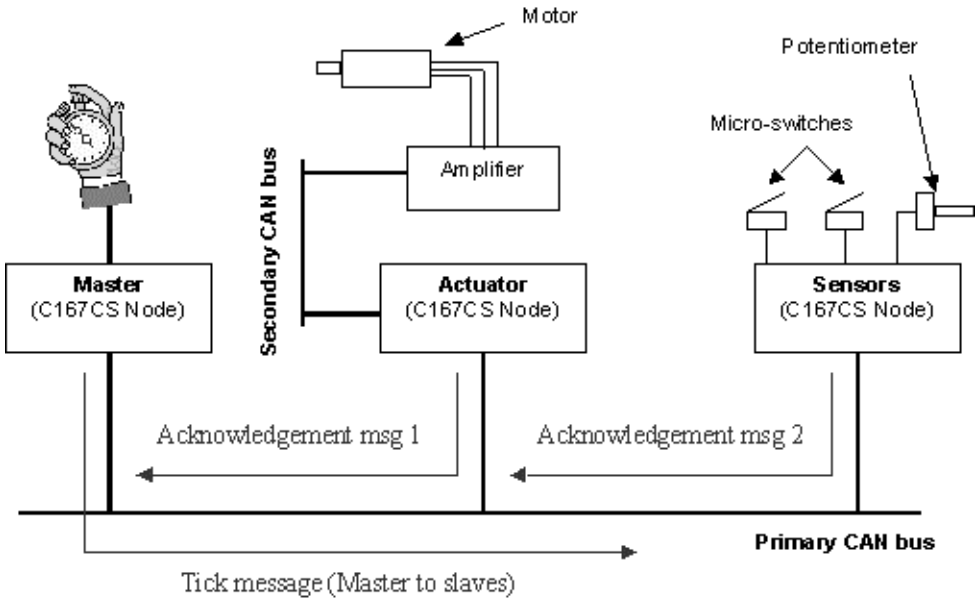


Figure 14 - *The basic inverted pendulum control network*

4.1 Measuring System Performance

Figure 15 shows logged data using the multi-processor architecture described above. For these tests the sample frequency and gains were kept the same as for the single-processor controller.

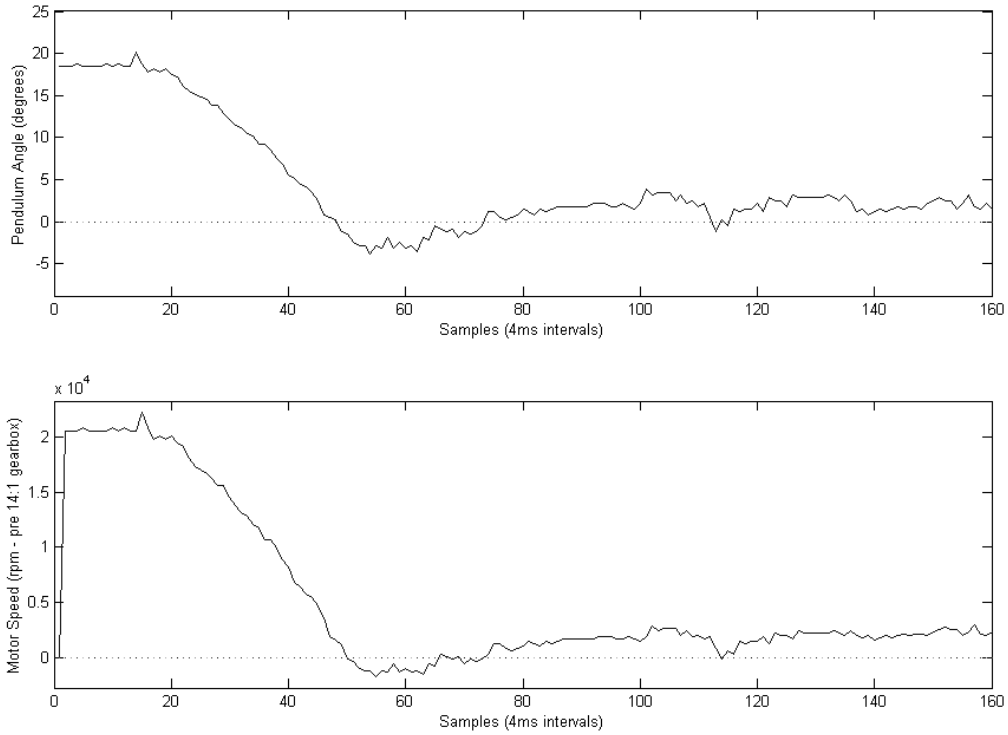


Figure 15 - *Plot of pendulum angle and actuator commands. Inverted pendulum under multi-processor PID control, where Kp = 350, Ki = 3 and Kd = 20.*

4.2 Summary

This section has described a simple distributed architecture for control of the test bed. This architecture uses three C167 nodes to manage different parts of the control loop.

The shared-clock scheduler forms a reliable time base to keep the nodes synchronized, but inevitably in does introduce extra latency after every tick, due to the transmission and receipt of tick messages. This nature of this extra delay can be unpredictable so any arbitrary compensation would still be susceptible to some jitter.

Logged data from the test bed shows that despite the extra complications of the distributed architecture, performance can still be maintained to an adequate standard under normal conditions. This data and the supporting video footage is to be used a target level of performance for the remainder of the project.

5. Conclusions

This paper has described the development of a test bed as means to researching fault-tolerance in safety-critical control applications such as drive-by-wire. The test bed, based around a simple inverted pendulum incorporates a redundant set of sensors and actuators for forthcoming experiments.

The inverted pendulum problem is well known and in its most simple form can approximated to a linear single-input, single-output system. Two different control architectures, for the test bed have been described. Both implementations used a PID algorithm, but one used a single-processor and the other a network of three processors. A time-triggered co-operative scheduling approach has been used to regulate the timing of control tasks. In the distributed controller, a shared-clock architecture was used to keep the different processors synchronised.

The future aims of this project are to subject different network and software architectures to fault-insertion tests, with a view to adding designers of future drive-by-wire systems. Records of performance have been made using the implementations described in this paper by logging data on a desktop PC and by videoing the test-rig in operation.

6. Acknowledgements

The authors would like to thank Tony Forryan (University of Leicester) for his invaluable assistance in the construction of the test bed described in this paper.

This project is funded by the EPSRC and TRW Conekt.

7. References

Beach M., (1997), “Fuel Injection for a 12 Cylinder Formula 1 Engine”, Microcontrollers ApNote – AP1635, Infineon.

Callender A., and Stevenson A. B., (1939), “Automatic Control of Variable Physical Characteristics.”, U.S. patent 2,175,985. Filed February 17, 1936 in the United States. Filed February 13, 1935 in Great Britain. Issued October 10, 1939 in the United States

Dorf R. C., and Bishop R. H., (2001), “Modern Control Systems – 9th Edition”, Prentice Hall.

Franklin G. F., Powell J. D., and Workman M. L., (1995), “Digital control of dynamic systems”, Menlo Park, California, Harlow, Addison Wesley.

Franklin G. F., Powell J. D., and Emami-Naeini A., (2002), "Feedback Control Of Dynamic Systems – 4th Edition", Prentice Hall.

Jantzen J., (1998), "Analysis of a Pendulum Problem", Technical Report 98-E 863, Department of Automation, Technical University of Denmark.

Kopetz H., (1998), "A Comparison of CAN and TTP", Vienna University of Technology.

Krahe C., (1999), "Lightning Strikes and Airbus Fly-by-wire Aircraft", Air & Space Europe Volume 1 No 2.

Ladkin P. B., (1995), "Analysis of a Technical Description of the Airbus A320 Braking System", CRIN-CNRS & INRIA.

Lian F., Moyne J., and Tilbury D., (2002), "Network Design Consideration for Distributed Control Systems", IEEE Transactions on Control Systems Technology, Vol. 10, No. 2, pp 297-306.

Lim C. K., (2001), "Control of a Double Inverted Pendulum System", BEng Thesis, University of Leicester.

Messner B., and Tilbury D., (1998), "Control Tutorials for Matlab and Simulink", http://wolfman.eos.uoguelph.ca/~jzelek/matlab/ctms/index.htm, accessed 21/05/04.

NASA Dryden Flight Research Centre, (1999), "F-8 Digital Fly-by-wire Aircraft – Project Summary", http://www.dfrc.nasa.gov/Newsroom/FactSheets/FS-024-DFRC.html, accessed 16/05/03.

Ogata K., (1995), "Discrete-Time Control Systems", Prentice Hall, London

Park H. S., Kim Y. H., Dong-Sung K., and Kwon W. H., (2002), "A Scheduling Method for Nework-Based Control Systems", IEEE Transactions on Control Systems Technology, Vol. 10, No. 3, pp 318-329.

Pratt R. W., (2000), "Flight Control Systems – Practical issues in design and implementation", IEE.

Pont M. J., (2001), "Patterns for Time-Triggered Embedded Systems", Addison-Wesley.

Rushby J., (2003), "A Comparison of Bus Architectures for Safety-Critical Embedded Systems", NASA Contractor Report CR-2003-212161.

SAE, (1993), "Class C Application Requirement Considerations", SAE Recommended Practice J2056/1, SAE.

Stanton N. A., and Marsden P., (1996), "From Fly-by-wire to Drive-by-wire: Safety impocations of Automation In Vehicles", Safety Science, Vol. 24, No. 1, Elsevier Science Ltd.

Stimac A. K., (1999), "Standup and Stabilisation of the Inverted Pendulum", BSc Thesis, Massachusetts Institute of Technology.

Wong W. M., (2002), "Construction, Modelling and Control of a Double Inverted Pendulum", BEng Thesis, University of Leicester.

The application of dynamic voltage scaling in embedded systems employing a TTCS software architecture

Teera Phatrapornnant and Michael J. Pont

Embedded Systems Laboratory, University of Leicester,
University Road, Leicester LE1 7RH, UK
http://www.le.ac.uk/eg/embedded/

Abstract

Dynamic voltage scaling (DVS) is a well-known technique which seeks to reduce power consumption in embedded systems by altering the operating voltage (and thereby the performance) of the CPU core. This paper discusses and compares a range of techniques for implementing DVS in applications using a time-triggered, co-operatively scheduled (TTCS) software architecture. Techniques for determining whether the use of DVS is likely to lead to an overall reduction in CPU power consumption for a given TTCS application are also discussed.

1. Introduction

Power consumption in high-performance processors is of growing concern for the developers of many embedded systems. The power consumption in CMOS circuits is proportional to the square supply voltage: see Equation 1 (Burd and Brodersen, 1995), where V_{DD} is the supply voltage, f is the clock frequency and C_L is the load capacitance.

$$P = V_{DD}^2 . f . C_L \quad (1)$$

As a result, lowering the supply voltage of the processor core is a particularly effective method of reducing power consumption (Burd and Brodersen, 1995). However, decreasing the supply voltage increases the delay in CMOS circuits (Sakurai and Newton, 1990) and the circuit delay is inversely proportional to clock frequency: see Equation 2 (Wolf, 2002), where t_d is circuit delay and L_d is the depth of critical path.

$$f = \frac{1}{L_d . t_d} \quad (2)$$

Dynamic Voltage Scaling (DVS), has been developed as a means of balancing performance and power consumption on scheduled systems (Weiser *et al.*, 1994.; Govil *et al.*, 1995; Ishihara and Yasuura, 1998; Pering and Brodersen, 1998; Pering *et al.*, 1998; Lee and Sakurai, 2000; Swaminathan and Chakrabarty, 2000; Lorch and Smith, 2001; Pillai and Shin, 2001; Pouwelse *et al.*, 2001; Simunic *et al.*, 2001; Zhang and Chanson, 2003). DVS achieves a reduction in power consumption by lowering the supply voltage of the CPU when the full performance is not required.

DVS algorithms can be divided into two main groups: interval-based and profile-based (Zhang and Chanson, 2004). Interval-based DVS (Weiser *et al.*, 1994; Pering and Brodersen, 1998; Lorch and Smith, 2001) predicts the processor speed required to execute the upcoming CPU load by considering the past load behaviour. Profile-based DVS (Shin and Choi, 1999; Pillai and Shin, 2001; Pouwelse *et al.*, 2001; Shin and Kim, 2001) uses computational resource requirements, such as execution time and deadline, to determine the optimal running speed of each task. Note also that, whereas most previous studies have explored the energy minimisation problem for periodic tasks, Qadi has described a DVS scheduling algorithm that supports canonical sporadic tasks (Qadi *et al.*, 2003).

Whatever form of DVS is used, there will be a related overhead (Xie *et al.*, 2003). This overhead arises from both the frequency/voltage switching itself, and calculations required to determine the required frequency/voltage settings (Qadi *et al.*, 2003). The switching overhead largely depends on the processor and hardware set up (for example, on the speed at which the oscillator PLL locks after a change in settings, and how rapidly the voltage settles after a change). The calculation overhead clearly depends on the complexity of the scheduling algorithm.

In this paper, we consider the application of DVS systems using a time-triggered co-operative scheduling (TTCS) architecture (Pont, 2001). We explore the reductions in power consumption that can be achieved when using DVS in this context. We also describe how to predict whether the application of DVS is likely to result in a net reduction in power consumption when applied to a particular TTCS system.

The rest of this paper is organised as follows. Section 2 presents TTCS scheduling algorithm. Section 3 describes hardware platform and presents our DVS scheduling algorithms. In Section 4, we assess and compare DVS scheduling algorithms. In Section 5, we describe how to determine the break-even point of DVS algorithms. We conclude with a discussion of results in Section 6.

2. The TTCS scheduling algorithm

To study the potential impact of DVS on the power consumption of embedded systems implemented using a time-triggered, co-operatively scheduled (TTCS) software architecture, we used a real-time co-operative scheduler from (Pont, 2001). This co-operative scheduler is based on fixed-priority algorithm and has a small overhead. It is available in various versions. For the purposes of this study, we used code that was written for the ARM architecture.

In this paper, we focus on implementation of a DVS algorithm in the scheduler "dispatcher" function. The dispatcher is invoked after a clock tick occurs. It will detect all tasks from its list to find which tasks are due to run. The dispatcher functions then releases these tasks in order. The detail of the dispatcher function is listed, in pseudo code, in Figure 1.

```
Task DISPATCH_TASKS:

    while clock tick count > 0
        for  loop all tasks in tasks array
                if  delay parameter is decrease until zero then
                        release the task ;
                        if  it is periodic task then
                            reload period parameter into array ;
                        else
                            delete task from array ;
                        end if
                end if
            end for
            decrease clock tick count ;
    end while
    idle until the next clock tick interrupt ;

end DISPATCH_TASKS
```

Figure 1 Task dispatcher algorithm of TTCS

The key to applying DVS in a TTCS application is the presence of slack time. Under DVS, tasks - which normally run at a fixed CPU speed - will be stretched to fill the available slack time (see Figure 2). Therefore, the speed-setting policy is determined by the available slack time for each task.

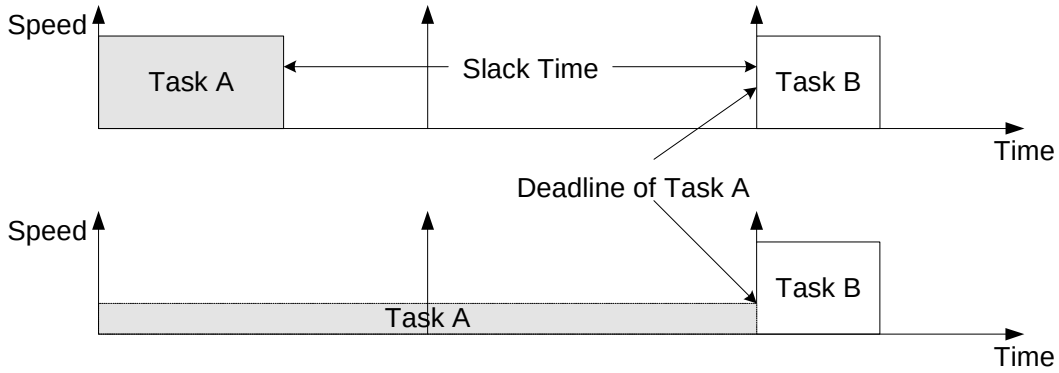


Figure 2 Example illustrating the possibility of task stretching.

In most TTCS systems, the task deadline is the end of the tick interval in which the task is released (e.g. see Figure 2). In theory, therefore, the available execution time is simply the interval between the release time and the time of the next tick. However, the overhead of the scheduling algorithm

itself must also be considered in practical systems (Figure 3). For example, in the present study schedulers with a 1 ms "tick" interval were employed and – on the hardware platform used – the scheduler overhead exceeded 15% of the tick interval.

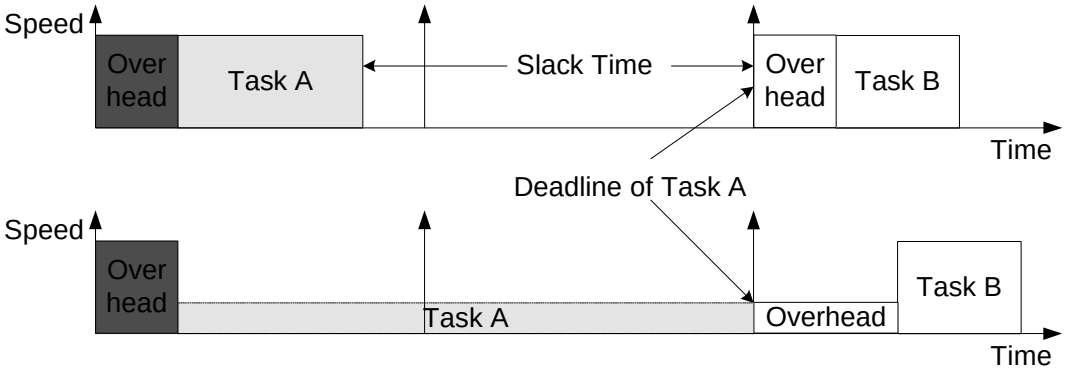


Figure 3 Illustrating a more realistic DVS implementation.

The overhead is divided into its two key components: speed finding (including task scheduling) and frequency/voltage scaling. Values for each of these components can be determined by measurement.

3. Implementing DVS in TTCS systems

There are many possible ways of implementing DVS and a number of possible designs were explored. Four of these designs are described in this section, and compared experimentally in Section 4.

The four designs were named as follows (for ease of reference):

- The Compute-Direct (CD) algorithm
- The Lookup Table (LT) algorithm
- The Circular Array (CA) algorithm, and,
- The Circular Skip (CS) algorithm

Each of these algorithms is described in turning this section, after an overview of the hardware platform used in this study.

3.1 The hardware platform

The studies reported in this paper used a Philips LPC2106 processor. The LPC2106 is 32-bit microcontroller with an ARM7-core which can run – under control of an on-chip PLL – at frequencies from 10 MHz to 60 MHz (Philips, 2003).

For the purposes of these studies, the LPC2106 was mounted on an Ashling LPC2000 evaluation board (Ashling, 2003). The board provided separate 1.8V (CPU core) and 3.3V (I/O) supply jumpers, making it easy to implement a variable-voltage CPU supply. An external digital to analogue converter (DAC) was controlled by the processor via an SPI interface. This DAC was used to vary the reference voltage on a DC-DC converter, thereby producing the required CPU voltage.

In these studies, the CPU speed was divided into 6 steps. An external crystal oscillator of 10 MHz was linked to a PLL multiplier (1x to 6x) to generate 10, 20, 30, 40, 50 and 60 MHz system clocks. The supply voltage also had 6 levels corresponding to the 6 speed steps. The required voltage level

at each speed was determined by measuring the lowest voltage at which processor still worked properly and then adding a 10% safety margin (Figure 4).

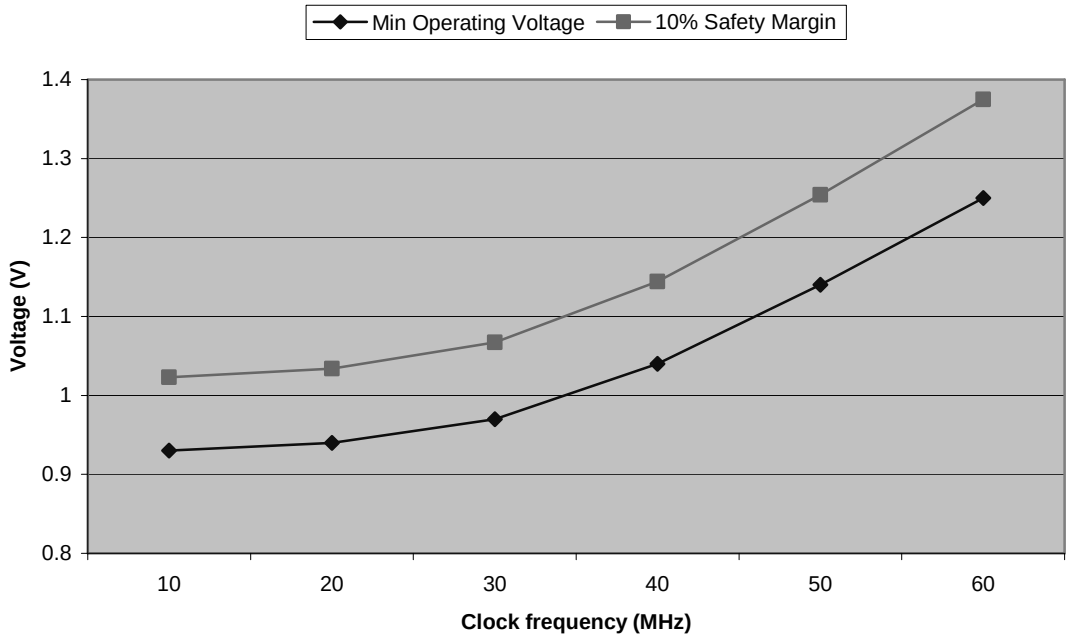


Figure 4 Setting the CPU supply voltage (LPC 2106).

3.2 Compute-Direct (CD) algorithm

The first DVS algorithm considered in this study is the most straightforward, and will be referred to here as “Compute Direct” (CD).

The CD algorithm employed to calculate the DVS settings is run from the Dispatcher function (see (Pont, 2001)). At every clock tick, the characteristics of the task due for release (if any) are examined, to find the deadline. The CD algorithm will then aim to determine if the task can meet its deadline if executed at the lowest speed setting: if it can, the speed is set to this value and the task is released. If the deadline cannot be met at this speed, the next increment is tried. This process is repeated, as necessary, until the maximum speed setting is reached. Note that it is assumed that all tasks will meet their deadlines at the maximum speed setting.

