μSystems Research Group

School of Electrical and Electronic Engineering

Maximising Microprocessor Reliability through Game Theory and Heuristics

J. Docherty

Technical Report Series

NCL-EEE-MICRO-TR-2014-187

March 2014

Contact: james.docherty@ncl.ac.uk

NCL-EEE-MICRO-TR-2014-187

μSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Maximising Microprocessor Reliability through Game Theory and Heuristics

J. Docherty

March 2014

Maximising Microprocessor Reliability Through Game Theory and Heuristics

A Thesis submitted for the degree of Doctor of Philosophy

James Docherty
School of Electrical and Electronic Engineering

February 6, 2014

Abstract

Embedded Systems are becoming ever more pervasive in our society, with most routine daily tasks now involving their use in some form and the market predicted to be worth USD 220 billion, a rise of 300%, by 2018. Consumers expect more functionality with each design iteration, but for no detriment in perceived performance. These devices can range from simple low-cost chips to expensive and complex systems and are a major cost driver in the equipment design phase. For more than 35 years, designers have kept pace with Moore's Law, but as device size approaches the atomic limit, layouts are becoming so complicated that current scheduling techniques are also reaching their limit, meaning that more resource must be reserved to manage and deliver reliable operation. With the advent of many-core systems and further sources of unpredictability such as changeable power supplies and energy harvesting, this reservation of capability may become so large that systems will not be operating at their peak efficiency.

These complex systems can be controlled through many techniques, with jobs scheduled either offline prior to execution beginning or online at each time or event change. Increased processing power and job types means that current online scheduling methods that employ exhaustive search techniques will not be suitable to define schedules for such enigmatic task lists and that new techniques using statistic-based methods must be investigated to preserve Quality of Service.

A new paradigm of scheduling through complex heuristics is one way to administer these next levels of processor effectively and allow the use of more simple devices in complex systems; thus reducing unit cost while retaining reliability, a key goal identified by the International Technology Roadmap for Semiconductors for Embedded Systems in Critical Environments. These changes would be beneficial in terms of cost reduction and system flexibility within the next generation of device. This thesis investigates the use of heuristics and statistical methods in the operation of real-time systems, with the feasibility of Game Theory and Statistical Process Control for the successful supervision of high-load and critical jobs investigated. Heuristics are identified as an effective method of controlling complex real-time issues, with two-person non-cooperative games delivering Nash-optimal solutions where these exist. The simplified algorithms for creating and solving Game Theory events allow for its use within small embedded RISC devices and an increase in reliability for systems operating at the apex of their limits. Within this Thesis, Heuristic and Game Theoretic algorithms for a variety of real-time scenarios are postulated, investigated, refined and tested against existing schedule types; initially through MATLAB simulation before testing on an ARM Cortex M3 architecture functioning as a simplified automotive Electronic Control Unit.

”I don’t know anything, but I do know that everything is interesting if you go into it deeply enough.” *Richard P. Feynman*

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

- ABS — Anti-lock Braking
- ANOVA — Analysis of Variance
- ASIL — Automotive Safety Integrity Level
- BAM — Broadcast Announce Message
- CAFE — Corporate Average Fuel Economy
- CAN — Controller Area Network
- Cpk — Capability Index
- DMA — Direct Memory Access
- DMAIC — Define, Measure, Analyse, Improve, Control
- DoE — Design of Experiments
- ECU — Electronic Control Unit
- EDF — Earliest Deadline First
- ELF — Early Life Failures
- EMC — Electromagnetic Compliance
- EMI — Electromagnetic Interference
- EOS — Electrical Overstress
- ESD — Electrostatic Discharge
- ETS — European Test Symposium
- FCFS — First Come First Served
- FET — Field Effect Transistor
- FITS — Failures per Trillion Hours
- FMEA — Failure Mode Effects Analysis
- GLM — General Linear Model
- HALT — Highly Accelerated Life Testing