A pseudo-code representation of this algorithm is shown in Figure 5.

```
Task DISPATCH_TASKS:

    while clock tick count > zero
      number of task = 0 ;
      next task released tick = 0 ;
      for  loop all tasks in tasks array
            if  delay parameter is decrease until zero then
                store task index run in this tick in current tick array ;
                number of task ++ ;
             else
                if next task released tick > current task delay or == 0
                      next task released tick = current task delay ;
                end if
             end if
       end for

       switch  number of task
             case  0 :  assign cpu speed to the lowest ;
                        scaling frequency and voltage ;
                       break
             case  1 :  if  deadline parameter is not defined
                              deadline = next task released tick ;
                        else
                              deadline = deadline parameter ;
                        end if
                        find current speed ;
                        cpu speed = lowest speed ;
                        while  !((calculated task duration < deadline  and  cpu speed < max_speed)
                                    cpu speed ++ ;
                         end while
                         break
             case  another :  assign cpu speed to the highest;
                        break
       end switch

        for  loop all tasks in tasks array
             if  delay parameter is decrease until zero then
                 scale frequency and voltage ;
                 release the task ;
                 if it is a periodic task then
                       reload period parameter into array ;
                 else
                       delete task from array ;
                 end if
              end if
         end for
            decrease clock tick count ;
    end while
    idle until the next clock tick interrupt ;

end DISPATCH_TASKS
```

Figure 5 Compute-Direct algorithm

3.3 Lookup Table (LT) algorithm

The CD algorithm is conceptually simple but computationally expensive. To reduce the CPU load, a lookup table can be employed.

Using such a table, the task scheduling and frequency / voltage scaling execution time will be measured, calculated and stored. Note that the required calculations depend not only on the task to be executed, but also the context of this execution (for example, whether Task A is immediately followed by the execution of Task B).

To simplify the discussions here and later in the paper, the task durations will be represented in “utilisation units”. We consider each task slot as 1U (Figure 6).

If a task employs the whole time slot at full speed (60 MHz in our system), it takes 100% of the available utilisation: such a task cannot run at lower speed because its utilisation will exceed 1U. Similarly, if a task takes 100% of the utilisation at 30 MHz, it has a load of 0.5U at 60 MHz.

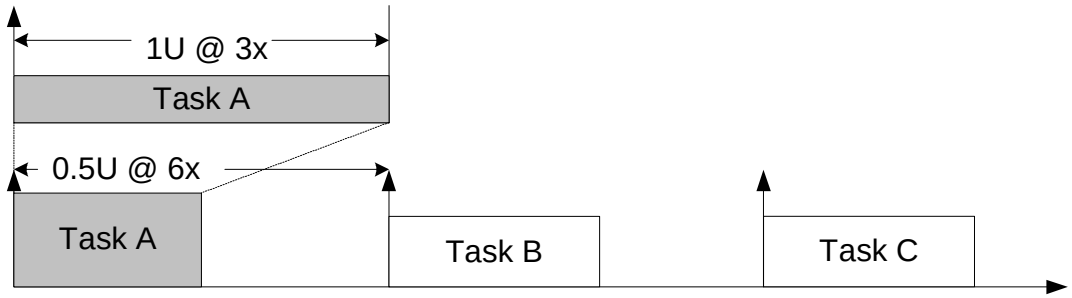


Figure 6 Considering intra-task utilization

Figure 7 shows the maximum utilisation at various speeds, based on the maximum speed at 60 MHz. We can find available task slot at each speed by deducting the scheduler overhead from these values. The available task slot values will then be stored in the lookup table.

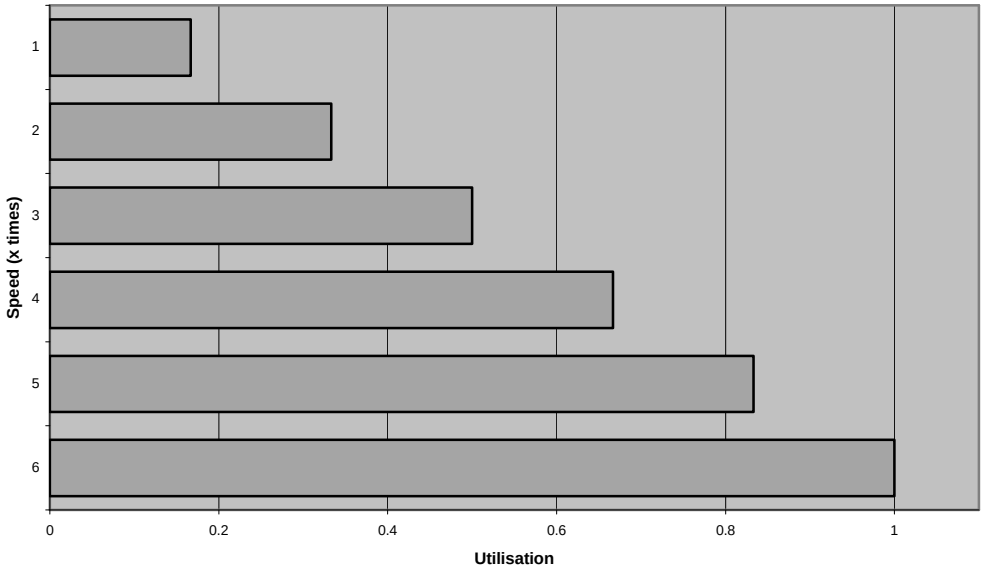


Figure 7 The boundary of utilisation based on 60 MHz

Task DISPATCH_TASKS:

```
while clock tick count > zero
  number of task = 0 ;
  next task released tick = 0 ;
  for  loop all tasks in tasks array
        if  delay parameter is decrease until zero then
            store task index run in this tick in current tick array ;
            number of task ++ ;
        else
            if next task released tick > current task delay or == 0
                next task released tick = current task delay ;
            end if
        end if
  end for

  switch  number of task
        case  0 :  assign cpu speed to the lowest ;
                   scaling frequency and voltage ;
                   break
        case  1 :  if  deadline parameter is not defined
                       deadline = next task released tick ;
                   else
                       deadline = deadline parameter ;
                   end if
                   task utilisation = task duration parameter / deadline ;
                   find current speed ;
                   if task utilisation less than task slot[current speed][1x]
                       assign cpu speed to 1x ;
                       break ;
                       end if
                   if task utilisation is between task slot [current speed][1x] and task slot [current speed][2x]
                       assign cpu speed to 2x ;
                       break ;
                       end if
                   if task utilisation is between task slot [current speed][2x] and task slot [current speed][3x]
                       assign cpu speed to 3x ;
                       break ;
                       end if
                   if task utilisation is between task slot [current speed][3x] and task slot [current speed][4x]
                       assign cpu speed to 4x ;
                       break ;
                       end if
                   if task utilisation is between task slot [current speed][4x] and task slot [current speed][5x]
                       assign cpu speed to 5x ;
                       break ;
                       end if
                   if task utilisation greater than task slot [current speed][5x]
                                assign cpu speed to 6x ;
                       break ;
                       end if
                   break
        case  another :  assign cpu speed to the highest;
                   break
  end switch

   for  loop all tasks in tasks array
        if  delay parameter is decrease until zero then
            scale frequency and voltage ;
            release the task ;
            if it is a periodic task then
                reload period parameter into array ;
            else
                delete task from array ;
            end if
        end if
    end for
        decrease clock tick count ;
end while
idle until the next clock tick interrupt ;

end DISPATCH_TASKS
```

Figure 8 Lookup Table algorithm

3.4 Circular Array (CA) algorithm

Both the CD and LT algorithms execute every time a task is dispatched. In the Circular Array algorithm, the scheduling overhead is reduced by moving the core of the speed-finding procedure to a system initialisation function, which is run only once.

For example, suppose there are three periodic tasks, A(0,5), B(2, 10), C(5,15). Here the tasks are represented as *name*(*delay, period*): task A, B and C start at 0, 2 and 10 ms (respectively) and repeat every 5, 10 and 15 ms (respectively). The cycle period is determined by finding the greatest common factor from the task periods. In the example, the cycle value is 30.

If these periodic tasks are assigned a running speed, they will be run at the same speed every cycle. Consequently, the speed of the task in each tick can be determined once and applied in every cycle. To implement this algorithm, a circular array, which has a size equal to the number of task slots in one cycle, can be used to store the required CPU speed. When the system is initialised, the speed-finding procedure will be run for a full cycle (without dispatching tasks) in order to calculate and store the required speed values. A scheme to synchronise the circular array pointer with task slot will then be run before the scheduler starts (see Figure 9).

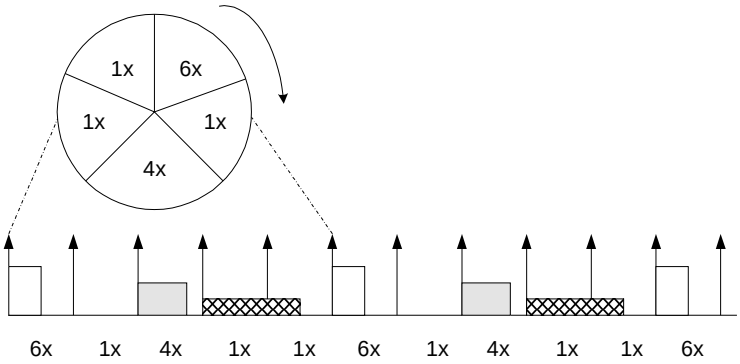


Figure 9 A schematic representation of the Circular Array algorithm.

```
Task DISPATCH_TASKS:

    while clock tick count > zero
       if circular array pointer > cycle value then
            reset circular array pointer ;
        for  loop all tasks in tasks array
               if  delay parameter is decrease until zero then
                   load speed parameter from circular array ;
                   scale frequency and voltage ;
                   release the task ;
                   if it is a periodic task then
                         reload period parameter into array ;
                   else
                         delete task from array ;
                   end if
                end if
          end for
              decrease clock tick count ;
    end while
    idle until the next clock tick interrupt ;

end DISPATCH_TASKS
```

Figure 10 Circular Array algorithm

3.5 Circular Skip (CS) algorithm

To minimise power consumption, we would like to run all tasks at the minimum speed and never change the speed setting. This is rarely practical, but – with appropriate task scheduling – we can often reduce the number of speed changes required, thereby reducing the power consumption.

This idea is exploited in what we call here the "Circular Skip" (CS) algorithm. CS is applied after speeds have been calculated and stored in the circular array (using the CA algorithm). Using CS we examine the array and look for instances where no speed changes are required between the execution of two tasks which are scheduled to run consecutively: when such a situation is found, the superfluous speed-change step is removed. This change itself reduces power consumption.

In addition, when the speed change is removed, this frees up CPU time and – as a consequence – it may be possible to run some tasks at a lower speed than was possible using the CA algorithm. In this way CS adds to the reduction in power consumption.

```
Task DISPATCH_TASKS:
    static previous speed = 0 ;
    while clock tick count > zero
       if circular array pointer > cycle value then
            reset circular array pointer ;
        for  loop all tasks in tasks array
                if  delay parameter is decrease until zero then
                    load speed parameter from circular array ;
                    if previous speed != speed parameter then
                          scale frequency and voltage ;
                          save the current speed to previous speed ;
                    end if
                    release the task ;
                    if it is a periodic task then
                          reload period parameter into array ;
                    else
                          delete task from array ;
                    end if
                end if
          end for
               decrease clock tick count ;
    end while
    idle until the next clock tick interrupt ;

end DISPATCH_TASKS
```

Figure 11 Circular Skip algorithm

4. Assessing and comparing the DVS algorithms

The experiments carried out to assess and compare the DVS algorithms described in Section 3 are described here.

4.1 The task set

For a preliminary study, we set up 1 ms clock tick for all schedulers. We created 5 dummy tasks that had a total utilisation from 0.1U to 0.9U. All tasks had a 5 ms period.

Each dummy task was implemented using a simple loop, adapted as required to give the required duration. For example:

```
Dummy Task:
      for loop
         {
         a = a + 1;
         }
```

Note that, in time-triggered co-operative scheduling system, all task durations should be less than the scheduler tick interval. In this case, the duration of all tasks was less than 1 ms.

4.2 Overhead analysis

Figure 12 illustrates the scheduling overhead for each of the DVS algorithms. To draw this figure, the voltage and frequency scaling time of all speed-setting and DVS algorithms were measured at 60MHz and then the scheduling overhead at all speeds was predicted. The resulting figures permit the calculation of the available execution time.

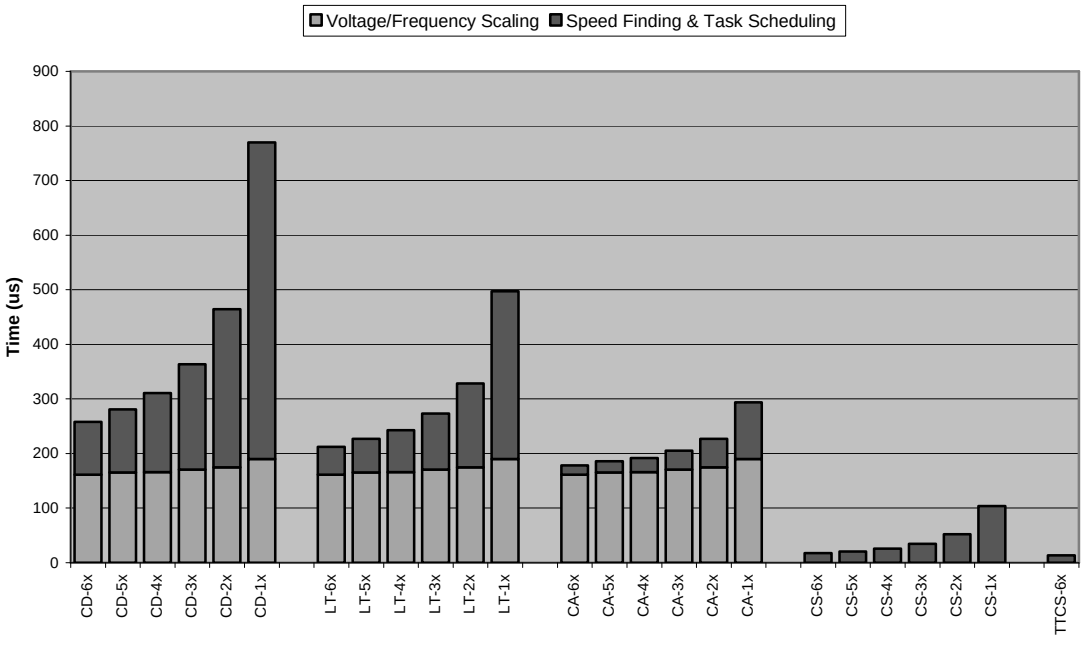


Figure 12 Scheduling overhead of DVS algorithms estimation

From Figure 12, the CD overhead is around 257.8 μs when run with the task set at 60 MHz. This figure increases when lower speed are applied: it is 769.8 μs at 10 MHz. It is obvious that the range between minimum and maximum values is also high: this is due to the speed-finding calculation, which has a duration dependent on the number of calculations required.

The available CPU time for tasks can be increased by reducing the time taken for speed-setting calculations, as the results from the LT and the CA algorithms show. In the case of the CS algorithm, if voltage and frequency scaling can be avoided, the overheads are close to those of the original TTCS algorithm.

From these scheduling overhead values, the available task utilisations of the various DVS algorithms can be calculated (Figure 13 - Figure 16).

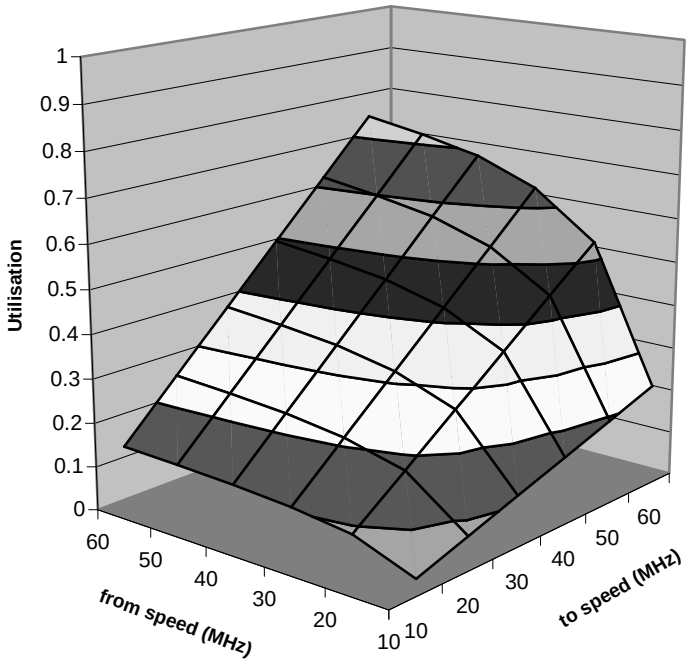


Figure 13 Maximum available utilisation of task – CD algorithm

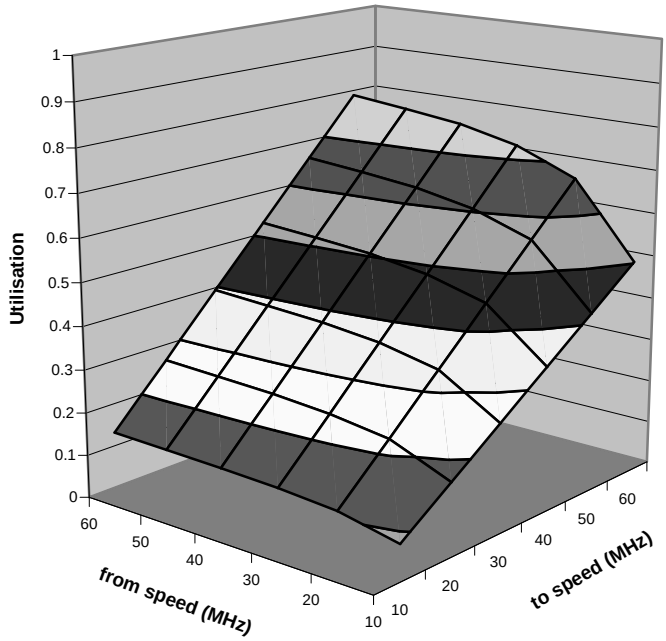


Figure 14 Maximum available utilisation of task – LT algorithm

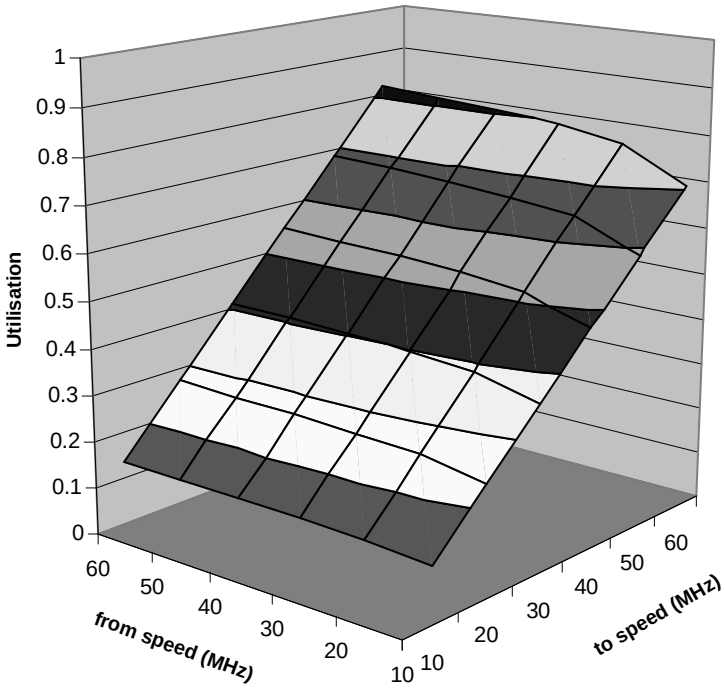


Figure 15 Maximum available utilisation of task – CA algorithm

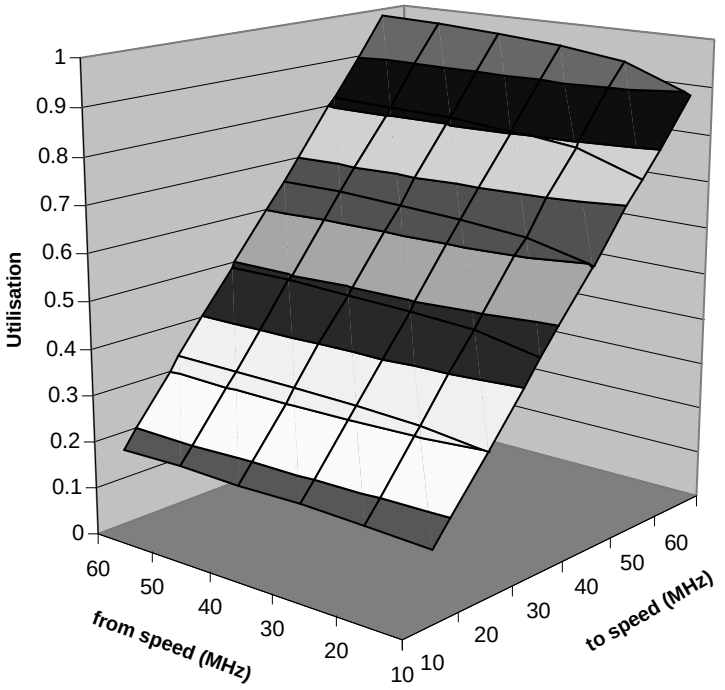


Figure 16 Maximum available utilisation of task – CS algorithm

4.3 More general performance comparisons

The results presented in Section 4.2 describe the overhead of the various algorithms. We provide a more direct comparison of the power consumption resulting from these various techniques in this section.

To perform this empirical comparison, task sets (with task durations from 0.1 U to 0.9 U), were set up and executed using the TTCS, CD, LT, CA and CS algorithms. The average power consumption of CPU core in each case was measured and is shown in Figure 17.

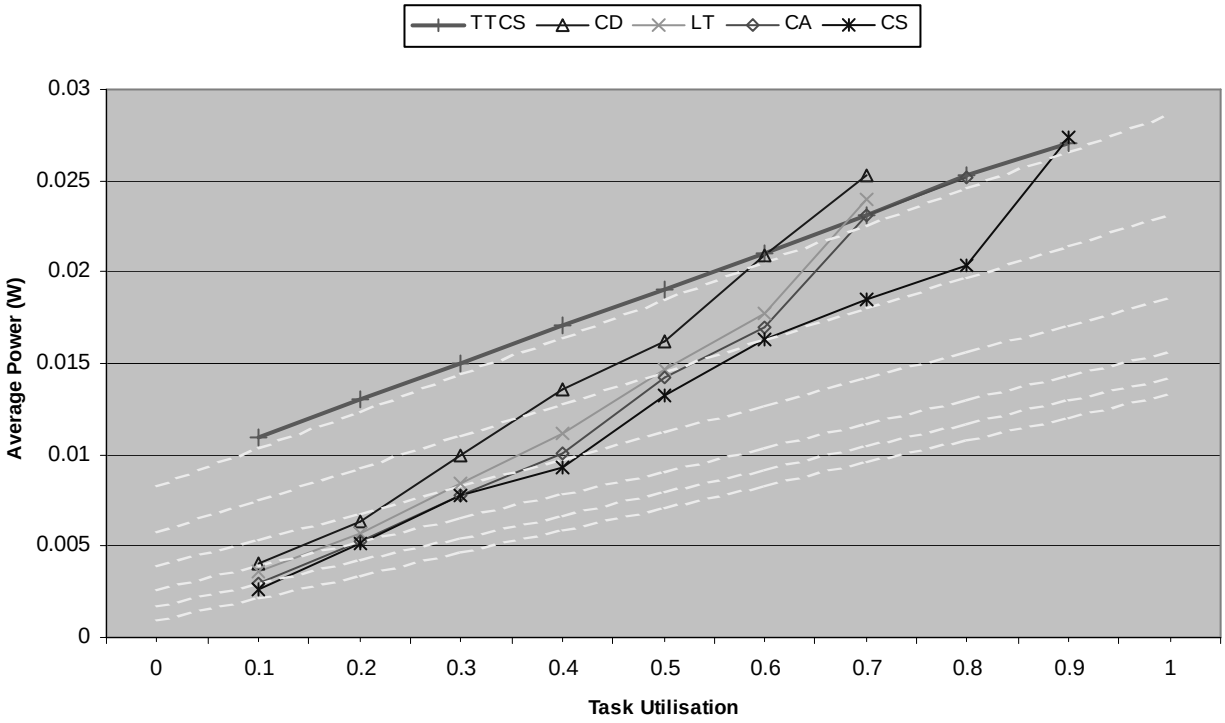


Figure 17 Power consumption of CPU core on different scheduling algorithms

Because it runs at a fixed 60 MHz frequency (when executing tasks) and in "idle" mode at other times, the TTCS power consumption shown in this figure is almost linearly related to the workload. Overall, the power consumption of the TTCS scheduling overhead is low: approximately 8.8 mW at 60 MHz.

At low levels of task utilisation all of the DVS algorithms succeed in reducing the system power consumption, when compared to the original TTCS algorithm.

CD is the most power-hungry of the DVS algorithms considered here, while the LT, CA and CS are the second, the third and the fourth, respectively. At every workload, the CD consumes more power than the rest of variable-speed approaches. Its power is close to the TTCS at 0.6 U and greater than TTCS at 0.7 U.

CS is the most power-efficient of these DVS algorithms. It has the lowest power consumption at all levels of utilisation, except at 0.3 U. In this case, the CS slightly takes power more than the CA because the task set that was run had large gap between their durations (100, 300, 600, 300 and 100 μs). In this particular configuration, the CS could not optimise the speed. With this exception, CS saves power, until we reach a utilisation level of 0.9 U: here, CS, which runs all tasks at full speed, consumes more power than the TTCS (because of the algorithm overhead).

5. Determining the break-even point

DVS scheduling is more complicated than fixed-voltage scheduling. As the additional instructions can increase power consumption, it is important to know – for a given system – whether use of DVS is likely to result in a net reduction in CPU power consumption.

Figure 18 shows the power consumption graphs for fixed-voltage and DVS scheduling systems at different levels of task utilisation. The fixed-voltage power line lies just above pure-load line. The dash-dot line represents the utilisation boundary.

Note that in this figure the task has duration 0.1U, was set to run at a speed of 1x. The remaining tasks (duration, 0.2, 0.3, 0.4, 0.5 and greater than 0.6), they were set to run at 2x, 3x, 4x, 5x and 6x, respectively.

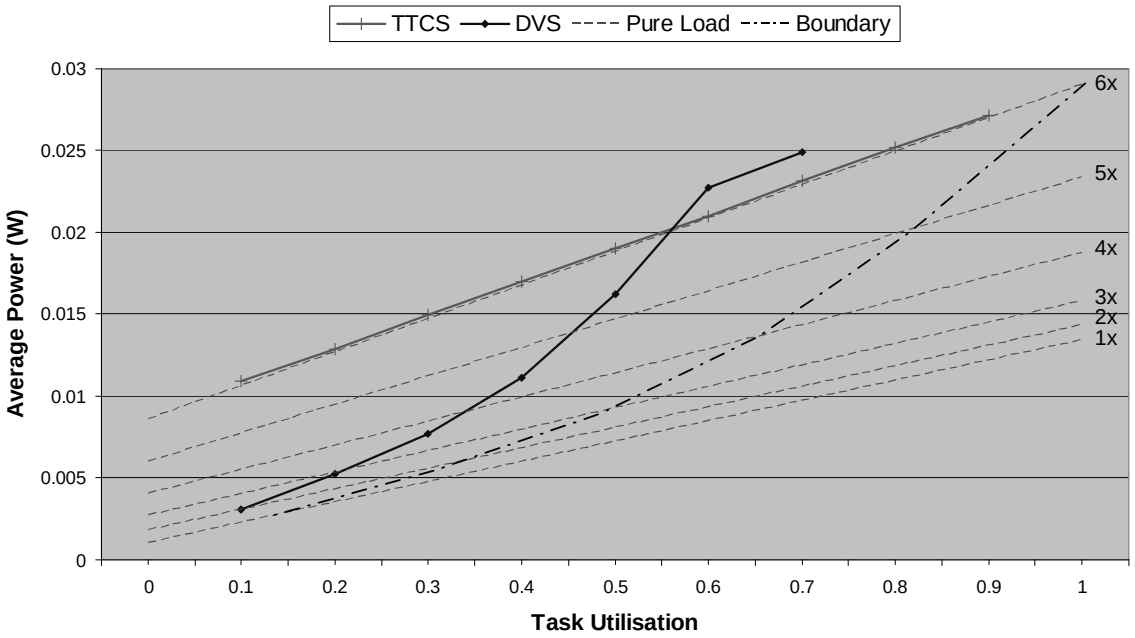


Figure 18 Illustrating the break-even point of DVS scheduling system of 1 task – 1ms period.

Figure 18 shows that the DVS system that runs at a speed below 6x consumes less power than an equivalent fixed-voltage scheduling system which always runs at 6x.

From this result, we can conclude that applications that have tasks that can all run at speeds of 5x (or below) will consume less CPU power than the equivalent fixed-voltage TTCS implementation. More precisely, we can specify the break-even points of the various DVS scheduling algorithms CD, LT, CA and CS to be 0.599 U, 0.644 U, 0.678 U and 0.816 U, respectively (see Figure 19).

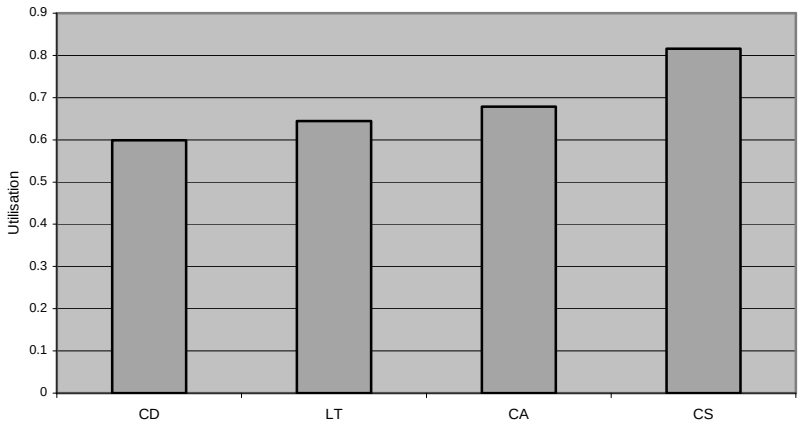


Figure 19 The break-even points for the various DVS algorithms.

6. Conclusion

Based on the studies presented in this paper, we can conclude that the application of DVS in TTCS embedded systems has the potential to significantly reduce the CPU power consumption.

Further studies are required to confirm these findings. We are currently conducting such studies.

7. References

Ashling Microsystems (2003), *"LPC2000 Evaluation and Development Kits datasheet"*, http://www.ashling.com/pdf_datasheets/DS266-V7U-EvKit2000.pdf.

Burd, T. D. and Brodersen, R. W. (1995), *"Energy Efficient CMOS Microprocessor Design"*, Proc. 28th Hawaii Int'l Conf. On System Sciences, Vol. 1, pp. 288-297.

Govil, K., Chan, E. and Wasserman, H. (1995), *"Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU"*, Proceeding of the first ACM International Conference on Mobile Computing and Networking (MOBICOM 95), pp. 13-25.

Ishihara, T. and Yasuura, H. (1998), *"Voltage Scheduling Problem for Dynamically Variable Voltage Processors"*, The International Symposium on Low Power Electronics and Design, pp. 197-202.

Lee, S. and Sakurai, T. (2000), *"Run-time Voltage Hopping for Low-Power Real-Time Systems"*, Proceedings of Design Automation Conference, pp. 806-809.

Lorch, J. R. and Smith, A. J. (2001), *"Improving Dynamic Voltage Scaling Algorithms with PACE"*, Proceedings of SIGMETRICS, pp. 50-61.

Pering, T. and Brodersen, R. (1998), *"Energy Efficient Voltage Scheduling for Real-Time Operating Systems"*, Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98.

Pering, T., Burd, T. and Brodersen, R. (1998), *"The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms"*, Proceedings of the International Symposium on Low Power Electronics and Design, pp. 76-81.

Philips Semiconductors (2003), *"LPC2104/2105/2106; Single-chip 32-bit microcontrollers"*, http://www.semiconductors.philips.com/acrobat/datasheets/LPC2104_2105_2106-04.pdf.

Pillai, P. and Shin, K. G. (2001), *"Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems"*, ACM Symposium on Operating Systems Principles, pp. 89-102.

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

Pouwelse, J., Langendoen, K. and Sips, H. (2001), *"Energy Priority Scheduling for Variable Voltage Processors"*, International Symposium on Low Power Electronics and Design, pp. 28-33.

Qadi, A., Goddard, S. and Farritor, S. (2003), *"A Dynamic Voltage Scaling Algorithm for Sporadic Tasks"*, 24th IEEE International Real-Time Systems Symposium, Cancun, Mexico, pp. 52-62.

Sakurai, T. and Newton, A. R. (1990), *"Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas"*, IEEE J. Solid-State Circuits, Vol. 25, pp. 584-594.

Shin, D. and Kim, J. (2001), *"A profile-based energy-efficient intra-task voltage scheduling algorithm for real-time applications"*, Proceedings of the 2001 international symposium on Low power electronics and design, California, pp. 271-274.

Shin, Y. and Choi, K. (1999), *"Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems"*, Proceeding of Design Automation Conference, pp. 134-139.

Simunic, T., Benini, L., Acquaviva, A., Glynn, P. and Micheli, G. D. (2001), *"Dynamic Voltage Scaling for Portable Systems"*, The seventh annual international conference on Mobile computing and networking, pp. 251-259.

Swaminathan, V. and Chakrabarty, K. (2000), *"Real-Time Task Scheduling for Energy-Aware Embedded Systems"*, IEEE Real-Time Systems Symposium.

Weiser, M., Welch, B., Demers, A. and Shenker, S. (1994), *"Scheduling for Reduced CPU Energy"*, Proceeding of the First Symposium on Operating Systems Design and Implementation, pp. 13-23.

Weiser, M., Welch, B., Demers, A. and Shenker, S. (1994.), *"Scheduling for Reduced CPU Energy"*, Proceeding of the First Symposium on Operating Systems Design and Implementation.

Wolf, W. (2002), *"Modern VLSI Design"* Prentice Hall.

Xie, F., Martonosi, M. and Malik, S. (2003), *"Compile-time Dynamic Voltage Scaling Settings: Opportunities And Limits"*, Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation, pp. 49-62.

Zhang, F. and Chanson, S. T. (2003), *"Processor Voltage Scheduling for Real-Time Tasks With Non-Preemptible Sections"*, Proceedings of IEEE Real-Time Systems Symposium, Austin, Texas, pp. 235-245.

Zhang, F. and Chanson, S. T. (2004), *"Blocking-Aware Processor Voltage Scheduling for Real-Time Tasks"*, ACM Transactions on Embedded Computing Systems, pp. 307-335.

"Little HTTP Server on Embedded MicroBlaze Processor"

E. CABAL-YEPEZ (E.Cabal-Yepez@sussex.ac.uk)
P. GOUGH (m.p.gough@sussex.ac.uk)
T. CAROZZI (T.Carozzi@sussex.ac.uk)

SPACE SCIENCE CENTRE
UNIVERSITY OF SUSSEX

1 INTRODUCTION

The number of gates and the rates of data transmission managed in current FPGA devices have made possible the birth of new areas for Digital System research and development. Following the System On-a-Chip philosophy, more and more developers are incorporating the use of **Embedded Systems** in their design processes.

This paper shows a HTTP server built on an embedded system and describes peripherals, system configuration and tool functionality used during system design. Although through the design process specific design environment, hardware and software analysis tools and IP cores are used, the same analysis tools used here are also available on their corresponding presentation for development software from different device manufactures and the IP cores used are present on almost every embedded system design.

This project is intended to create a tool for giving support to data processing of a specific application data base. The goal is to reduce processing time and increase data transmission rate using an embedded system as an interface between users' applications and the data base main server. Besides giving support for creating and keeping a communication channel, the system will process data from the data server using specific application hardware in order to reduce processing time with the aim of avoiding overloading of users' machines and the data server.

In this document we show a system application on an embedded microprocessor in a Xilinx Virtex II FPGA on a MEMEC V2MB1000 Development Board. We use the P160 Communication Module Board for communication through Internet and debugging. This system application is the first stage of a larger design involving the

use of a TCP/IP stack for receiving and sending information, and hardware for data processing with the embedded microprocessor in charge of communication protocols. The work in this document just describes the implementation of an HTTP server in a XC2V1000 Xilinx device using the embedded soft core MicroBlaze processor, and Xilinx tools, Libraries and peripheral cores included in the Xilinx Embedded Development Kit software.

2 SYSTEM DEFINITION

Figure 1 shows the block diagram of the system with users accessing the website using a commercial web browser. The HTTP server on the V2MB1000 development board runs on the MicroBlaze processor soft core utilizing the network library LibXil Net including functions for handling the TCP/IP stack and the higher level programming application interface (Socket APIs) [1].

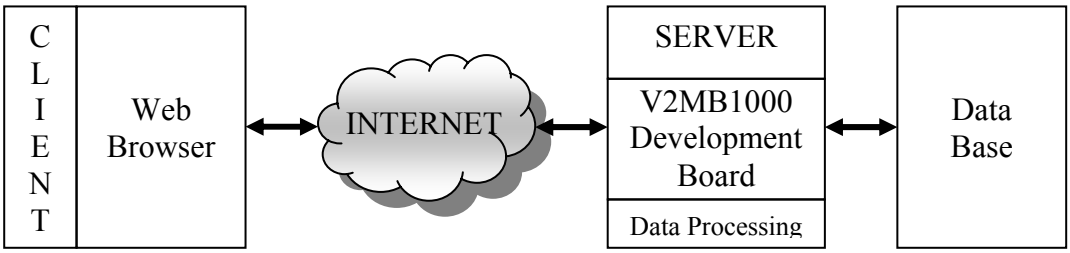


Figure 1. System Block Diagram

Figure 2 depicts the embedded system block diagram showing MicroBlaze processor core and peripheral cores as well as the interconnection among them.

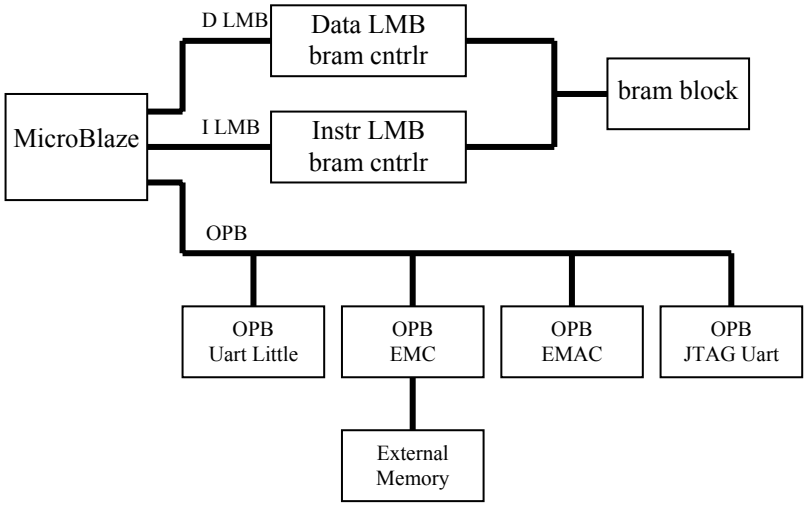


Figure 2. Embedded System Block Diagram

- A brief description for each block in figure 2 is given in APENDIX A.

3 SYSTEM CONFIGURATION

Figure 3 depicts a tools and hardware connection view between designer's machine and embedded system and between embedded system and the external SRAM on the P160 communications module board.

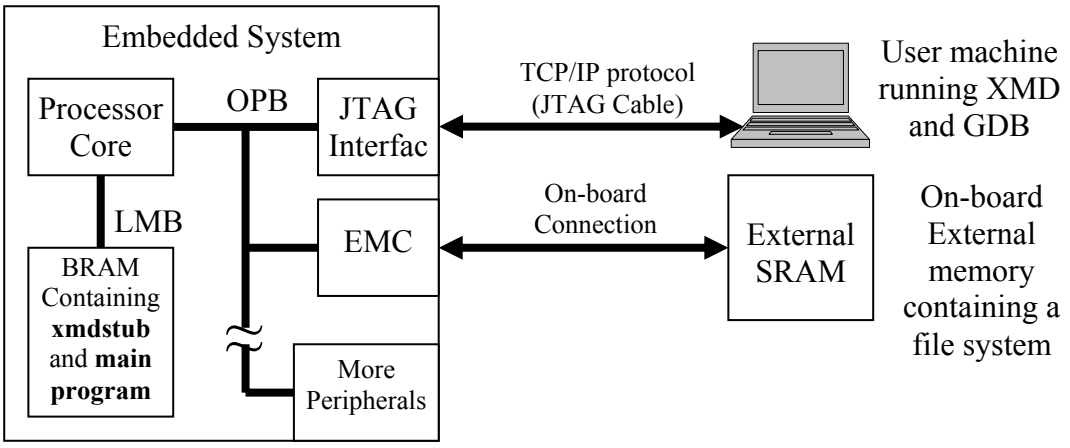


Figure 3. User design tools, External SRAM and Embedded System Connection

The design configuration showed in figure 3 allows the use of XMD (*Xilinx Microprocessor Debugger*) and GDB (*GNU Debugger*) tools. XMD is a tool that provides a unified GDB interface as well as a Tcl (*Tool Command Language*) interface for debugging programs and verifying systems using Microblaze microprocessors [1].

With a stub target, user programs can be downloaded from mb-gdb (*Microblaze GDB*) directly onto an On-chip Local Memory or an on-board external hardware and be executed with support of xmd stub running on the embedded system. User data also can be downloaded onto external hardware with support of xmd stub and Tcl commands.

3.1 ADDRESS MANAGEMENT

Figure 4 shows the address mapping restrictions for a MicroBlaze system using a stub target. The xmdstub is downloaded at position 0x0, if the user code start address is not

specified; it is set at position 0x400 by default in order to ensure that the compiled program starts after xmdstub.

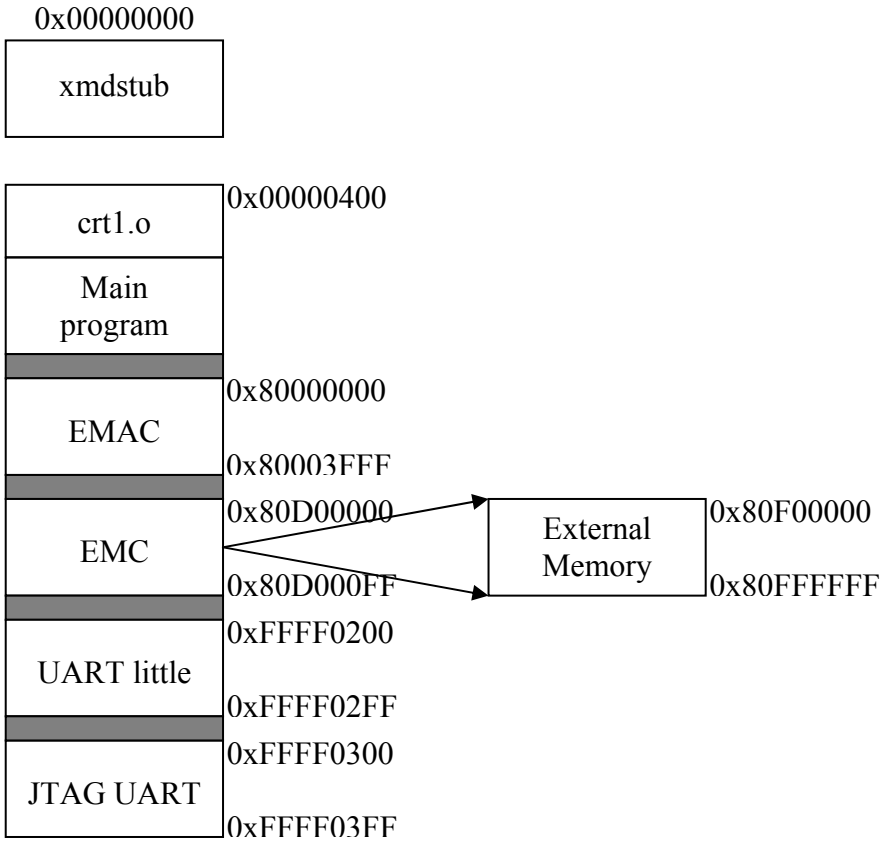


Figure 4. Memory Mapping

3.2 WEB SERVER CODING

3.2.1 LibXilNet Library

The Embedded Development Kit library, LibXilNet, allows a processor to connect to the Internet by including functions for handling the TCP/IP stack protocols and also providing a simple set of Socket Application Programming Interface functions enabling network programming [1].

The first step in the server configuration is the initialization of some parameter and devices. Functions from LibXilNet library are used for this purpose.

- **xilnet_mac_init** function initializes the MAC base address; this function must be used at the beginning.

- **xilnet_eth_init_hw_addr_tbl** function initializes hardware address table. This function must be called before using other functions of LibXilNet.
- **xilnet_eth_init_hw_addr** function initializes the Ethernet source address with a 48-bit, colon separated, hexadecimal Ethernet address string.
- **xilnet_ip_init** function initializes the ip address for the processor to the address represented in the argument received as a dotted decimal string.

The server is created using the basic TCP socket connection flow on the server's side. Figure 5 depicts this flow as well as LibXilNet functions used for socket connection.