- HAPM — Harvesting Aware Power Management
- HIL — Hardware in the Loop
- IC — Integrated Circuit
- I-MR — Individual - Moving Range
- ITRS — International Technology Roadmap for Semiconductors
- KPIV — Key Process Input Variables
- LCL — Lower Control Limit
- LSA — Lazy Scheduling Algorithm
- MNEDC — Modified New European Driving Cycle
- MTBF — Mean Time Between Failures
- MTTF — Mean Time to Failure
- MTTR — Mean Time to Repair
- NOx — Oxides of Nitrogen
- OTE — One Time Engineering
- PGN — Parameter Group Number
- PM10 — Particulate Matter below 10μm
- PV — Photovoltaic
- PWM — Pulse Width Modulation
- QoS — Quality of Service
- RDR — Remote Data Request
- RFID — Radio Frequency Identification
- RM — Rate Monotonic
- RR — Round Robin
- RTOS — Real-Time Operating System
- SJF — Shortest Job First
- SPC — Statistical Process Control
- TDC — Top Dead Center
- TMR — Triple Modular Redundancy
- UCL — Upper Control Limit
- VDM — Vienna Development Method
- WCET — Worst Case Execution Time

Preface

The passion of men for equality is ardent, insatiable, eternal, invincible.
Alexis de Tocqueville

This thesis summarises the work undertaken over the last three years in the School of Electrical and Electronic Engineering at Newcastle University. The initial kernel of this research came while reading Ken Binmore's excellent book "Game Theory: A Very Short Introduction", which first introduced me to the idea that events previously thought of as non-deterministic could be modelled and controlled mathematically. This combined with ideas raised during Six Sigma training during my time in industry that any Gaussian process can be predicted using simple tools, and theories introduced during my MSc to develop a proposal that quickly grew, being built upon with the outcome contained within these pages.

Acknowledgements

I would like to thank my supervisors; Professor Alex Yakovlev and Dr. Alex Bystrov for their assistance throughout this thesis. They have often been beacons that have prevented items of work running onto the rocks. Thanks also go to Dr. Albert Koelmans and Professor Alan Burns for agreeing to be my internal and external examiners respectively.

I would also like to thank Mr. Graeme Coapes and Mr. Johnson Fernandes for their helpful discussions, especially at points where all looked hopeless. Also thanks go to Dr. Panagiotis Asimakopoulos and Dr. Robin Emery for introducing me to the LaTeX language, which has allowed the professional creation of this thesis.

For their support over many years, special thanks go to my parents Carole and Brian and to my wife Emma for giving me the courage to take on this task and for all the tireless support and help she has given over the last three years.

This work was funded by a Doctoral Teaching Account from the EPSRC.

Bibliography

Items from this work have appeared in the following publications:

- Journal Papers

  1. J. Docherty, A. Bystrov, A. Yakovlev; "Testing of a Real-Time Heuristic Scheduler with Automotive Benchmarks"; International Journal of Simulation: Systems, Science and Technology; Volume 14

- Conference Papers

  1. J. Docherty, A. Bystrov, A. Yakovlev; "Simulation and Validation of a Heuristic Scheduling Algorithm for Multicore Systems", SIMUL 2013; 27th Oct - 1st Nov 2013;
  2. J. Docherty, A. Bystrov, A. Yakovlev; "Testing of a Real-Time Heuristic Scheduler with Automotive Benchmarks"; UKSIM2013; 10-12 April 2013; doi: 10.1109/UKSim.2013.98
  3. J. Docherty, A. Bystrov, A. Yakovlev; "Identification of Key Energy Harvesting Parameters Through Monte Carlo Simulations"; UKSIM2012; 28-30 March 2012; doi: 10.1109/UKSim.2012.73

- Workshop Papers

  1. J. Docherty, A. Bystrov, A. Yakovlev; "The Use of Game Theory Within Automotive Job Scheduling"; Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN) at ETS13; May 30-31 2013
  2. J. Docherty, A. Bystrov, A. Yakovlev; "Using Game Theory for Managing Power and Reliability in a Circuit"; Workshop on Low Power Design Impact on Test and Reliability (LPonTR) at ETS11; May 26-27 2011
  3. J. Docherty, A. Yakovlev; "Game Theoretic Power Management Techniques for Real-Time Scheduling"; Workshop on Micro Power Management for Macro Systems on Chip at DATE 2011; March 18 2011