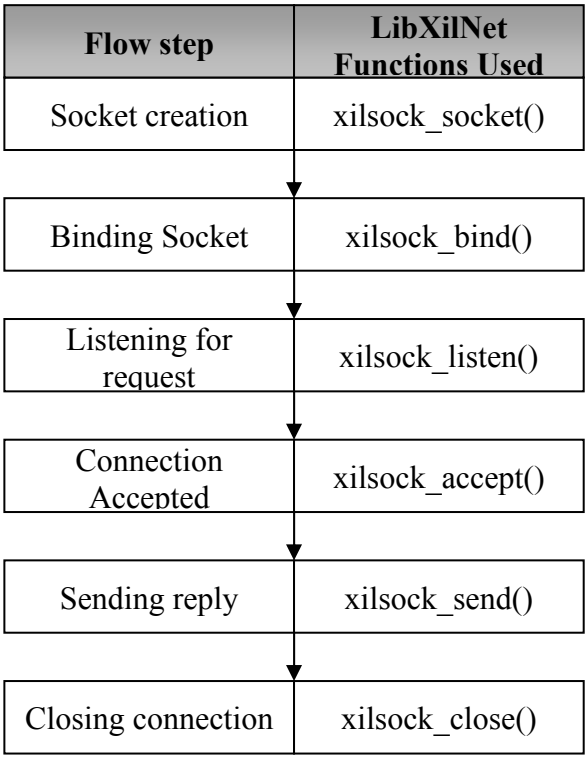


Figure 5. Basic TCP socket connection flow

3.2.1.1 Aspects to be taken into account when using XilNet Library.

- Since there are no timers used, every "*send*" over a TCP connection waits for an "*ack*" before performing the next "*send*" [1].
- TCP connection using the XilLibNet library, work as a finite state machine. The function *xilsock_accept* is in charge of setting the state in the *xilsock_status_flag* variable according to the communication stage.

The possible values for the *xilsock_status_flag* variable are

- *XILSOCK_NEW_CONN*. New connection incoming.
- *XILSOCK_SYNACK_RCVD*. System synchronization.
- *XILSOCK_TCP_DATA*. Data request received.
- *XILSOCK_TCP_ACK*. The client has received data correctly.
- *XILSOCK_EXISTING_CONN*. Existing connection.
- *XILSOCK_CLOSE_CONN*. Connection is closed.

- For *xilsock_accept* function, arguments *addr* and *addrlen* are in place to support the standard Socket accept signature. At the present, they are not used in the function [1].
- The maximum buffer length to be sent through *xilsock_send* function is 1514 bytes of which 14 are used for the link header, 20 are used for the TCP header and 20 more are used for the IP header, leaving 1460 bytes remaining for user data transmission.

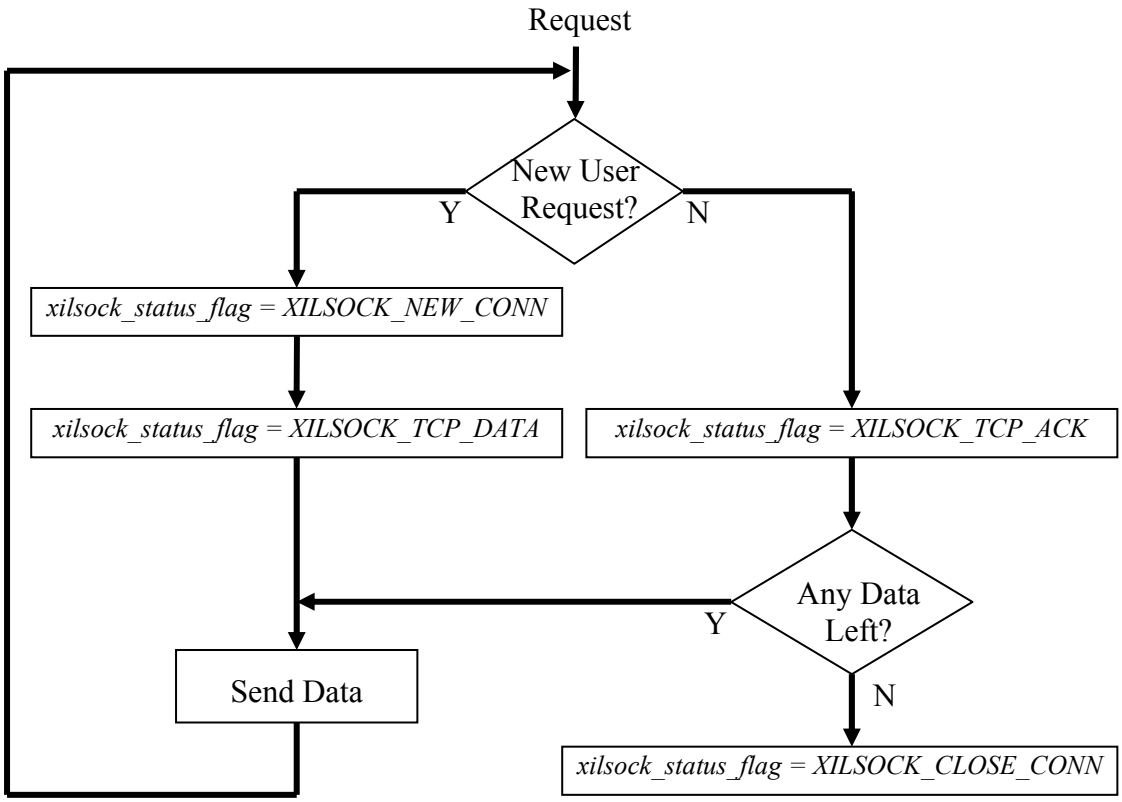


Figure 6. State Machine for XilNet Library

Figure 6 depicts the main states of the TCP/IP state machine for the XilNet Library involved on communication and data transmission as well as the values for *xilsock_status_flag* variable for each state.

3.2.2 FILE SYSTEM MANAGEMENT

LibXilMFS provides users the capability to manage program memory in the form of file handles. The file system can be accessed from the high level C-language through function calls specific to the file system. Also a program called ***mfsgen***, which is provided along with the MFS library, can be used to create an MFS memory image on a host system that can subsequently downloaded to the embedded system memory [1].

A system file, that contents all of the HTML files used by the server, is created in a host system running on Linux. The file system contains the following files:

- web_page.html – webpage returned on an HTTP request from client application.
- mtd_error.html – webpage returned when request's method is different from GET.
- file_created.html – webpage returned to indicate that the client's request has had success.

The generated MFS memory image file **filesys01.mfs** is created on the same directory. Figure 7 shows this procedure:

Figure 7. MFS memory image file created with ***mfsgen*** tool

Once the memory image file has been created, xmd is used for downloading the file to the embedded system memory.

3.2.3 XILINX MICROPROCESSOR DEBUGGER (XMD)

XMD is a tool for debugging programs and verifying systems using the PowerPC and MicroBlaze microprocessors. It supports debugging user programs in different targets such as:

- PowerPC system on a hardware board.
- Cycle-accurate MicroBlaze instruction set simulator.
- MicroBlaze connected to **opb_mdm** peripheral on a hardware board.
- MicroBlaze system running **xmdstub** on a hardware board [1].

XMD Tcl command ***xdownload*** is used for downloading the MFS memory image file to the external or embedded system memory. This action is shown on figure 8.

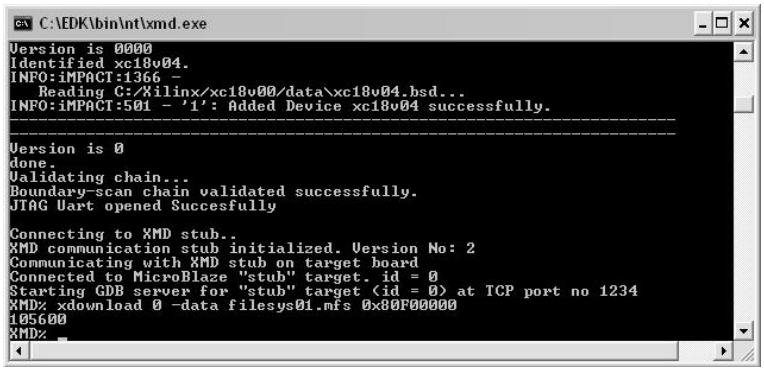


Figure 8. MFS memory image file downloading.

Although XMD is used along with MicroBlaze GDB for debugging, in this work it was used as a tool for giving assistance on downloading the MFS memory image file only and it was never used for debugging purpose.

4 RESULTS

The embedded HTTP server has been tested using a commercial web browser (Mozilla/5.0 on Linux) as client. In response to the client's HTTP request, the embedded server sends the web page shown in figure 9. The web page is designed as

a user interface to collect user information. This information will be used for data processing in a successive stage.

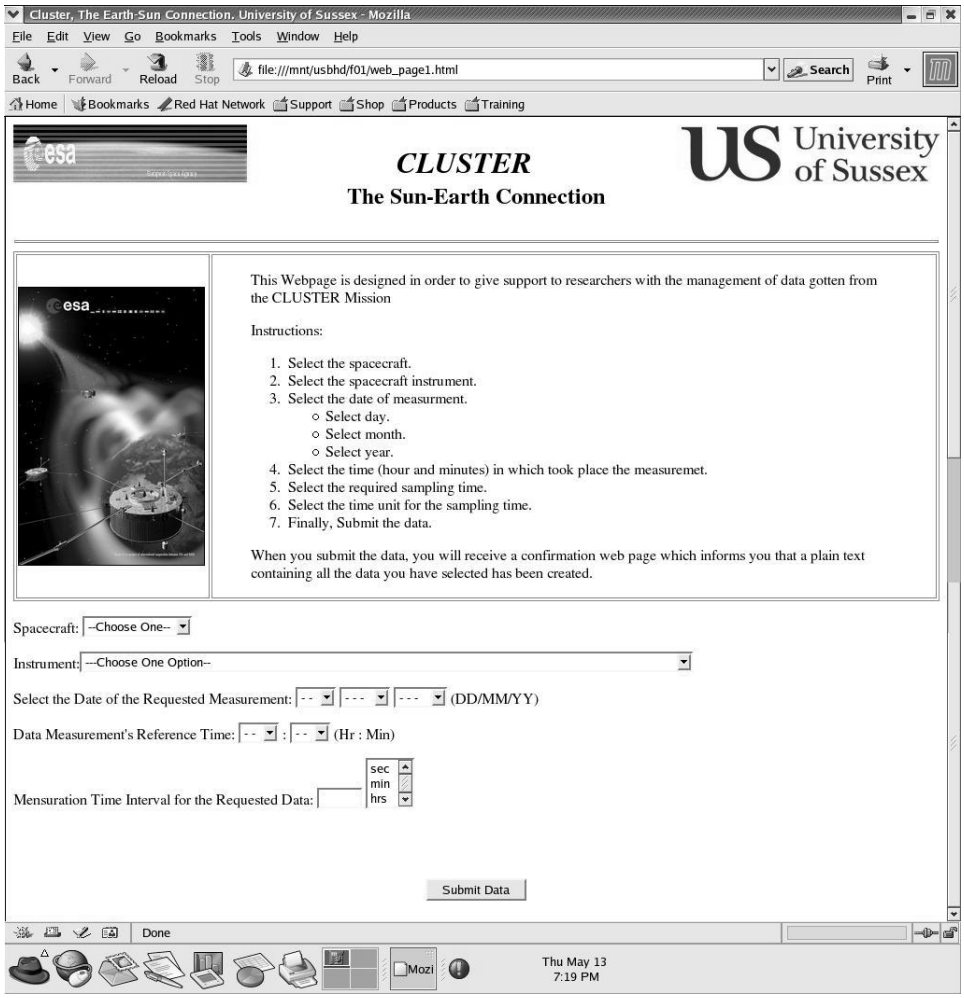


Figure 9. Embedded HTTP server's web page

5 FUTURE WORK

Next stage on the project development is the application code improvement as well as addition of needed hardware in order to make the Embedded System board able to request data from the data base server using information provided by a user analyse data returned by the data base server and forward the analysed data to the original client. Figure 10 depicts a block diagram that explains how the system could be constructed.

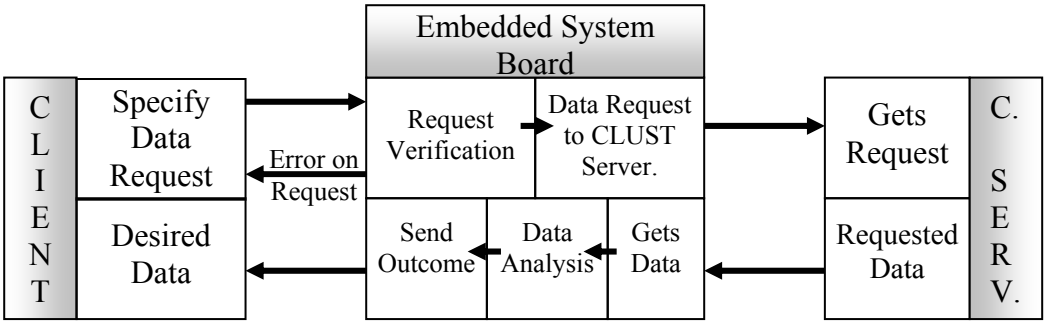


Figure 10. Next stage system block diagram

APENDIX A

- **MicroBlaze** embedded soft core is a 32-bit reduced instruction set computer (RISC) optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGA) [2].
- **LMB** (*Local Memory Bus*) is a fast, local bus for connecting MicroBlaze instruction and data ports to high-speed peripherals, basically on-chip block RAM (BRAM) [3].
- **OPB** (*On-Chip Peripheral Bus*) is a general-purpose synchronous bus designed for easy connection of on-chip peripheral devices. The bus interconnect is a distributed multiplexer implemented as an AND function with the master or slave driving the bus and an OR to combine the drivers into a single bus [4].
- **Data LMB bram cntrlr** is an *lmb_bram_if_cntlr* peripheral that used in conjunction with the *bram_block* peripheral provides a fast BRAM memory solution for MicroBlaze ILMB and DLMB [5].
- **Bram Block** is a parameterizable memory module that can provide memory sizes from 2KB to 128KB [6].
- **OPB UART (*Universal Asynchronous Receiver Transmitter*) Little** is a module that attaches to OPB for interfacing a Hyper Terminal to be used as standard output [7].
- **OPB EMC** is a module that supports data transfer between the On-Chip Peripheral Bus and external asynchronous and synchronous memory devices [8].
- **External Memory** is built by tow Toshiba TH50VSF2581 devices (2M x 16 each) are used to achieve a density of 8M. These devices are external devices on the P160 Communication Module [9].
- **OPB EMAC (*Ethernet Media Access Controller*)** is a soft intellectual property core designed for implementation in some Xilinx FPGA devices that supports the IEEE Std.802.3 Media Independent Interface to industrial standard Physical Layer device and communicates to a processor via an IBM OPB interface. This design provides 10 Mbps and 100Mbps EMAC interface [10].

- **OPB JTAG UART** is a module that mimics UART functionality to MicroBlaze but sends data over JTAG. This module requires xmd (*Xilinx Microprocessor Debugger*) or xmdterm to run on host for JTAG communication [11].

REFERENCES

[1] Xilinx "Embedded System Tools Guide, EDK", May 2003.

[2] Xilinx "MicroBlaze Processor Reference Guide, EDK", April 2003.

[3] Xilinx LogiCore "Local Memory Bus (LMB)", Product Specification, January 2003.

[4] Xilinx LogiCore "On-Chip Peripheral Bus with OPB Arbiter", Product Specification, January 2002.

[5] Xilinx LogiCore "LMB Block RAM (BRAM) Interface Controller", Product Specification, January 2003.

[6] Xilinx LogiCore "Block RAM (BRAM) Block", Product Specification, January 2003.

[7] Xilinx LogiCore "OPB UART Little", Product Specification, January 2003.

[8] Xilinx LogiCore, "External Memory Controller", Product Specification, January 8, 2003.

[9] Memec Design, "P160 Communications Module User Guide", Version 2.0, December 2002.

[10] Xilinx LogiCore, "Ethernet Media Access Controller", Product Specification, January 8, 2003.

[11] Xilinx LogiCore, "OPB JTAG UART", Product Specification, January 8, 2003.

Rapid Core Generation for System Level Design using an Architecture Template.

Darren Reilly[1], Roger Woods[1], John McAllister[1] and Richard Walke[2]
[1]*Department of Electrical and Electronic Engineering, Queen's University, Belfast*
E-mail: {darren.reilly, r.woods, jp.mcallister}@ee.qub.ac.uk

[2]*Real Time Embedded Systems (RTES), QinetiQ Ltd., St. Andrew's Road, Great Malvern, Worcestershire WR14 3PS, UK*
E-mail: walke@signal.qinetiq.com

Abstract

This paper introduces a novel architecture template for targeting hardware implementations on FPGA as part of a system level design flow for heterogeneous platforms. The architecture consists of a fixed scalable structure configurable for a range of algorithms. It provides flexibility in memory sizing and data ordering and can be scaled and configured to meet a particular performance requirement. It provides the designer with high level support for system exploration and system level optimisation which may result in data re-ordering and changes in memory requirements. Whilst providing the flexibility at the system level, it places a restriction on the synthesis and place and route tools at the lower level. In doing this, control is maintained more rigidly at the lower level to prevent simple changes from a high level creating dramatically different results at the low level. A matrix multiplication example is used to demonstrate the flexibility of the architecture.

1. Introduction

Today's systems are becoming more complex given the demands of various industries including the military industry and the mobile technology industry. Moore's Law allows more functionality to be implemented on chip and hence allows for higher levels of parallelism and pipelining to be exploited in order to meet current performance requirements. However, the designer's inability of using the available resources efficiently due to lack of design tools and methodologies is increasing. This is commonly known as the design productivity gap [1].

Given the increasing complexity of signal processing systems, a growing trend is to model systems at a higher abstract level. It is not uncommon for systems to be implemented on multiple different platforms and/or multiple instances of a particular platform. Not only is it difficult to efficiently move from a system model to an efficient implementation in terms of resource and performance but it is a challenge to optimally partition a system unto multiple platforms consisting of hardware and software. System partitioning is carried out once a system has been captured and functionally verified at a high level.

Software compilers and assemblers have reached maturity and allow high level code to be implemented on various processors. However, support for developing efficient dedicated hardware architectures specific to the algorithm that make efficient use of on-chip resources is lagging behind. The use of FPGA's is becoming more common in

signal processing as they provide the capability of implementing specific architectures without the time consuming design and verification process required for ASIC's.

Tools such as SystemC [2] or Handel-C [3] can take algorithmic descriptions onto FPGA. When using tools like these, limited control is maintained over synthesis or placement and routing and therefore minor high level changes can lead to changes in the layouts at the low level. This gives rise to different signal routing at the low level and as a result can lead to varying performance results leaving it difficult for the system designer to gain any feel of how an implementation will perform after making even the smallest of changes at the high level.

In this paper an architecture template for FPGA implementations is introduced. The main aim of this work is to provide an architecture template on which to map complex DSP functions. The architecture needs to be flexible so that it can be scaled and refined to a minimum to do exactly the job that is required of it. The motivations for developing such an architecture template are:

- to facilitate the rapid generation of efficient and controlled architectures without the dependence on expensive tools.
- to alleviate the designer from having to deal with low level issues like interconnect or control signals.
- to provide a mechanism of facilitating already existing techniques and algorithms for optimising hardware in a controlled manner for optimum system level usage.

To achieve this, it is intended that control is maintained at the lower level to reduce the impact of the synthesis and place and route tools. Generic control signals are required, such as clocks and resets, as well as a well defined communications interface to allow communication to take place between cores and other platforms.

The paper is structured as follows. Section 2 introduces dataflow as a model of computation which we use to capture the functionality of complex systems at an abstract level. In Section 3 we highlight issues introduced when partitioning and mapping such complex systems to heterogeneous platforms. An architecture template is introduced in Section 4 as a mechanism to ease the flow from algorithm to architecture whilst maintaining control of the implementation and providing the necessary flexibility for system exploration and support for system level design. In Section 6 we provide an example mapping for matrix multiplication to demonstrate the capabilities and flexibility of the architecture.

2. Dataflow

A system level design flow for heterogeneous platforms is being developed as part of a collaborative project between QinetiQ, BAE Systems and Queen's University Belfast [4]. The model used for capturing system functionality in this project is dataflow as it provides a way of capturing functionality at an abstract level without any detail about the implementation platform.

Dataflow works on the concept of *tokens* and *actors*. A token can be anything from a single scalar to a multi-dimensional data structure. Actors will fire whenever the required

data resides on its inputs. On firing, actors will consume tokens on each of their inputs, process them and produce tokens on each of their outputs depending on the functionality of the actor. Actors are connected by infinite queues. An example actor with tokens is shown in Figure 1 below.

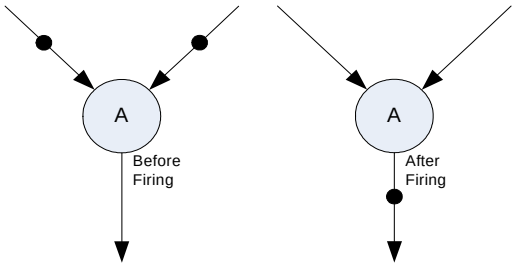


Figure 1: Example dataflow graph

In this example actor A has one token on each of its inputs. Assuming that the required conditions for the actor to fire is one token on each of its inputs, it can then fire and produce one or more tokens on its output depending on the actor operation. Once system functionality has been captured, the system can be simulated and functionally verified using GEDAE [5]. GEDAE is a tool for capturing system functionality at a dataflow level and a screenshot is shown in Figure 2 along with a sample scope output.

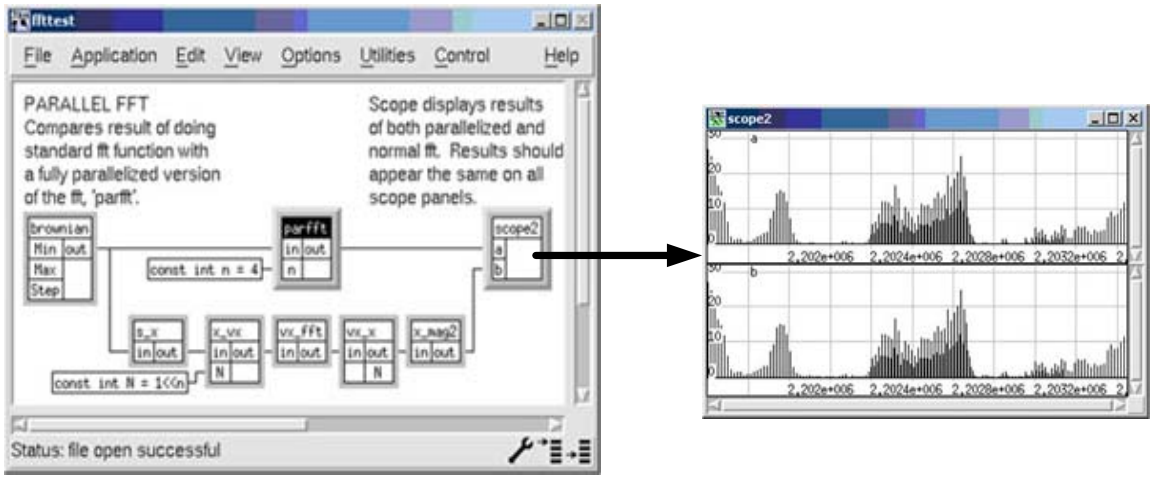


Figure 2: Sample screenshot of GEDAE

The tool provides a powerful means of scheduling and partitioning dataflow graphs onto multiple processors but has limited support for hardware implementations on FPGA. Within this project capability has been provided to generate netlists from user specified parts of the dataflow graph. This netlist can then be optimised and translated to VHDL for use in synthesis and place and route tools for final implementation. Various issues arise from system level design flow and these are highlighted in the next section.

3. System Level Issues

When implementing systems, decisions have to be made on the system platforms which may include a configuration of processors and FPGAs or ASICs and the partitioning of the algorithms. It is a difficult and complex task to optimally partition a system.

Once partitioned, communication becomes an issue as it dictates the amount of cross platform data transfer. With a complex system, it is not uncommon to have data flowing in both directions between platforms which can lead to large latencies and poor utilisation of resources. Decisions are required as to the communications fabrics such as RocketIO, Ethernet or simply wire connections between the platforms. Many protocols also exist to implement communication types over these fabrics such as TCP or UDP which introduce their own overheads and require large amounts of data to be buffered before sending across the communication channel thereby increasing latency and decreasing utilisation.

Buffering is also used to improve overall system performance in terms of resource utilisation and throughput by balancing the delays in between processor paths in a dataflow graph so that each actor may be fully utilised and throughput increased. Whilst this allows the sampling rate to be met, it is not usually enough to simply increase buffering to get the required throughput. Parallelism and pipelining are normally used along with buffering to meet throughput requirements. The cost of pipelining is that it adds extra latency in the system and the exploitation of parallelism may have an impact on the communications required as it is typical that multiple simultaneous accesses to memory will be required and therefore data duplication or distribution will be needed. This adds to the buffering in a system.

Whilst these are all critical to system performance in terms of speed, throughput and area, from a system level the designer has the ability to modify and optimise graphs which can reduce communication requirements by balancing computation across the processor network and re-ordering data so that overall system buffering may be minimised. Support for such optimisations and control is not currently supported at a low level. In the next section we introduce an architecture template to support and handle these issues whilst alleviating the designer from direct involvement in low level detail.

4. Architecture Template

Various issues have been highlighted in the previous sections regarding FPGA interconnect, mapping and partitioning along with other system level issues. With regard to lower level issues, the designer has multiple ways of implementing the required functionality on FPGA. Tools such as SystemC [2], Handel-C [3], and Compaan [6] can produce FPGA implementations and interact with lower level tools but they exhibit limited control over the layout and routing on the FPGA which can lead to small changes at an algorithm or system level impacting performance at the low level. A control mechanism is required to prevent changes at the system level having such an impact on performance.

The concept of developing circuit architectures for specific implementations has been explored to some extent in the Imagine Stream Processor [7] and the picoArray [8]. The Imagine Processor is mainly targeted towards image processing whilst the picoArray is

mainly targeted at 3G cellular base stations which require performance of the order of hundreds of GOPS. These solutions are fixed architectures and in some cases will provide more functionality in terms of memory or larger word sizes than is required. This is a waste of resources and the flexibility does not exist to refine these. On the other hand the word sizes may not be big enough in which case complexities are introduced in providing a work around. The architectures do not have the flexibility required at a system level to provide an optimal solution. The template design here offers more flexibility.

Another way of implementing the actors in a dataflow graph is with the use of existing IP cores. However, it is a difficult task to integrate existing cores into a system as they are unlikely to conform to a dataflow model or any other model of computation. It would be necessary to build a wrapper specific to each core, which would be a time consuming and error prone task. In addition the core might not have the required programmability to allow it to be efficiently used at the dataflow graph level.

As a result of the issues highlighted and the lack of system level support at a low level, we have deemed it essential to put restrictions in place to reduce the scope of synthesis and place and route tools. By doing this, more control is maintained in the lower level to ensure efficiency in both resource utilisation and speed. The proposed solution for this is an architecture template. This provides us with a rigid structure that is configurable and can be scaled to the problem size. The idea of this is to capture the generalised problems for system level design and in doing allows control to be maintained over these issues. The proposed architecture template is shown in Figure 3 below.

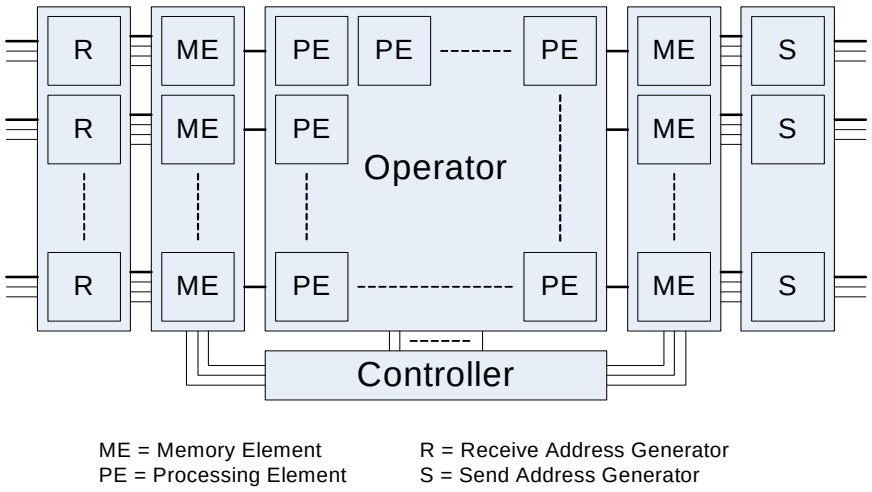


Figure 3: Proposed architecture template

The following sections provide more detail on the various aspects of the architecture and how they provide support for the system level flow.

4.1. Interface

As this architecture is to be targeted from a system level flow it is important that the interface is in keeping with the concepts of dataflow. Dataflow works on the basis of transferring and processing tokens where tokens may be anything from scalars to multidimensional arrays. The architecture consists of both input and output dual port memories. This isolates the external communications from the internal processor providing a dynamic interface in keeping with dataflow. The layered structure of the architecture is shown in Figure 4. The memories are provided so that the consumption and production concepts of dataflow can be accurately implemented. Providing this interface allow actors to be mapped to the architecture template and connected in a similar manner as in the dataflow graph.

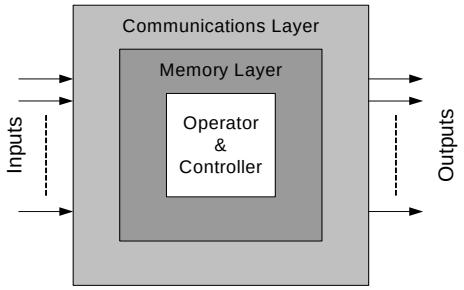


Figure 4: Layered architecture

Throughout our examples we have used a simple two-way handshake to communicate between cores. This involves having data, transmit and receive signals. Whenever both receive and transmit signals are high the data is transferred.

4.2. Memory

Each input and output has a memory, which is required so that the architecture is in keeping with the dataflow model and allows the concept of tokens to be realised. To provide flexibility in scaling the memory for multiple processors and the transfer of vectors of various sizes, the structure in Figure 5 has been proposed.

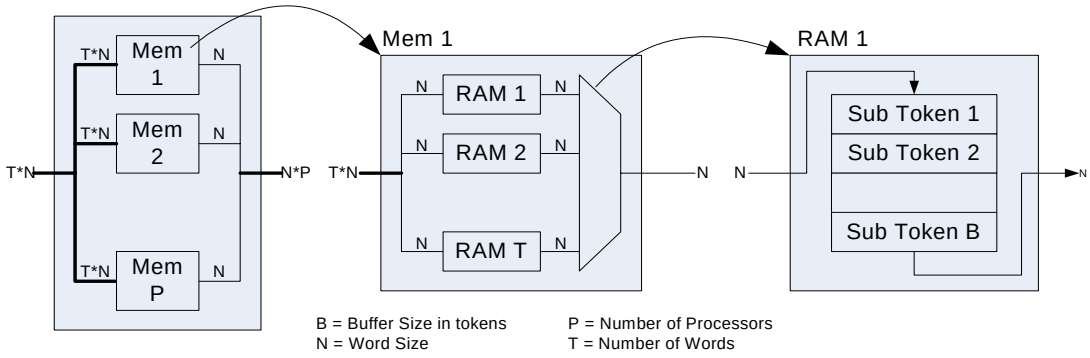


Figure 5: Memory structure

The memory is structured such that if there are *P* processors in the operator, the data will be duplicated *P* times. The reason for this is that each memory has 1 read port and with multiple processors in the operator all requiring different data from memory, it is necessary to duplicate the memories so that each processor can independently access the required data without any blocking from other processors. If a vector consisting of *T* scalars is transferred to an input in a single time cycle, *T* RAM's as shown will be inferred, one for each scalar. This is done so that access is provided to single scalars within a vector. Each of the RAM's are broken into sub tokens. The buffer size measured in tokens dictates the size of these RAM's.

The memory is implemented using either block RAM's or distributed LUT based RAM provided on the FPGA depending on the size instantiated. The memories are dual port, one side read, the other side write. This allows both communications and processing to take place simultaneously but only if the memory is capable of storing at least two tokens. The reason for this is that whilst the operator/controller is accessing a token, the receive and send processors cannot access that same token space. However, if there are two token spaces in memory, whilst one is being operated upon, then the other can be written into or read from.

4.3. Controller

The purpose of the controller is to schedule the data onto a processor or processors within the operator. As a large percentage of DSP algorithms can be expressed in the form of a nested loop program (NLP) due to their recursive nature, it was decided to implement the controller with cascaded counters so that the loops in the algorithms can be directly mapped to the counters as demonstrated in Figure 6.

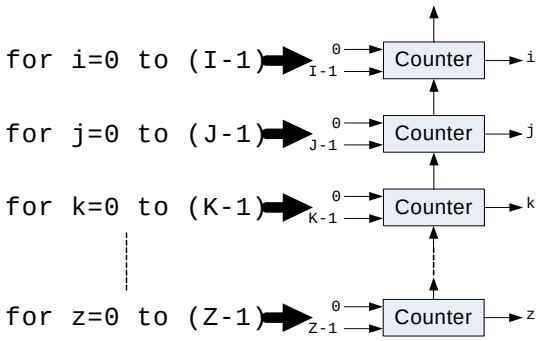


Figure 6: Controller structure

The inner most loop is mapped to the least significant counter with the outer most loop being mapped to the most significant counter. This method provides a fully scalable technique of handling one to many loops in an algorithm and can also be parameterised at runtime by changing the loop bounds if necessary. The controller provides an enable signal to the operator based on the condition that the input and output memories have data and space respectively. This is shown in Figure 7.

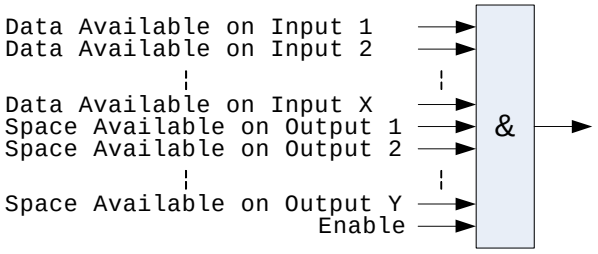


Figure 7: Operator enable signal

This signal is simply generated by a large AND function in the controller. A reset signal is provided to reset the counters to their lower bounds and initialise the system back to a known state.

4.4. Operator

The operator consists of a number of identical processors dictated by the designer depending on the amount of parallelism that is necessary to meet the performance requirements in terms of speed. The communications between the processors if any is implied by the algorithm to be implemented. The processor itself is also derived from the algorithm. For a matrix multiply core, the processor is a simple multiply accumulate (MAC) circuit. The operator is viewed as a black box with a latency in which a correct result will be produced if the data is scheduled by the controller correctly onto it. The schedule as seen in the previous section is generated directly from the loops in the algorithm. This is demonstrated more clearly in the next section where we provide example mappings.

5. Example Mappings for Matrix Multiplication.

Matrix multiplication has been chosen as an example because of its computationally complex nature and because it is a core component in many DSP applications. It highlights the scalability aspect of the architecture and the need to carry out more than one operation in parallel. Many techniques exist for matrix multiplication implementations [9], [10] which have different scheduling methods to produce the same end result only with various data/process orderings.

The generalised algorithm for matrix multiplication in terms of M, N and P can be given in the form of a nested loop program (NLP) as shown below.

```
for i=0 to (P-1){
  for j=0 to (M-1){
    C[i,j] = 0;
    for k=0 to (N-1){
      C[i,j] = C[i,j] + (A[i,k] * B[k,j]);}}}
```

Before starting the mapping process it is better to bring all the functionality together into the inner loop (shown below). The operator is then derived by the functionality specified within this inner loop. For our examples we will assume M=2, N=3 and P=4.

```
for i=0 to (P-1){
  for j=0 to (M-1){
    for k=0 to (N-1){
      if k=0 then{
```

```
      C[i,j] = 0;}
    Else{
      C[i,j] = C[i,j] + (A[i,k] * B[k,j]);}}}}
```

5.1. Matrix Multiplier with 1 processor

The simplest case to look at is mapping the algorithm to a single processor. As mentioned in the previous section the controller is constructed from cascaded counters. It can be easily identified from the algorithm above that 3 counters will be required, 1 for each loop in the algorithm. The mapping and bounds are shown in Figure 8 for the loops in the algorithm.

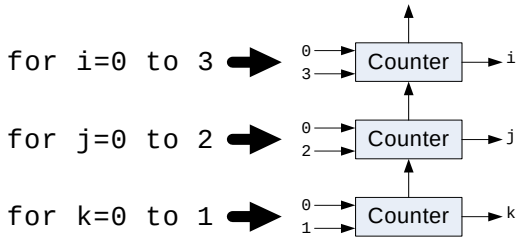


Figure 8: Mapping of loops to counters

The bounds of the counters have been directly derived from the bounds of the 'for' loops and so the address for Memory A, Memory B and the output memory are functions of (*i*,*k*), (*k*,*j*) and (*i*,*j*) respectively. The 'if' condition in the algorithm implies a condition to be set for the processor. This is simply a comparator in the controller that sets a signal high when *k*=0 as per the condition in the algorithm. The memory sizes are given in Table 1. It is assumed here that both inputs and outputs are required to have a buffering capability of two tokens each.

Port	Token Size	Buffer Requirement	Memory Size
Input A	M*N	2	2*M*N
Input B	N*P	2	2*N*P
Output	M*P	2	2*M*P

Table 1 Memory requirement for matrix multiplier

The processor for the operator is derived from the inner loop functionality. Also by changing the 'for' loops so that the condition for *C*[*i*,*j*] = 0 is in the inner loop, it makes it more simple to see the overall processor structure as shown by the loops below.

```
for i=0 to P{
  for j=0 to M{
    for k=0 to N{
      if k=0 then{
        Reg[0]Inp = 0;
      }
      else{
        Reg[0]Inp = Reg[0]Out + (A[i,k] * B[k,j]);
```

```
      }
      C[i,j] = Reg[0]Out;
    }
  }
}
```

Figure 9 below shows the structure of the processor to carry out this functionality.

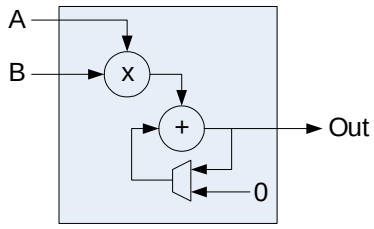


Figure 9: Processor

This was a simple case for a single processor. However it is simple to expand upon this and generate solutions for multiple processors. The first step in doing this is to modify the 'for' loops to highlight the parallelism available. This is demonstrated in the following section.

5.2. Matrix Multiplier with more than one processor

To refine the algorithm in the first section, let us replace the functional part within the inner loop with 'Operator' as follows.

```
for i=0 to P
  for j=0 to M
    for k=0 to N
      C[i,j] = Operator([i,k],[k,j]);
```

It is straightforward to elaborate these for loops for various processor numbers by applying transformations such as loop unrolling or skewing. This can be seen from some of the work done by Leiden University in their Compaan tool [6]. They effectively split the job between the processors available.

The following NLP's provide are various loop unroll transformations suitable for M, N and P processors respectively.

M Processors

```
for i=0 to P
  for k=0 to N
    C[i] = Operator([i,k],[k,0 to M]);
```

N Processors

```
for i=0 to P
  for j=0 to M
    C[i,j] = Operator([i,0 to N],[0 to N,j]);
```

P Processor

```
for j=0 to M
```

```
  for k=0 to N
    C[0 to P,j] = Operator([0 to P,k],[k,j]);
```

As before, these loops map directly to the counters in the controller. It can be observed from the algorithms shown that there are only 2 loops in these algorithms as 1 loop has been fully unrolled in each case. Unrolling all of the loops would mean that the controller would not require any counters and the operator would be an array of processors similar to a systolic type implementation. The operator can be expanded to identify the processors contained within it. Taking the example for *P* processors, the expanded algorithm would be as follows.

```
for j=0 to M
  for k=0 to N
    C[0,j] = Processor([0,k],[k,j]);
    C[1,j] = Processor([1,k],[k,j]);
    C[2,j] = Processor([2,k],[k,j]);
      "          "          "
    C[P,j] = Processor([P,k],[k,j]);
```

The processor is identical to that previously derived only there are more of them in the operator.

6. Conclusion

In this paper we have introduced the concept of an architecture template to provide system level support. We have demonstrated the ease of mapping algorithms to the architecture and gave a quick overview of how the architecture can be optimised. It has been clearly shown that the mapping process is relatively simple and the optimisations available lead to a powerful and controlled mechanism for system level design and implementation. Preliminary results of the architecture on a Xilinx Virtex 2 Pro indicate clock speeds above 170MHz whilst resource usage for the single processor example is in the region of 250 LUT's which is under 1% of the total LUT's available on the FPGA. It should be noted that the LUT's were also used to implement the memories in the architecture rather than the available block RAM's.

7. Acknowledgements

This work has been sponsored by the UK Ministry of Defence Corporate Research Programme, and the Department of Education and Learning (DEL) NI. This work has been undertaken in collaboration with QinetiQ Ltd., Great Malvern, UK and BAE SYSTEMS ATC, Gt. Baddow, UK.

References

1 D. Edenfield, A.B. Kahng, M.Rogers, Y. Zorian, "2003 technology roadmap for semiconductors", *IEEE Comput.*, vol. 37, no. 1, pp. 47-56, January 2004.

2. SystemC: http://www.systemc.org.

3. Handel-C: http://www.celoxica.com.

4 J. McAllister, Y. Yi, R. Woods, R. Walke, D. Reilly, K. Colgan, "Design Technologies for DSP Algorithm Implementation on Heterogeneous Architectures",

Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, vol. 5205, pp. 585-596 August 2003.

5. GEDAE: http://ww.gedae.com.

6 Bart Kienhuis, Edwin Rijpkema, and Ed F. Deprettere, "Compaan: Deriving Process Networks from from Matlab for Embedded Signal Processing Architectures", *In Proc. 8th International Workshop on Hardware/Software Codesign (CODES)*, San Diego, CA, USA, May 3-5 2000.

7 Ujval J. Kapasi, William J. Dally, Scott Rixner, John D. Owens, Brucek Khailany, "The Imagine Stream Processor", *Proc. IEEE Int'l Conf. on Computer Design*, pp. 282-288, September 2002.

8. picoArray: http://www.picochip.com.

9 Prassana Kumar V. K., Tsai Y.: "On Synthesising Optimal Family of Linear Systolic Arrays for Matrix Multiplication", *IEEE Trans. Comput.*, vol. 40, no. 6, pp. 770-774, June 1991.

10 I. V. Ramakrishnan an P.J. Varman, "Modular Matrix Multiplication on a Linear Array", *IEEE Trans. Comput.*, vol. C-35, no. 11, pp. 952-958, 1986.

A transistor-level delay-insensitive register file for deep sub-micron SOC

Yijun Liu

APT Group
The Department of Computer Science
The University of Manchester
Manchester M13 9PL, UK
yijun.liu@cs.man.ac.uk

Abstract

With the rapid development of deep sub-micron CMOS technology, wire and gate delays are accounted to have equal or nearly equal effect on circuit behaviour. This phenomenon presents a great challenge to the traditional synchronous VLSI CMOS design, which requires that all the delays of gates and wires are predictable. In big system-on-a-chip circuits, global clock signals also waste a lot of silicon area and power consumption. Delay-insensitive logic design provides the most robustness under any arbitrary delays. Moreover, the absence of clock signals gives delay-insensitive circuits an inherent advantage of low power consumption. The gate-level fully delay-insensitive register files have the shortcomings of big size and power hunger. We propose a transistor-level delay-insensitive register file in this paper, which occupies less silicon size than any other delay-insensitive register files published so far.

1 Introduction

System-on-a-chip (SOC) systems are the direct products of the dramatically down-scaling CMOS technology as predicted by Moore in 1975 [1]. No signs of slowing down of Moore's law are seen so far. The semiconductor technology has progressed to an era that millions of transistors are integrated in a single chip, which only costs a few dollars and consumes several watts. However, with metal wires and vias getting smaller and smaller, wire and gate delays are accounted to have equal or nearly equal effect on circuit behaviour and the delays become more and more unpredictable. So the orders of signals are not promised anymore and it is very difficult to use an extra clock signal to indicate the validation of data. These delays, including propagation delay, lateral coupling and crosstalk-induced delay, present a big challenge to deep sub micrometer SOC circuits. CMOS designers use symmetric routing, such as H-tree, to gain a precise system global timing. However, with this solution, the global clock consumes a lot of silicon area and power [2]. Asynchronous logic designers offer another solution called delay insensitive method. In delay-insensitive circuits, no clock signals are used to direct the data flow. Moreover, delay-insensitive circuits can operate correctly in spite of the unknown delay in wires as well as in gates, thus having the most robust characteristics [3].

In this paper, we propose a delay-insensitive register file, which is smaller than other register files we can find in published papers. The remainder of this paper is arranged as follows: Section 2 introduces background knowledge on asynchronous logic; Section 3 lists some related works, including synchronous logic files based on the 6 transistors static memory cell and gate-level delay-insensitive register files; Section 4 proposes the design of a novel transistor-level delay-insensitive register file. Section 5 gives the result of experiments on the proposed register file. Finally, conclusions are drawn in Section 6.

2 Asynchronous logic

The global clock signal is used as the timing reference in synchronous design to control dataflow and communication of the datapath. Asynchronous design, on the other hand, employs locally generated control signals as timing reference. Delay-insensitive circuit is a kind of asynchronous circuit, in which, a redundant logic representation is used instead of a conventional logic representation which contains timing information. Some codes

in the redundant logic represent valid data and some represent empty data. The valid data and empty data are separated from each other. A typical delay-insensitive communication channel is illustrated in Figure 1.a.

An abstract view of the communication between the sender and the receiver is as follows:

- The sender issues a valid codeword;
- The receiver absorbs the codeword and pulls acknowledge high indicating “data received”;
- The sender responds by issuing a empty codeword;
- The receiver uses the empty codeword to initialize itself for the next communication cycle and pulls acknowledge low.
- After the sender detects a low signal in acknowledge, it may initiate the next communication cycle.

As can be seen, this protocol is very robust. A reliable communication can be built between the sender and the receiver regardless of delays in the connection.

The most commonly used delay-insensitive logic representation is called dual-rail code [4], as shown in Figure 1.b. A dual-rail code uses two wires (say d.t and d.f) to represent one bit. “00” represents empty; “01” represents valid 0; “10” represents valid 1 and “11” is not used.