- Technical Reports and Memos

  1. J. Docherty, A. Yakovlev; "Game Theoretic Power Management Techniques for Real-Time Scheduling"; Technical Memo Series NCL-EECE-MSD-MEMO-2011-002; January 2011
  2. A. Yakovlev; "Energy Modulated Computing"; Technical Report Series NCL-EECE-MSD-TR-2010-167; December 2010

Chapter 1

Introduction

An investment in knowledge pays the best interest.
Benjamin Franklin

1.1 Motivation

Our dependency and use of electronic devices continues to grow at a rapid rate. While the market for home computers is flat, the market for embedded systems passed seven billion units in 2012 and is predicted to reach nine billion by 2015 and be worth over USD 220 billion by 2018 [63]. This has led to many chip manufacturers, including Intel, to focus on low power, high performance devices for use in this expanding market [39]. However, each new design adds to the burden of expectation placed upon designers and manufacturers. Mobile devices are now expected to be capable of streaming high definition video, coping with high quality graphics rendering and matching consumers expected quality of service, often while running on battery power. Other embedded devices have successfully pervaded our lives with less awareness, controlling many aspects of our lives from ensuring toast is always perfectly golden brown to sensing an impact within a car that requires the air-bags to deploy and reduce the risk of serious injury.

To allow the achievement of these eclectic and varying goals, microprocessors have had to increase in the processing power and efficiency. With designers pushing to obey Moore's Law, that the number of transistors on a chip will double every 18 months [68], turning this law into a design goal and developing new design methodologies and manufacturing techniques to ensure each generation of chip stays near target. Initially, more parts were added by simply reducing the size of each transistor. However as channel length went below $1\mu m$, this quick fix became more difficult. Currently sizes of $< 90nm$ are common, meaning exotic materials such as strained silicon and high-k dielectrics have been adopted to retain the increase in performance, though these add cost to the manufacture of each device, which is passed on to the end customer. Therefore, due to these changes, it has been suggested that the classic Von Neumann architecture of computers, which can be seen in Figure 1.1, is no longer a feasible layout for microprocessors [114]. The increase in parallelism and the need for resources to be shared between differing job types and roles mean that research into the next

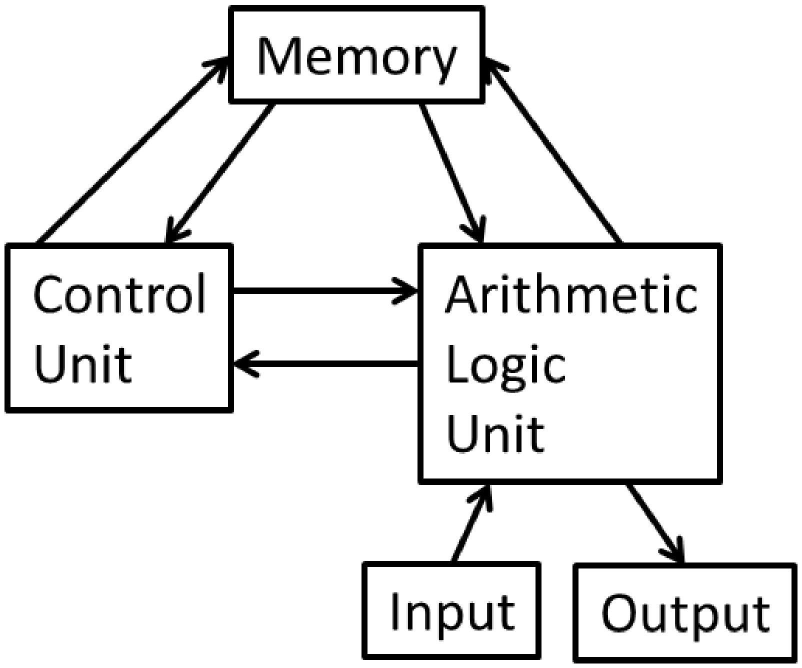


Figure 1.1: The Von Neumann Architecture

generation of systems must look beyond this model, which has been effective for more than 70 years, and instead to more advanced concepts. Designers have looked beyond the reduction of transistor feature size to increase device count and are now using multi-core systems to achieve performance targets.