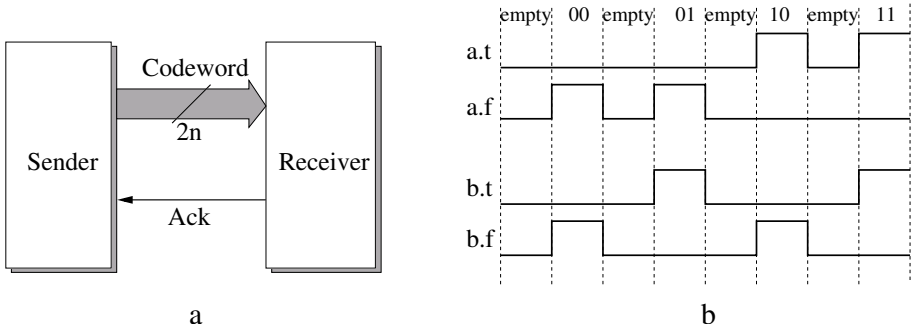
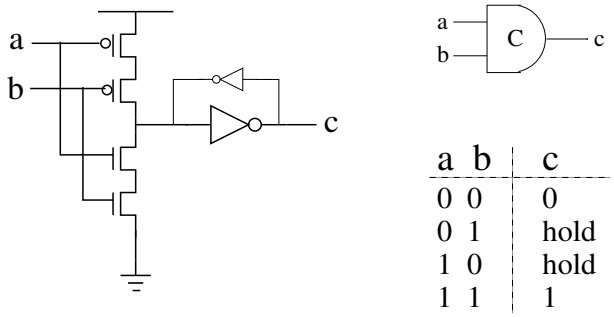


Figure 1: A delay-insensitive communication channel using dual-rail coding

2.1 Indication theory

A synchronous circuit is controlled by a global signal. Between two clock-ticks, the combinational logic block may exhibit some temporal results called hazards. In asynchronous circuits, the situation is different. The absence of a global signal means that the redundant logic representation contains timing signals. So signals are required to be valid all the time and hazards must be avoided. Signal transitions that are not indicated or acknowledged in other signal transitions should be avoided because they potentially cause hazards [3]. The Muller C-element [5] shown in Figure 2 provides a good example of indication theory.



a	b	c
0	0	0
0	1	hold
1	0	hold
1	1	1

Figure 2: A symmetric Muller C-element

As can be seen from Figure 2. Only when both a and b are 1 does c become 1; only when both a and b are both 0 does c become 0. If one of the inputs of the Muller C-element is 0 and another input is 1, the output will hold on its former state. So if we find there is a 1 on the output of a C-element, we are sure that a rising transition happens on each input of the C-element. We say that the Muller C-element indicates when both inputs are 1. The Muller C-element also indicates when both input are 0.

For a 2-input OR-gate, it only indicates when both input are 0, because when seeing the output change from 0 to 1 it only knows that at least one input is 1, but it does not know exactly which. However, if we can promise that at most only one of the inputs is 1 at a time (which is called 1-hot or 1-of-n coding), we can use an OR-gate to indicate the signals. For example, only one word line of a register files can be valid at a time. So we use a big OR-gate to indicate the completion of the column decoder.

As a result, a Muller C-element indicates when all of its inputs are valid and an OR-gate can be used to indicate 1-hot code. We will address this issue in Section 4 again.

3 Related work

3.1 Synchronous register files

A typical synchronous register file [6] employs cross-coupled memory cells because cross-coupled inverters are static and robust as state holders. Pass nMOS transistors are used to connect memory cells with read and write ports. A register cell with two read ports and one write port is illustrated in Figure 3. The bit lines are normally run as complementary signals needed for sensor amplifiers in big register files. In order to minimize the silicon area of register file, the pMOS is generally small. As an nMOS transistor is poor at passing a one, the evaluation of bit lines is executed by pulling the bit lines from 1 to 0. Thus before each reading action, the bit lines should be charged to high.

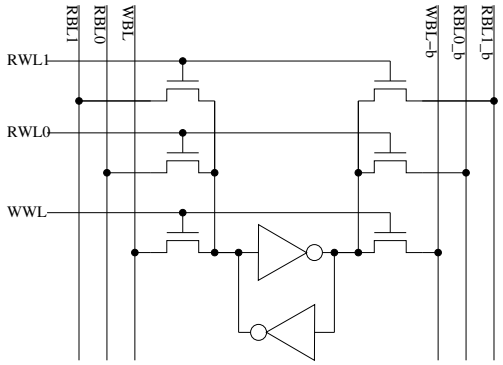


Figure 3: A typical synchronous register memory cell

3.2 Delay-insensitive register files

There are not many research works we can find about delay-insensitive register files. One of those is a gate-level register file used in two delay-insensitive microprocessors: [7] and [8], which is shown in Figure 4. This is a single-bit register cell with one write port and one read port. The two cross-coupled nor-gates ($g1$ and $g2$) are used as a memory cell to store the input data. The two and-gates ($g3$ and $g4$) are used to detect the completion of the storing action of the memory cell. If the inputs of the memory cell are the same with the corresponding outputs, one of $g3$ and $g4$ will issue a one on its output, which will be propagated by the or-gate ($g5$) as an indication of the completion of the writing action. The two and-gates ($g6$ and $g7$) are controlled by req signal to propagate the data in memory cell to the output. If both $d.t$ and $d.f$ are 0, the indication signal ack will become 0. Thus the register cell can execute a delay-insensitive communication with the left writer and the right reader.

The register cell shown in Figure 4 presents a good example of delay-insensitive circuits because it is very easy to understand. However, this gate-level implementation, which contains 18 nMOS transistors and 18 big pMOS transistors (it needs 6 nMOS transistors and 6 pMOS transistors more to support another read port), occupies big silicon area and consumes significant power.

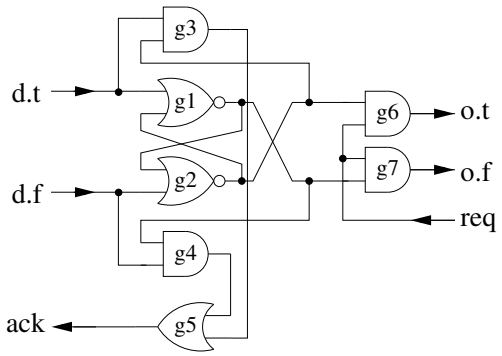


Figure 4: A gate-level delay-insensitive register memory cell

Most of the asynchronous register files do not promise the delay of writing action. They use an extra delay block to match the delay of writing the memory cells, which makes their designs not fully delay-insensitive.

4 A novel transistor-level delay-insensitive register file

In this section, we propose the design of a tight delay-insensitive register file based on the traditional cross-coupled inverters in transistor level.

4.1 Single-bit register cell

A single-bit transistor-level register cell with one read port and one write port is illustrated in Figure 5. The two cross-coupled inverters are used as memory cell to save the input data. The inputs are two complementary signals called wbl and wbl_b, represented in a dual-rail code. The same data representation is used by the outputs. WWL is the write word line controlling the inputs of the memory cell. RWL is the read word line controlling the outputs. The two transistors $T1$ and $T2$ act as writing completion indicator. Only after the memory cell finish storing the input data, will the signal Ack be pulled high. If the Ack is pulled down again, it means that the write bit lines have returned to zero and indicates that the sender can issue another input codeword. Ack only indicates write bit lines and the completion of memory cell. It does not indicate the write word line — WWL.

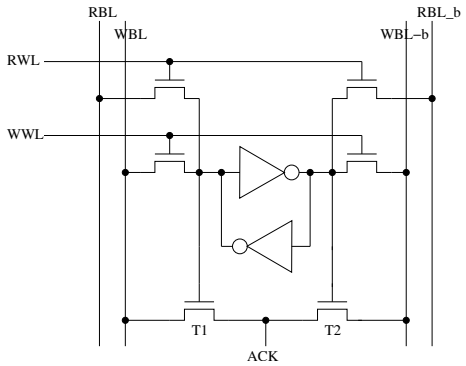


Figure 5: A transistor-level delay-insensitive register memory cell

The register cell shown in Figure 5 is very compact, and needs only two transistors more compared to the normally used synchronous register cells to generate the acknowledge signal of writing action.

4.2 The register file design

We designed a 32×8-bit register file for an 8-bit delay-insensitive microprocessor. The schematic of the register file is shown in Figure 6. The register file contains one write port and two indistinctive read ports. However, in

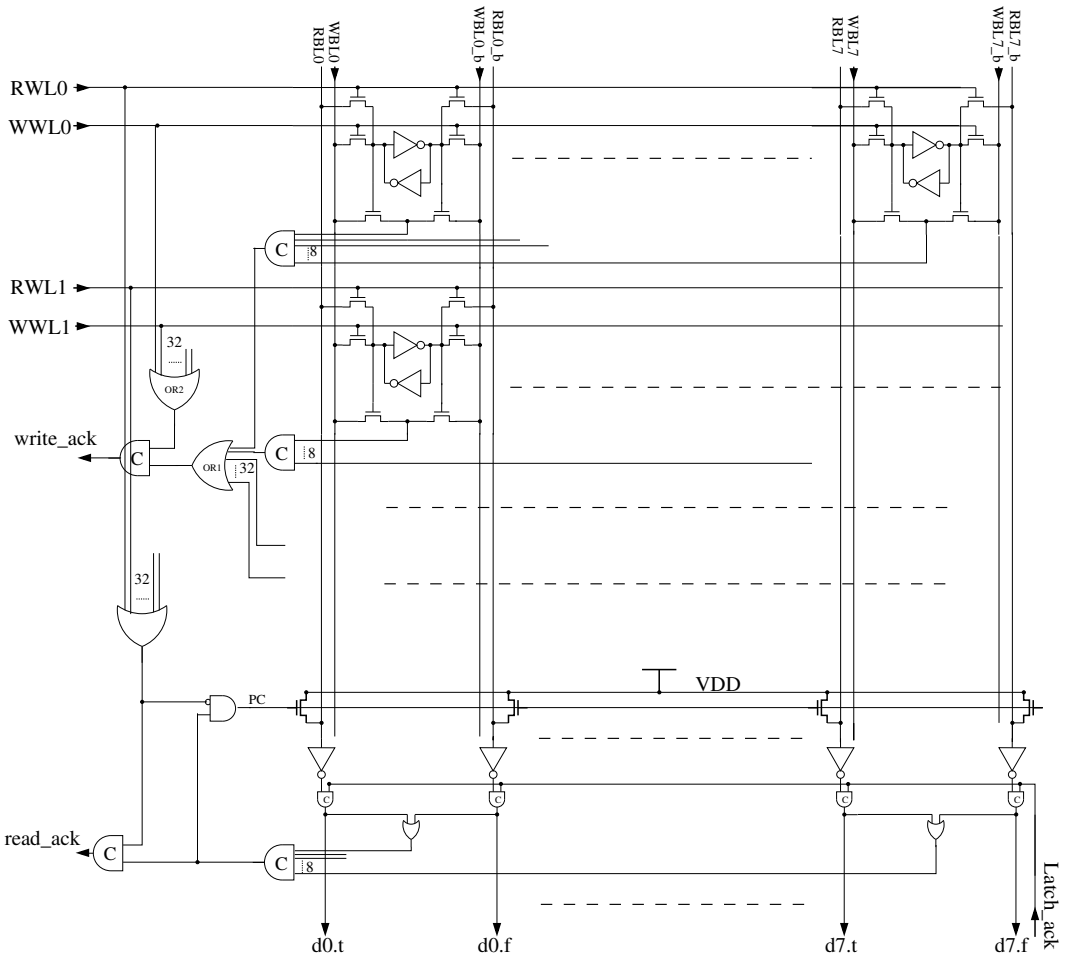


Figure 6: The schematic of the proposed register file

order to simplify the diagram, we only show one read port in the figure. The read port and the write port are independent and the environment controls the register file to avoid conflicts between reading and writing.

The writing action happens as follows:

The write circuit sends 8 bits dual-rail code and 32 word line bits (1-of-32 coding) to the register file. One of the word lines is driven high to select the corresponding bit row. 8 memory cells in that row copy data from dual-rail bit lines. After they finish storing data, the Acks of all the memory cells in that row become high, which make the output of 8-input C-element as 1. Then the 1 on the output of the 32-input OR-gate ($or1$) indicates one row of the register file has finished writing. Because the word line is not indicated by Ack as we discussed in Section 4.1, we need another 32-input OR-gate ($or2$) to indicate the validation of word lines. Then we use a 2-input C-element to combine these two indication signals and send the completion signal to the write circuit. The write circuit detect a 1 on *write_ack*, it pulls all the write bit lines and write word lines down to zero and prepares for the next writing.

In the reading circuit, a row of two-input C-elements is used as latches to store the data for output. *Latch_ack* controls the output of the register file. 8 two-input OR-gates and one 8-input C-element compose the completion detector to indicate the output of the read port. As we discussed in Section 3.1, read bit lines need to be precharged to 1 before each reading. Precharging happens when all the read word lines are invalid (0) and the output is valid and stored in the C-element latches. An AND-gate is used to generate the precharge signal. NMOS transistors are used here instead of pMOS transistors because nMOS is poor at passing 1. The bit lines are not fully precharged to *VDD* to increase the speed and minimize the power consumption. The reading action of the register file can be described by a FSM shown in Figure 7.

From Figure 6, we can see that in each row of memory cells, there are 8 horizon Ack wires and for 32 rows, there are $32 \times 8 = 256$ horizon wires. These wires waste a lot of silicon area. In order to save the silicon area, we

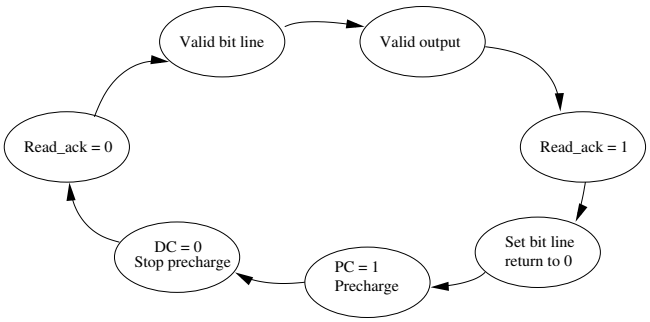


Figure 7: The FSM description of reading action

split the C-element and distribute nMOS and pMOS transistors to each memory cell, as shown in Figure 8. By doing this, only 64 horizon wires are needed instead of 256. However, 8 transistors connected in serial make the circuit very slow. We separate the memory array into 2 parts. Each part is 4 bits wide.

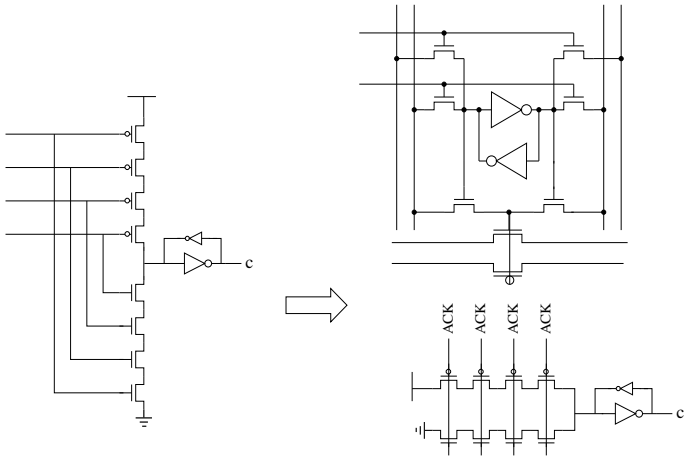


Figure 8: Split the C-element

5 Experimental results

The implementation of the proposed delay-insensitive register file was simulated using HSPICE, with the netlist exported from Cadence Composer-Schematic. A Verilog test harness was constructed to present the input stimuli and to check that the outputs were current.

We implemented the delay-insensitive with three word line decoder (one for writing and two for reading) and two reading output latches. The whole system was analyzed under the conditions of a 1.8 volt supply and 27 °C temperature on a 0.18 micron CMOS technology. The delay for a writing operation (including decoding and writing) is 1.49 ns and the delay for a reading operation (including decoding, precharging and latching) is 1.91 ns.

The most interesting characteristic of the proposed register file is its robustness and self-adaptive timing signals. The register gives the correct results in spite of big supply voltage swing, even below the transistor threshold voltage. This is also demonstrated by another delay-insensitive microprocessor [9]. No effort is needed to adjust the speed of the global signal to indicate the current timing reference because the redundant logic contains timing information and it will automatically match the delay of logic block under different voltage.

6 Conclusion

In this paper, we proposed the implementation of a transistor-level delay-insensitive register file. This register file is much smaller than those gate-level delay-insensitive register files we can find. We plan to use the proposed register file in an 8-bit delay-insensitive microprocessor for sensor net, which spends most of (more than 99%) its time in sleeping. The absence of global signal give the microprocessor an inherent advantage in low power especially when it spends most of its time in idle state. The current consumption of synchronous microprocessors varies from 10 to 100 uA to maintain the clock signals during idle periods. Asynchronous microprocessors, on the other, do not need to support the global signals, so they can have zero standby power consumption except for the leakage current [10]. The robust characteristic of delay-insensitive circuits makes them very easy to be combined with the voltage-swing technique, which is a very commonly used technique in low power systems.

References

[1] G. Moore, "Progress in digital integrated electronics", in *Proceeding of 1975 IEEE international Electron Devices Meeting, Dig. Tech. Papers*, pp. 11-13

[2] V. Tiwari et. al., "Reducing Power in High-performance Microprocessors" , *35th DAC*, June 98.

[3] J. Sparsø, S. Furber (eds). *Principles of Asynchronous Circuit Design: A systems Perspective*. Kluwer Academic Publishers, 2001

[4] T. Verhoeff , "Delay-insensitive codes — an overview". *Distributed Computing*, 3(1):1-8, 1988.

[5] Sutherland, I.E., "Micropipelines", *Communications of the ACM*, 32 (6), June 1989, pp 720-738

[6] Neil H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design — A System Perspective, Second Edition*, Addison-Wesley Publishing Company, 1993

[7] Alain J. Martin, Mika Nystrom and Catherine G. Wong. "Three generations of asynchronous microprocessors". *IEEE Design and Test of Computers, special issue on Clockless VLSI Design*, Nov 2003.

[8] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, A. Takamura, "TITAC: design of a quasi-delay-insensitive microprocessor", *Design and Test of Computers, IEEE* , Volume: 11 , Issue: 2 , Summer 1994

[9] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus. "The Design of an Asynchronous Microprocessor". *ARVLSI: Decennial Caltech Conference on VLSI, ed. C.L. Seitz*, 351-373, MIT Press, 1989.

[10] S.B. Furber, J.D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. "Amulet2e: An asynchronous embedded controller". in *Proc. Async'97*, pages 290-299. IEEE Computer Society Press, 1997.

AN ACM APPLICATION IN BROOM BALANCER

Fei Hao, Fei Xia, Graeme Chester, Alex Yakovlev
School of Electrical, Electronic and Computer Engineering,
Univ. of Newcastle upon Tyne, UK

ABSTRACT

Asynchronous data Communication Mechanism, also known as ACM, is potentially useful in systems with heterogeneous timing as data connectors between processes belonging to different timing domains. In distributed, concurrent and embedded digital systems, there is often a desire to have some temporal decoupling between different parts of a system. ACMs provide a means with which concurrent processes can communicate with one another and yet still avoid synchronization. One of the ACM algorithms was described and successfully inserted into a broom balancer, a feedback control system.

Key words: ACM, Broom balancer, MATLAB

1 INTRODUCTION

Inter-process Asynchrony is inevitable for computation networks in the future, firstly because different and diverse functional elements, especially those connecting to analogue domains, tend to have different timing requirements[1], and secondly because concurrent and distributed system implementations lead to greater asynchrony between components as semiconductor technology advances and the degree of integration increases. The size of computation networks is becoming larger, and the traffic between the processing elements is increasing. Handling the data communications which make up the traffic, therefore, may determine much of the performance and characteristics of such systems.

An ACM is a scheme which manages the transfer of data between two processes not necessarily synchronised for the purpose of data transfer. The provider of data is called the “writer” of the ACM, and the user of data is referred to as its “reader”. The ACM is therefore a data connector linking the two processes, the writer and the reader. The general scheme of these kinds of data communication mechanisms is shown in Figure 1. Most ACM implementations tend to include shared memory, accessible to both writer and reader, for the data being transferred, and control variables, each of which is usually set by one side and read by the other.

ACMs emphasize the asynchrony between the reader and writer processes during data transfer, and are therefore especially suitable for systems of the future where multiple time domains not fully synchronized with one another predominate. ACMs can be classified into four types according to the qualitative properties of the inter-process asynchrony during data transfer as shown in Table 1.

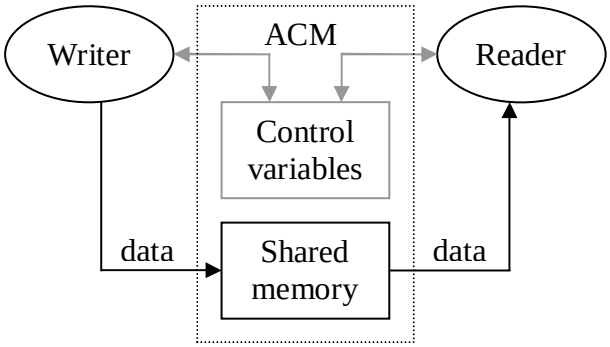


Figure 1 ACM with shared memory and possibly control variables

When discussing ACMs, it is assumed that the data being transferred consists of a stream of items of the same type, and the writer and reader processes are single-thread loops, during each cycle of which a single item of data is transferred to or from the ACM.

Table 1 ACM Classification

	NRR	RR
NOW	Channel	Message
OW	Signal	Pool

In Table 1, which follows the tradition of 2x2 matrix classification schemes found in [1] etc., NRR and RR stand for non-rereading and rereading, while NOW and OW mean non-overwriting and overwriting. Whether rereading is permitted determines if the reader may be held up waiting for new data to arrive from the writer. Whether overwriting is permitted determines if the writer may be held up waiting for previous data in the ACM to be accessed by the reader. Therefore, a Channel without overwriting and rereading provisions may require either process to wait under certain circumstances. A Message may require the writer to wait when previous data items have not been read. A Signal may require the reader to wait when no newer data has been made available by the writer after the previous read. A Pool, however, does not require either side to wait under any circumstances.

The study of ACMs so far, though extensive, has not extended to their direct modelling in application-level tools. Previous proposals for modelling ACMs at a higher level, treating them as components in larger systems, have employed Petri nets. This was suitable for the case where systems containing ACMs can be regarded and analysed as general discrete event digital systems. However, in order to study the effect of including ACMs in such engineering application systems as control systems, especially when analogue parts are present, ACM models need to be integrated into popular application-level tools such as MATLAB.

2 STATEFLOW MODELS OF ACM

MATLAB is a widely used modelling, simulation and analysis tool for engineering application systems in such fields as control, signal processing, large scale hybrid systems with analogue and discrete parts, etc. It also includes a Stateflow facility with which

discrete state-transition subsystems such as ACMs can be modelled. In order to broaden the application space of ACMs, we have developed a method to model and simulate ACMs using MATLAB, based on Stateflow.

2.1 Represent Handshake in the Stateflow Model

The progress of ACM algorithms can be controlled by the writer and reader processes via request-acknowledgement handshakes. A four phase handshake protocol follows this order: sending a request, waiting for the acknowledgement sent from the other side, releasing (resetting) the request, and resetting the acknowledgement from the other side.

This can be modelled in Stateflow as shown in Figure 2. One handshake cycle is represented in the following way: a request is generated in the state entry actions (the "En" statements), which are executed when entering the state; the state itself represents waiting for the acknowledgement in the transition conditions (the conditions in the square brackets, in the case of Figure 2, ACK becoming 1), which lead to the exit from the state (end of waiting) and executions of the transitions; on exiting a state, the requests are released (in the "Ex" statements); and then the acknowledgements can be reset.

The Stateflow models of ACMs were built based on this representation of the handshake protocol.

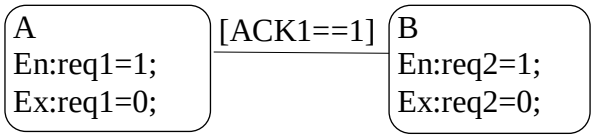


Figure 2 Handshake protocol in Stateflow

2.2 Modelling of a 3-Slot Signal ACM

The Petri net model of a Signal ACM is shown in Figure 3. The description of this model can be found in [2].

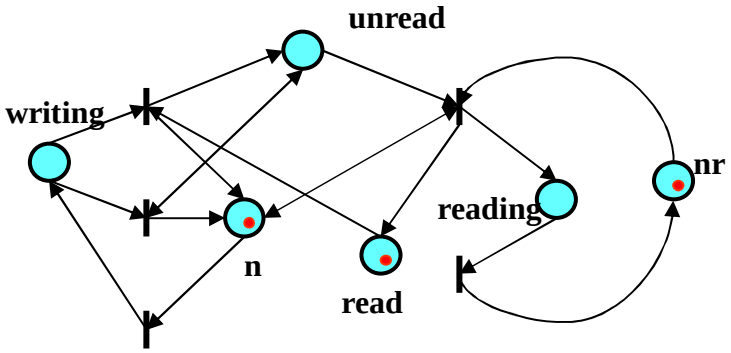


Figure 3 Algorithm of Signal Type ACM

The Petri net model was divided into 3 parts which executed parallel. Each of them was created in the Stateflow (see Figure 4) according the method of converting from a Petri net model [3] and the handshaking representations.

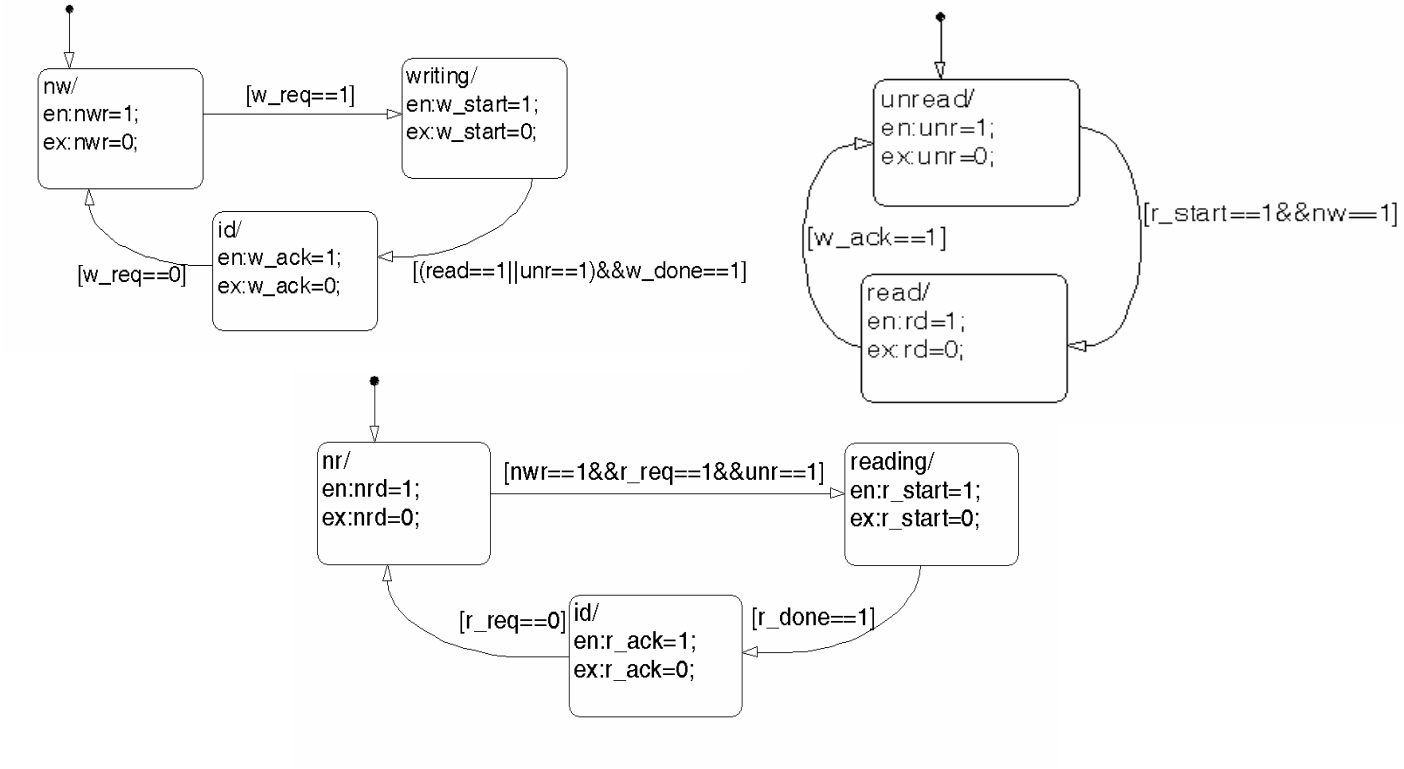


Figure 4 Stateflow Model of the Signal

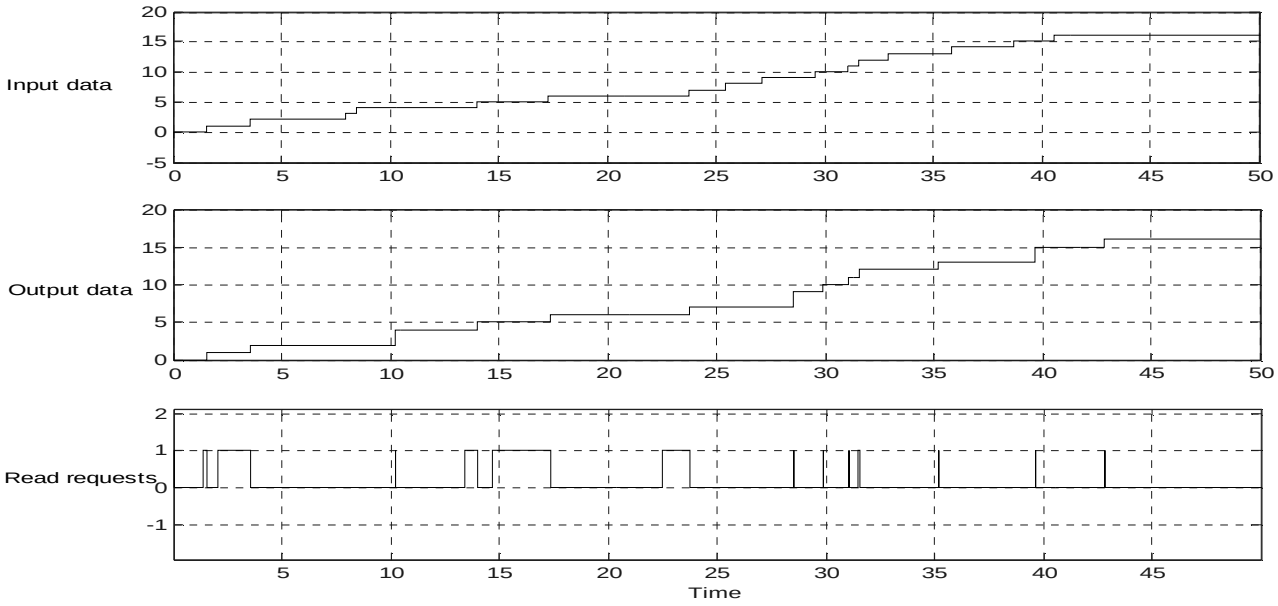


Figure 5 Simulation Results for Signal

This Stateflow model can be plugged into the Simulink environment. The test environment generates write requests, read requests and the input data items. This ACM was simulated

in this environment, with resulting waveforms shown in Figure 5. Reader waiting occurred when read requests came without new data items available, as in the case after the data items 4 and 5 were read (read requests keeping high). While the overwritten happened when new data items came faster than the reader requests, such as data items 8 and 19 were overwritten. The time taken by the reader and writer outside the ACM was controlled by two independent random number generators. An exponential distribution was assumed.

3 BROOM BALANCER

The task of a broom balancer is to keep a rigid broom handle hinged to a trolley and free to fall in a track, in a roughly vertical orientation by moving the trolley horizontally in the track while keeping the trolley within some maximum distance (1 meter in this case) of its starting position (as shown in Figure 6).

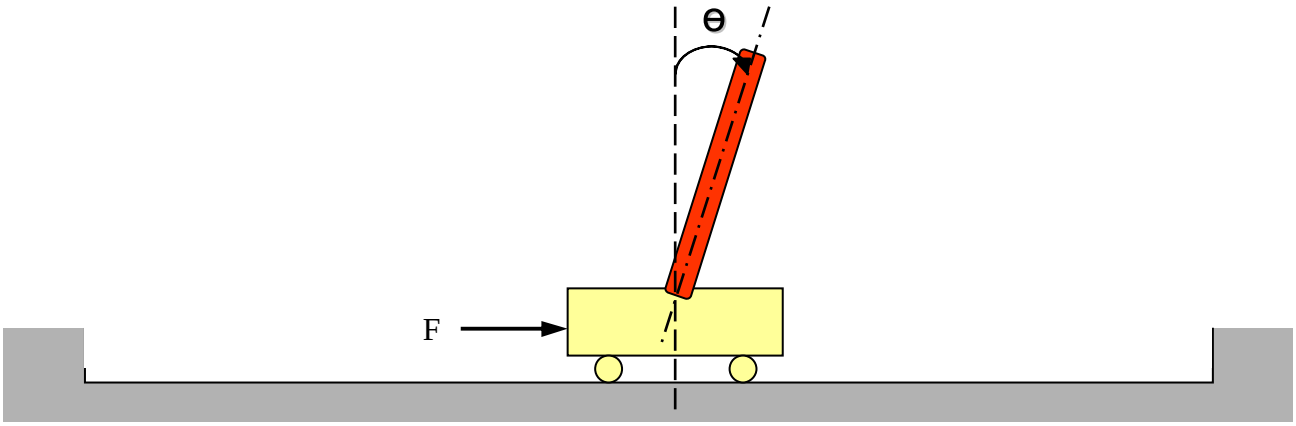


Figure 6 Broom Balancer

The objective is to keep the broom upright and the trolley in the middle of the track. In other words, the angle θ must be kept as close as possible to zero and the position from a fixed reference point must be kept as close as possible to zero.

The broom balancer has been modelled mathematically. [4] derived the formulae, relating the angle θ and the position x to the angular velocity of the broom about its pivot, θ', the angular acceleration of the broom about its pivot, θ'', the velocity of the trolley, x', and the acceleration of the trolley, x''. In [5], the formulae were simplified to linear differential equations:

$$x'' = \frac{F}{M}, \qquad \theta'' = \frac{3g\theta}{4l} - \frac{3F}{4Ml} = \frac{3}{4Ml}(Mg\theta - F)$$

where
F is the applied force (N)
M is the mass of the trolley (1.0kg)
g is the acceleration due to gravity (ms^{-2})
l is the half-length of the broom ($2l$ = 1m).

The transfer function of the broom and trolley can be derived by applying Laplace transforms on the equations:

$$\frac{\Theta}{F} = \frac{\frac{-3}{4Ml}}{s^2 - \frac{3g}{4l}}$$

With substituting the values of *g*, *l* and *M*, it becomes:

$$\frac{\Theta}{F} = \frac{-1.5}{s^2 - 14.7}$$

Figure 7 shows a block diagram of a closed-loop control system of the broom balancer. The broom and trolley block is used to justify θ according to the applied force. The force is produced by a PID controller.

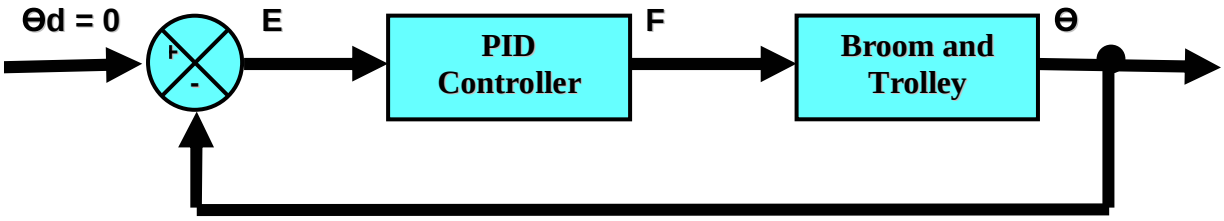


Figure 7 Closed-Loop Control of the Broom Balancer

4 ACM APPLICATION IN BROOM BALANCER

A Signal type ACM can be inserted in the feedback loop. For doing this, the PID controller was modified to a digital controller coupled with two handshakes for its input and output as shown in Figure 8.

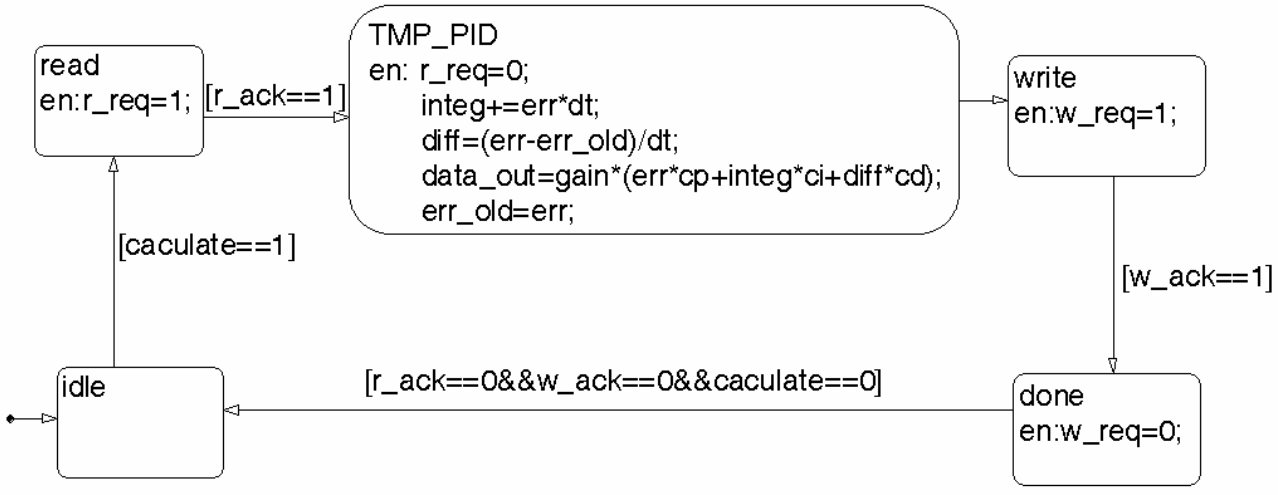


Figure 8 Stateflow Model of PID Controller

When a calculate signal comes to the controller, it sends a read request r_req to the previous block to request the data from the broom and trolley. After it received the data, a read acknowledgement is sent back and an applied voltage (data_out) is worked out. Consequently, a w_req is sent to the following block to request the permission for sending data, and the data will not be sent until a w_ack be received.

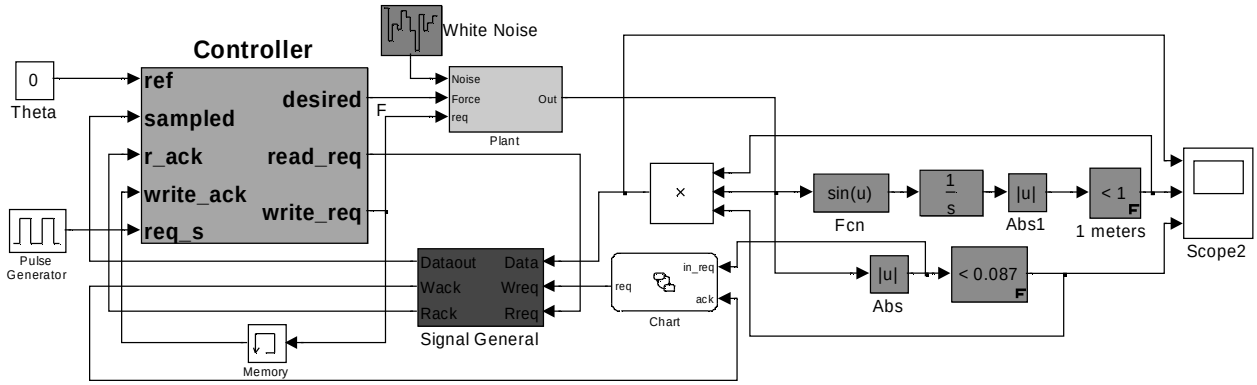


Figure 9 Broom Balancer with ACM Applied in

Figure 9 showed the system with an ACM plugged in. The white noise is the disturbance of the system. The first input of the scope was used to display the output of the system. While the second and third ones were used to monitor if the broom and trolley ran out of the range.

The ACM sampled data from the angular output of the broom and trolley block when the angles were greater than 0.5 degree (done by *chart*), and delivered them to the controller when it requested. The chart generated request pulses to the ACM when the output angles were greater than 0.5 degree. Therefore, the ACM was a non-uniform sampler to the broom and trolley block. The controller ran on a constant speed, it sent read requests in a regular frequency.

The output of the system is the angles between the broom and the vertical. As shown in Figure 10, most of the angles were in the range of -0.015 to 0.015 rads, which was less than 1 degree. Therefore, the broom was kept in a balance state.

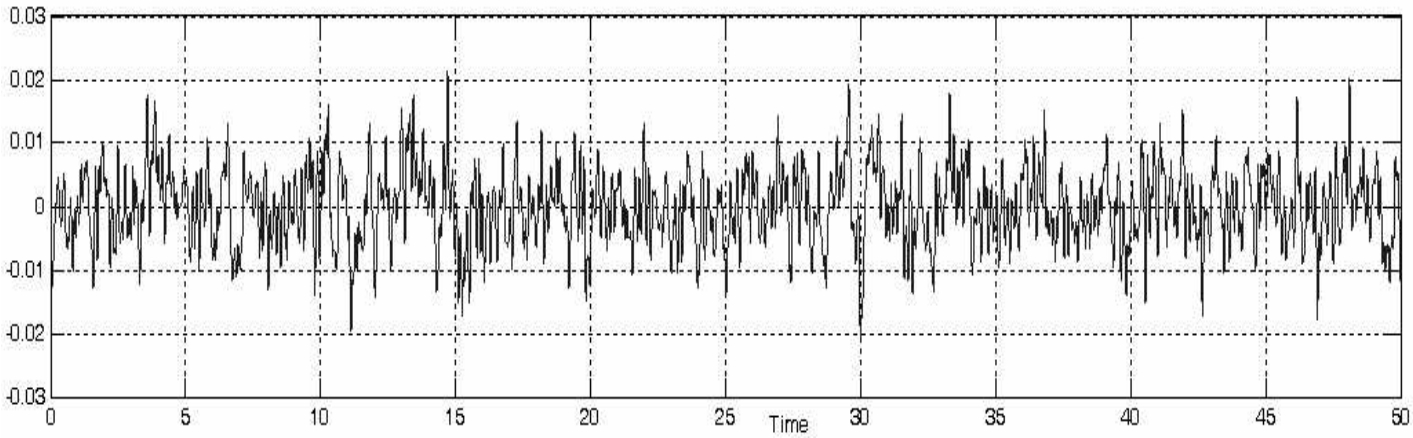


Figure 10 Simulation Result

5 CONCLUSIONS AND FUTURE WORKS

A Signal type ACM was modelled in MATLAB successfully. The results showed that it preformed as expected. As an example of control system, a broom balancer was also built to demonstrate that these kinds of ACM models can be plugged into MATLAB models of control systems for the purpose of simulation.

Future work includes the further development of MATLAB/Simulink models for non-ACM components which would highlight the effect of the various degrees of temporal decoupling ACMs bring to systems.

6 ACKNOWLEDGEMENT

This work is part of the Coherent project (http://async.org.uk/coherent) at the Newcastle University supported by the EPSRC grant (GR/R32666). The authors benefited from extensive discussions with David A. Fraser, H. Simpson and E. Campbell and wish to express our gratitude.

7 REFERENCES

[1] H. R. Simpson, "Protocols for process interaction," *IEE Proc.-Comput. Digit. Tech*, 2003.
[2] A. Yakovlev and F. Xia, "Towards Synthesis of Asynchronous Communication Algorithms," in *Synthesis and control of discrete event systems*. Boston: Kluwer Academic Publishers, 2002, pp. 57-75.
[3] F. Hao and G. Chester, "ACMs in MATLAB," presented at 14th UK Asynchronous Forum, Newcastle upon Tyne, UK, 2003.
[4] A. G. Barto, R. S. Sutton, and C. Anderson, "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*, pp. 834-846, 1983.
[5] E. K. Kappos, D.J. Acarnley, P.P. Jack, A.G., "Design of an integrated circuit controller for brushless DC drives," presented at Fourth International Conference on Power Electronics and Variable-Speed Drives,, London, UK, 1990.

Reducing task jitter in shared-clock embedded systems using CAN

Mouaaz Nahas, Michael J. Pont and Aman Jain

Embedded Systems Laboratory, University of Leicester,
University Road, LEICESTER LE1 7RH, UK

Abstract

We have previously described a technique known as "shared-clock scheduling" for use in distributed embedded systems which must provide high levels of reliability at low cost. In this paper, we consider the levels of jitter that result from the use of a shared-clock approach running over CAN. We describe a modification to the original shared-clock algorithm and demonstrate that this helps to substantially reduce these jitter levels.

Acknowledgements

This project is support by the UK Government (EPSRC-DTA award). Some of the work described in this paper was carried out while Aman Jain was visiting the ESL from IIT Kanpur (India), on a trip funded by the British Council.

1. Introduction

Many modern embedded systems contain more than one processor. For example, a typical passenger car might contain some 40 such devices, controlling brakes, door windows and mirrors, steering, air bags and so forth (Leen *et al.*, 1999).