Multi-core systems are of two common types: Heterogeneous, where cores of different types are mounted on the same processor, and homogeneous environments where the same structure is repeated many times to give a parallel system on a single chip. Both these have been widely adopted within the embedded community, with designers now increasing the number of cores in a newly defined corollary of Moore's Law that the number of cores will double every 18 months. Currently, this can be seen to be the case, as Figure 1.2 shows.

This increase in processing power comes at a price, which is energy consumption. Smartphone users are all too aware of the need to continually recharge their devices, with lifetimes measured in hours rather than days. For most consumer devices, the issue of running out of charge is a mild inconvenience, with the user simply having to put up with being deviceless until they can plug in to a mains supply. However, within certain applications such as medical devices a loss of functionality is not acceptable. Therefore, methods of both reducing consumption and adding energy back to the battery from ambient conditions are active areas of research, ensuring reliability for these critical devices.

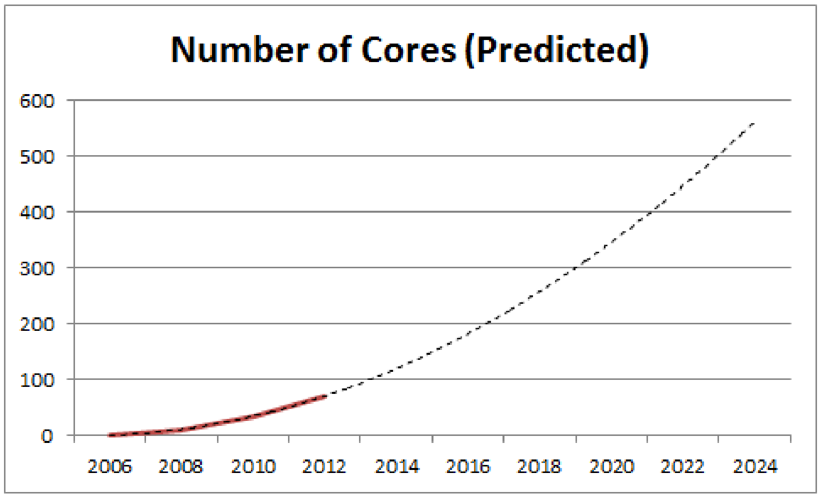


Figure 1.2: Number of Cores per chip with Predicted Curve [13]

1.2 Research Goals and Thesis Contribution

This thesis is concerned with the retention of quality of service for consumers. The work has concentrated on maximising executable time for systems running from battery or energy harvesting devices, along with the execution of hard real-time jobs in a mixed system. Energy Management through Statistical Process control is presented, which gives a significant increase in runtime within simulation for both a generic system and one based upon an established architecture.

Also presented is the use of heuristics based on known rules and game theory for the execution of real-time jobs within a kernel. This takes the concept of a system self-optimising and compares it to established schedulers using industry-defined benchmarks.

Finally, this thesis demonstrates the use of a simple processor to execute advanced tasks from an automotive benchmark and the feasibility of the developed heuristic scheduler within hardware. An illustration of the flow, showing concurrent development of concepts and theories can be seen in Figure 1.3.

The above goals are significant to minimise the trade-off between performance and battery consumption for mobile applications and to increase the longevity of established microprocessor architectures. If an increase in performance or lifetime can be delivered for mature products through establishment of a new management paradigm, it will allow system designers at all abstraction levels to deliver products that meet consumer expectations of performance, while using recognised and proven design tools and methods. This development would be significant as one-time engineering (OTE) costs for a new product would be reduced or negated, driving down product cost and time to market.