When implementing multi-processor embedded systems, the Controller Area Network (CAN) bus (Bosch, 1991) is a popular and cost-effective choice (Farsi and Barbosa, 2000; Fredriksson, 1994; Thomesse, 1998; Sevillano *et al.*, 1998). The CAN protocol was introduced by Robert Bosch GmbH in the 1980s (Bosch, 1991). Although originally designed for automotive applications, CAN is now becoming widely used in process control and many other industrial areas (Farsi and Barbosa, 2000; Fredriksson, 1994; Thomesse, 1998; Sevillano *et al.*, 1998; Pazul, 1999; Zuberi and Shin, 1995; Misbahuddin and Al-Holou, 2003). As a consequence of its popularity and widespread use, most modern microcontroller families now include one or more members with on-chip hardware support for this protocol (e.g. Philips, 1996; Siemens, 1997; Infineon, 2000; Philips, 2004).

CAN is usually viewed as an "event-triggered" protocol (Leen and Heffernan, 2002). However, as we have previously demonstrated (Pont, 2001), use of a simple "shared-clock" software architecture allows an effective and highly reliable time-triggered communication strategy to be implemented even with very low-cost microcontrollers.

The shared-clock (S-C) architecture is described in detail elsewhere (Pont, 2001). Briefly, the S-C protocol operates as follows (see Figure 1). On the (single) Master node, a conventional (co-operative or hybrid) scheduler operates and the system is driven by periodic interrupts generated from an on-chip timer. On the Slave nodes, a very similar scheduler operates. However, on the Slaves, no timer is used: instead, the Slave scheduler is driven by interrupts generated through the arrival of messages sent from the Master node[1]. Overall, the S-C architecture is simple and supports number of error-handling mechanisms that make it robust. The network communications follow a TDMA protocol, and the system behaviour is highly predictable.

---

[1] In this paper, we will be concerned with systems based on CAN. However the S-C architecture can be used with a wide range of network protocols, including RS-232 and RS-485.

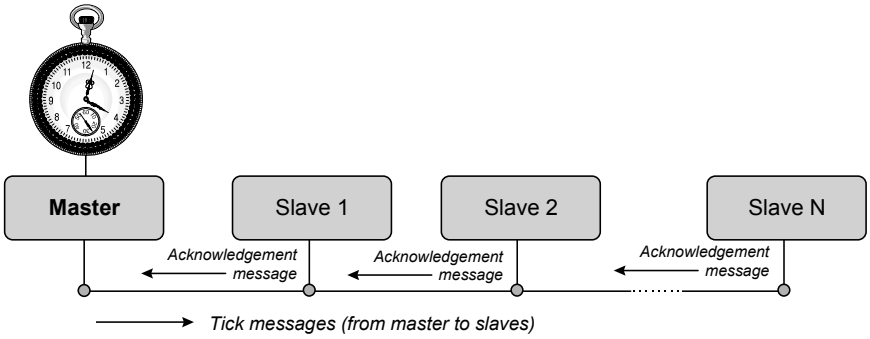


Figure 1: Simple architecture of CAN-based shared-clock (S-C) scheduler.

In any S-C network, there will be jitter in the timing of tasks on the Slave nodes if there is any variation in the time taken to send "tick" messages between the Master and the Slaves. The presence of significant jitter can have a detrimental impact on the performance of many distributed embedded systems. For example, Cottet and David (1999) show that – during data acquisition tasks – jitter rates of 10% or more can introduce errors which are so significant that any subsequent interpretation of the sampled signal may be rendered meaningless. Similarly Jerri (1977) has discussed the serious impact of jitter on applications such as spectrum analysis and filtering.

In this paper, we begin by presenting some empirical data which demonstrates the level of jitter in shared-clock networks implemented as originally described by Pont (2001). We then go on to describe a modification to this shared-clock architecture which can – for all shared-clock systems – reduce the jitter levels.

We begin the paper by describing the methodology used to obtain the results presented here.

2. Experimental methodology

The test platform used in the studies presented in this paper had two nodes: one Master and one Slave. Each node was based on a Phytec board (Phytec, 2004) supporting an Infineon C515C (8-bit) microcontroller, with a 10 MHz crystal oscillator. The C515C is based on the 8051 architecture with additional on-chip support for features such as CAN (Siemens, 1997). In this case, boards were connected with a twisted-pair CAN link.

2.1 S-C CAN scheduler setup

We used a standard implementation of the shared-clock scheduler, taken directly from Pont (2001).

The CAN baudrate was 333.33 kbaud, and 8-byte "Tick" messages were used, with one byte reserved for the Slave ID (see Pont, 2001). The scheduler tick interval was 6 ms.

2.2 Tasks

In the study described in this paper, only one task was scheduled on the Master node. In the Slave, we scheduled between one and ten tasks **in such a way that the impact of jitter would be maximized**.

The parameters of the "add task" function calls were as follows:

```
SCH_Add_Task(Task_A, 0, 10);
SCH_Add_Task(Task_B, 0, 9);
SCH_Add_Task(Task_C, 0, 8);
SCH_Add_Task(Task_D, 0, 7);
SCH_Add_Task(Task_E, 0, 6);
SCH_Add_Task(Task_F, 0, 5);
SCH_Add_Task(Task_G, 0, 4);
SCH_Add_Task(Task_H, 0, 3);
SCH_Add_Task(Task_I, 0, 2);
SCH_Add_Task(Task_J, 0, 1);
```

Listing 1: Modified Dispatcher function in the reduced-jitter scheduler.

2.3 Jitter measurement methodology

In the initial study described in this paper, we wanted to measure the differences between the start time of Task A on the Master (the only task running on this node), and the start time of Task A on the Slave (the first task running on this node).

To make these measurements, a pin on the Master node was set high (for a short period) at the start of the Task_A. Another pin on the Slave (initially high) was set low at the start of Task_A. The signals from these two pins were then AND-ed (using an 74LS08N chip: Texas Instruments, 2004), to give a pulse stream as shown in Figure 2.

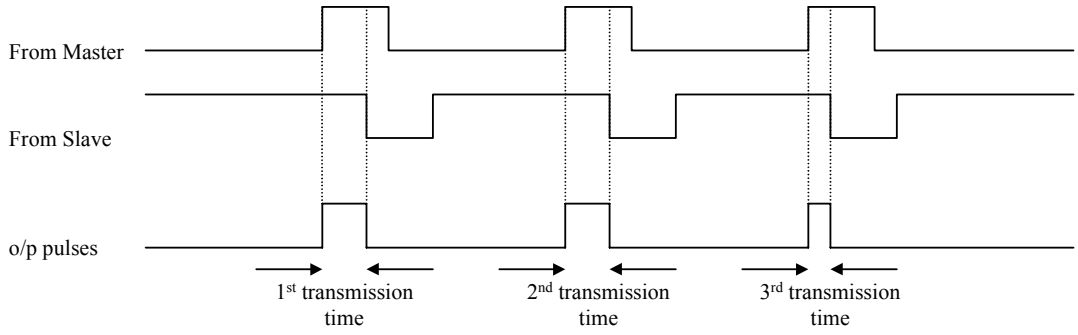


Figure 2: Method used to measure the transmission time in the CAN system.

In all cases, the widths of the resulting pulses was measured using a National Instruments data acquisition card 'NI PCI-6035E' (National Instruments, 2004), used in conjunction with appropriate software LabVIEW 6.1 (LabVIEW, 2004).

In each study, 1000 consecutive pulse widths were measured to give the results presented in this paper.

2.4 Data selection

The nature of the protocol means that, if a CAN message contained a long sequence of identical bits, the clock in the CAN receiver could drift, and there would be the risk of message corruption. To eliminate the possibility of such a scenario, CAN incorporates a "bit stuffing" mechanism. When five identical bits (e.g. five 0s) have been transmitted on the bus, a bit of the opposite polarity is "stuffed" (transmitted) by the sender controller. The receiver controller follows the reverse protocol and removes the stuffed bit after the specified number of identical bits, thereby restoring the original data stream.

Clearly, bit stuffing can have an impact on the message length and, hence – in a shared-clock design – on task jitter.

As we noted above, each CAN message in this study could contain up to 7 bytes of "user defined" data. The choice of data values in this trial needed to be made with care, because of the impact of bit stuffing.

In the studies presented here, the 7 bytes of user data in each message contain "best case", "worst case" or "random" data. The best-case data are 10101010 (for each byte). Such data requires no bit stuffing. The worst-case data occurs when the data are set to be

11111000011110000…… , since this causes the maximum level of bit stuffing (e.g. see Nolte *et al.* 2001). In the case of random data, pseudo-random values were sent in each data byte.

3. Measuring jitter in the original S-C scheduler

Table 1 shows the jitter record from the original S-C architecture.

Table 1: Jitter measurements from the original scheduler

		No. of tasks in the slave									
Expt.	**(µs)**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Seven**	**Eight**	**Nine**	**Ten**
Best Case	**Min**	497.3	521.2	545.3	569.3	593.2	641.9	666	690	738.5	787
	Max	503.7	552.4	600.3	649.5	696.9	746.7	793.7	843.2	889.3	939.4
	Average	500.5	527.3	557.3	584.9	617.1	665.6	701.9	734.2	782.8	831.3
	Max-Min	6.4	31.2	55	80.2	103.7	104.8	127.7	153.2	150.8	152.4
	Std dev	1.7	8.0	13.3	16.0	21.9	22.0	27.8	35.7	35.8	35.8
Worst Case	**Min**	539.5	563.2	587.3	611.2	635.3	683.9	707.8	731.8	780.5	829.1
	Max	545.8	594	642.3	690.4	739.7	787.6	837	885.5	931.7	980.5
	Average	542.5	569.2	599.4	626.9	659.1	707.7	743.9	776.2	824.8	873.4
	Max-Min	6.3	30.8	55	79.2	104.4	103.7	129.2	153.7	151.2	151.4
	Std dev	1.8	8.0	13.3	15.9	21.8	21.9	27.9	35.7	35.7	35.7
Random Data	**Min**	497.6	521.6	545.8	569.4	593.7	641.9	666	690	738.7	787.2
	Max	521.7	566.6	614.1	656.5	702.6	755.6	802.1	847.9	901.3	949.7
	Average	507.5	534.2	564.6	592.0	624.1	672.7	709.1	741.2	789.9	838.4
	Max-Min	24.1	45	68.3	87.1	108.9	113.7	136.1	157.9	162.6	162.5
	Std dev	4.6	8.8	13.7	16.3	22.2	22.3	28.3	35.8	35.9	35.7

By plotting the data for the best-case (Figure 3), worst-case (Figure 4) and random data (Figure 5) we can see more clearly that – in each case - the jitter level increases as further tasks are scheduled to run on the Slave node.

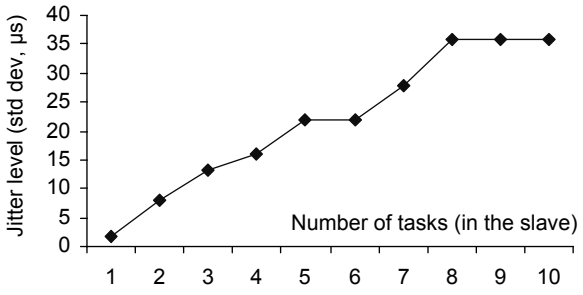


Figure 3: Jitter level in the original scheduler for best-case scenario.

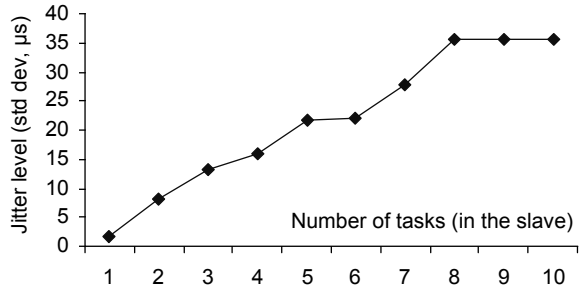


Figure 4: Jitter level in the original scheduler for worst-case scenario.

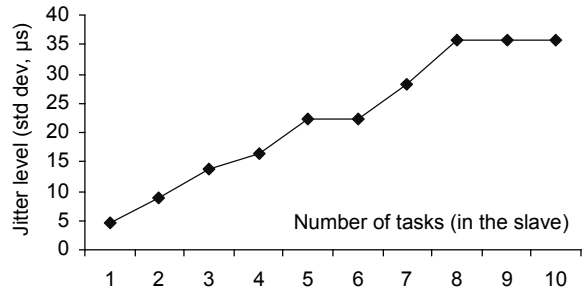


Figure 5: Jitter level in the original scheduler for random-case scenario.

4. Design of the reduced-jitter S-C scheduler

The results of the first study confirm that, in the original S-C implementation, the schedulers introduce jitter.

The underlying cause of this additional jitter is the interrupt behaviour, illustrated schematically in Figure 6.

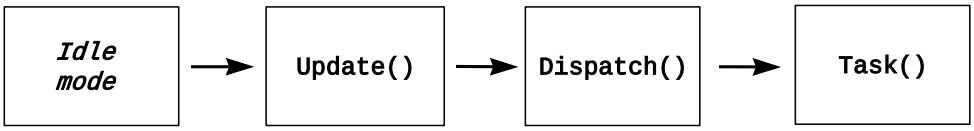


Figure 6: A schematic representation of the sequence of events that cause a task to be launched in a shared-clock scheduler.

For example, consider the behaviour of a Slave node. Before the arrival of a Tick message, the Slave will (in normal circumstances) be in idle mode. The time taken to leave idle mode and begin execution of the Update ISR is essentially fixed. In Update, the scheduler checks to see if any tasks are due to execute. Such checks will vary in duration if there is more than one scheduled task, and it is only after all checks are complete that the dispatcher be called and – if appropriate – one or more tasks executed.

The major cause of jitter in this arrangement is the time taken to check for the next task execution. We can greatly reduce this variation by moving this activity into the dispatcher function. Listing 2 and Listing 3 show one way in which this can be achieved.

```
void SCH_Update(void)
   {
   // Note that an interrupt has occured
   Tick_count_G++;

   // Clear interrupt flag (if necessary)
   }
```

Listing 2: Modified "update" ISR in the reduced-jitter scheduler.

```
void SCH_Dispatch_Tasks(void)
   {
   ...
   // Disable Tick interrupts

   if (Tick_count_G > 0)
      {
      Tick_count_G--;
      Update_required = 1;
     }

   // Re-enable interrupts

   while (Update_required)
      {
      // Go through the task array
      for (Index = 0; Index < SCH_MAX_TASKS; Index++)
         {
         // Check if there is a task at this location
         if (SCH_tasks_G[Index].pTask)
            {
            if (--SCH_tasks_G[Index].Delay == 0)
               {
               // The task is due to run
               (*SCH_tasks_G[Index].pTask)();  // Run the task

               if (SCH_tasks_G[Index].Period != 0)
                  {
                  // Schedule period tasks to run again
                  SCH_tasks_G[Index].Delay =
                                       SCH_tasks_G[Index].Period;
                  }
               else
                  {
                  // Delete one-shot tasks
                  SCH_tasks_G[Index].pTask  = 0;
                  }
               }
            }
         }

      // Disable tick interrupt

     if (Tick_count_G > 0)
        {
         Tick_count_G--;
         Update_required = 1;
       }
      else
         {
         Update_required = 0;
         }

      // Re-enable tick interrupt
      }

   SCH_Go_To_Sleep();
   }
```

Listing 3: Modified Dispatcher function in the reduced-jitter scheduler.

5. Measuring jitter in the modified scheduler

In this study, we repeated the experiments presented in Section 3, this time using the reduced-jitter scheduler described in Section 4. Table 2 presents the results.

Table 2: Jitter measurements from the modified scheduler

		No. of tasks in the slave									
Expt.	**(µs)**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Seven**	**Eight**	**Nine**	**Ten**
Best Case	**Min**	500.8	500.8	500.8	500.8	500.8	500.9	500.8	500.8	500.8	500.8
	Max	507.4	507.5	507.5	507.4	507.5	507.4	507.4	507.4	507.4	507.5
	Average	504.1	504.1	504.1	504.1	504.1	504.1	504.0	504.0	504.1	504.0
	Max-Min	6.6	6.7	6.7	6.6	6.7	6.5	6.6	6.6	6.6	6.7
	Std dev	1.8	1.8	1.8	1.8	1.8	1.8	1.8	1.8	1.8	1.8
Worst Case	**Min**	542.9	542.9	542.9	542.9	542.9	542.9	542.9	542.9	542.9	542.9
	Max	549.4	549.4	549.3	549.4	549.4	549.4	549.4	549.3	549.4	549.3
	Average	546.1	546.1	546.1	546.0	546.1	546.1	546.0	546.1	546.1	546.1
	Max-Min	6.5	6.5	6.4	6.5	6.5	6.5	6.5	6.4	6.5	6.4
	Std dev	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.7
Random Data	**Min**	501	501.1	500.9	500.8	501	501	500.9	500.9	500.9	501.5
	Max	527.4	533.5	527.4	529.7	533.6	528.1	534	530	529.5	532.1
	Average	511.1	511.0	511.0	511.0	511.0	511.0	511.0	511.2	511.0	511.0
	Max-Min	26.4	32.4	26.5	28.9	32.6	27.1	33.1	29.1	28.6	30.6
	Std dev	4.46	4.5	4.6	4.5	4.4	4.4	4.6	4.7	4.5	4.5

Figure 7 compare the results for the original and modified schedulers for the tests using random data.

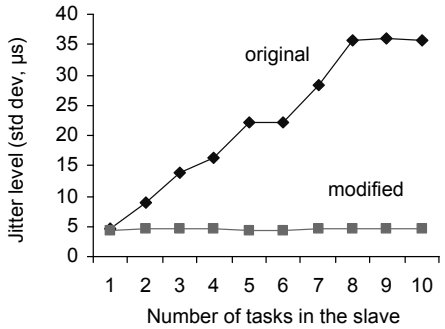


Figure 7: Jitter level in the original and the modified scheduler (random data)

6. Discussion and conclusions

Clearly, the modified scheduler has greatly reduced the jitter in the timings measured here. For both the best-case and worst-case data (see Table 2), the difference between the maximum and minimum transmission times is approximately 6 µs, or +/- 1 bit time for the CAN bus at the baud rate used here. We would expect this figure to be reduced by a factor of approximately 3 at the maximum CAN baud rate.

There remains a difference in timing of the tasks on the Master and Slave of around 30 µs in the situation where random data are sent (Figure 8).

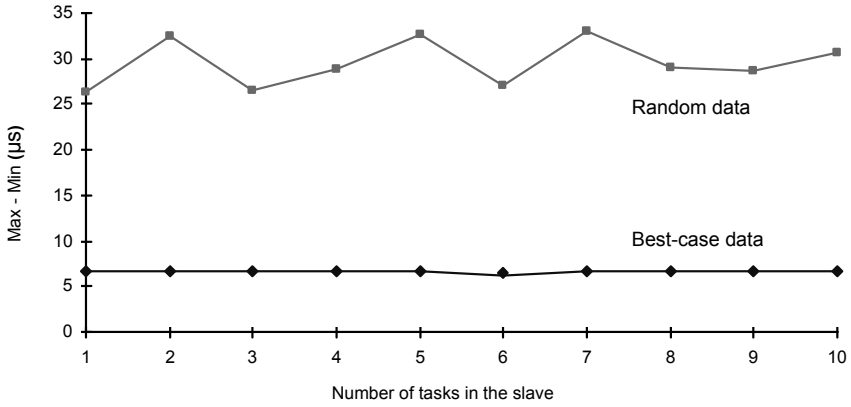


Figure 8: A comparison of the best-case data and random data for the modified scheduler (see text for details)

One explanation for this discrepancy is that it arises as a result of bit stuffing. For example, Nolte *et al.* (2001) have shown that – for a CAN bus at the maximum baud rate – a difference 24 µs can be expected between "best case" and "worst case" message transmissions.

Further studies are required to confirm the source of this remaining jitter, and to devise techniques to reduce its impact.

7. References

Bosch (1991), Robert Bosch GmbH "CAN Specification Version 2.0".

Cottet, F. and David, L. (1999) "A solution to the time jitter removal in deadline based scheduling of real-time applications", 5th IEEE Real-Time Technology and Applications Symposium - WIP, Vancouver, Canada, pp. 33-38

Farsi, M. and Barbosa, M. (2000) "CANopen Implementation, applications to industrial networks", Research Studies Press Ltd, England.

Fredriksson, L.B. (1994) “Controller Area Networks and the protocol CAN for machine control systems”, Mechatronics Vol.4 No.2, pp. 159-192.

Infineon (2000) “C167CR Derivatives 16-Bit Single-Chip Microcontroller”, Infineon Technologies.

Jerri, A.J. (1977) “The Shannon sampling theorem: its various extensions and applications a tutorial review”, Proc. of the IEEE, vol. 65, n° 11, p. 1565-1596.

LabVIEW 6.1 user guide: WWW webpage: http://www.ni.com/pdf/manuals/322884a.pdf [accessed May 2004]

Leen, G. and Heffernan, D. (2002) “TTCAN: a new time-triggered controller area network”, Microprocessors and Microsystems, Volume 26, Issue 2, Pages 77-94

Leen, G.; Heffernan, D. and Dunne, A. (1999) “Digital networks in the automotive vehicle”, Computing and Control, 10 (6): 257 – 66.

Misbahuddin, S.; Al-Holou, N. (2003) “Efficient data communication techniques for controller area network (CAN) protocol”, Computer Systems and Applications, 2003. Book of Abstracts. ACS/IEEE International Conference on, Pages:22.

National Instruments; PCI-6035E data sheet and specs; WWW webpage: http://www.ni.com/pdf/products/us/4daqsc202-204_ETCx2_212_213.pdf [accessed May 2004]

Nolte, T.; Hansson, H.; Norström, C. and Punnekkat, S. (2001) “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. Preliminary DS00713A-page 1 AN713.

Philips (1996) “P8x592 8-bit microcontroller with on-chip CAN, datasheet”, Philips Semiconductor.

Philips (2004) “LPC2119/2129/2194/2292/2294 microcontrollers user manual”, Philips Semiconductor.

Phytec: http://www.phytec.com/sbc/8bit/mm515c.htm [accessed May 2004]

Pont, M.J. (2001) "Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers", ACM Press / Addison-Wesley. ISBN: 0-201-331381.

Sevillano J L, Pascual A, Jiménez G and Civit-Balcells A (1998) “Analysis of channel utilization for controller area networks” Computer Communications, Volume 21, Issue 16, Pages 1446-1451

Siemens (1997) “C515C 8-bit CMOS microcontroller, user’s manual”, Siemens.

Texas Instruments: 74LS08 Datasheet, WWW webpage: http://www.cs.amherst.edu/~sfkaplan/courses/spring-2002/cs14/74LS08-datasheet.pdf [accessed May 2004]

Thomesse, J. P. (1998) “A review of the fieldbuses” Annual Reviews in Control, Volume 22, Pages 35-45

Zuberi, K. M. and Shin, K. G. (1995) "Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications", in Proc. Real-Time Technology and Applications Symposium, pp. 240-249.