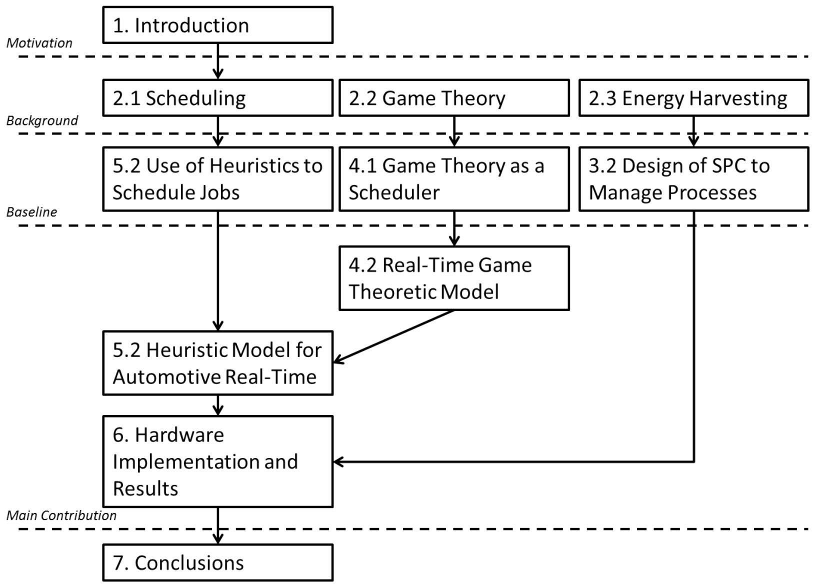

1. Introduction
Motivation
2.1 Scheduling
2.2 Game Theory
2.3 Energy Harvesting
Background
5.2 Use of Heuristics to Schedule Jobs
4.1 Game Theory as a Scheduler
3.2 Design of SPC to Manage Processes
Baseline
4.2 Real-Time Game Theoretic Model
5.2 Heuristic Model for Automotive Real-Time
6. Hardware Implementation and Results
Main Contribution
7. Conclusions

Figure 1.3: Layout of Thesis

1.3 Thesis Structure

This Thesis is structured as seven chapters:

1. **Chapter 1** outlines the motivation behind this work and the contribution of this thesis.

2. **Chapter 2** summarises the theories behind Real-Time Systems, Game Theory, Statistical Process Control and some of the tools used. Definitions of Nash Equilibrium, Cpk, Six Sigma and other key items are made.

3. **Chapter 3** studies the use of batteries and other energy storage devices and methods of maximising reliability. An overview of energy harvesters and storage devices is presented, along with a summary of existing power management techniques. The need for new energy control is shown, along with a method of performing this using Statistical Process Control.

4. **Chapter 4** presents the use of Heuristics to manage jobs and retain reliability within an embedded system. The use of a simple heuristic is demonstrated through MATLAB simulation, with further discussion and demonstration of the feasibility of Game Theory within a single-core environment.

5. **Chapter 5** continues the work from Chapter 4, testing a heuristic scheduler within an Automotive Real-Time environment simulated in MATLAB. The designed scheduler is compared to two other common algorithms and the benefit to system designers demonstrated.

6. **Chapter 6** shows the implementation of the scheduler within Chapter 5 on an embedded controller, along with further testing to assess reliability against the chosen benchmarks.

7. **Chapter 7** summarises the key results and achievements, while also suggesting areas for future work and research.

1.4 Research Questions

This thesis aims to examine the link between the use of statistical based techniques, such as game theory and heuristic scheduling, and the possibility of increasing system availability by reducing missed processing jobs and energy consumed during execution. It is hypothesised that a system will lose availability as energy within the system reduces. However, a well-designed energy management system will provide a saving greater than the extra overhead the additional circuitry and processing will consume. Therefore, a simple, well designed algorithm should significantly increase the lifetime and corresponding availability of the system for useful work. Furthermore, by changing from rigid scheduling paradigms to a technique based on environment awareness, critical jobs can have their importance preserved beyond the limits usually imposed by scheduler rules. Therefore, it should be possible to observe a reduction in missed jobs and and increase in system availability through a combined use of these scheduling and energy saving techniques.

Chapter 2

Background

You look at where you're going and where you are and it never makes sense, but then you look back at where you've been and a pattern seems to emerge.
Robert Pirsig, Zen and the Art of Motorcycle Maintenance

2.1 Scheduling

2.1.1 Introduction

Since the earliest computer systems were developed in the post-war period, the question of how best to share tasks on a processor has been a topic of constant investigation. As systems increased in complexity and operating systems took the place of manual program loading; execution order and scheduling became automatic. As the complexity grew, different methods of scheduling these jobs became commonplace. A selection of these can be seen in Table 2.2.

First Come First Served (FCFS)

In everyday life, the most common form of queue encountered is this one. Patrons join at the back and continue through the queue until they reach the head. Once the next clerk becomes available, they will be served. As Table 2.2 shows, this is a simple system to implement but rapidly degrades in performance if the number of clerks is not high enough or jobs take an unexpectedly long time to service. More on FCFS can be seen in Chapter 3.

Shortest Job First (SJF)

In contrast to FCFS, SJF looks through the queue and services the job with the shortest execution time. This maximises throughput and allows a high level of utilisation. However this comes at the expense of holding back longer and possibly more valuable jobs, which can cause a reduction in the quality of service (QoS) and reduce patron satisfaction. SJF is a common method used in production lines to improve takt time (the time a unit may remain on station to keep production flowing) and ensure a smooth supply of jobs.

Figure 2.1: A literal job queue for the EDSAC II, Cambridge University 1960 [118]

Priority Queue

Commonly seen in areas such as Airports, Priority Queue is part of the Multilevel and Cooperative schedulers in Table 2.1. Jobs or Customers are placed in queues according to the priority assigned by the scheduler and queues are serviced according to their priority level. This system has been successful for airlines [83], satisfying premium customers while not overly degrading non-priority patrons. However, if a surge of priority jobs occurred, this could lead to unacceptable waiting for both types of customer and a rapid reduction in QoS. Therefore the number of queuing levels, clerks servicing jobs and ratio of priority to non-priority jobs must be carefully considered to prevent this happening.

Round Robin

Used successfully as the IEEE802.5 token ring network, Round Robin (RR) scheduling is a totally fair method of allocating jobs to devices. Each job receives exactly the same time and may only use the device if it holds the token. Once its time has expired, the token must visit all other jobs in the queue before returning. Since token time is usually short, short jobs receive the advantage of SJF, while long jobs also have the chance to partially complete. RR is significantly more complicated than any of the methods above, with the ability to add priority further compounding this intricacy. Since all jobs receive the same time holding the token, very short jobs can be complete before passing it on; this reduces processor utilisation and therefore efficiency significantly.

Multilevel

Used in the Linux System, this scheduling paradigm works as a mixture of Priority Queue and FCFS. Several queues exist and the system will service the highest queue that has jobs present. The primary difference between simple Priority Queue and this system is jobs can jump up queues to assist their servicing if a set period of time has elapsed. This gives good and fair time to processes, while preserving Quality of Service for critical tasks.

2.1.2 Real-Time Systems

A real-time system has the added difficulty of not only generating correct results, but having these dependant on the physical instant they are produced [55]. The system classically acts as the intermediary between an operator and a controlled object; interpreting signals and providing instructions through an interface. A well-designed real-time system should deliver timeliness (correct output at correct time), consider peak load to prevent missing jobs, predict scheduling, tolerate and recover from faults and be easy to modify and maintain [18].

Real-time systems differ from standard software models through the addition of a deadline to system considerations [54]. A job within a real time system has five main parameters:

- Arrival Time A_i
- Start Time S_i
- Computation Time C_i
- Finish Time F_i
- Deadline D_i

These can be seen graphically in Figure 2.2. A job can be considered late if $F_i - D_i > 0$ with the slack in a process calculated as $X_i = D_i - A_i - C_i$.

When multiple jobs are run on a system, precedence and processor utilisation must be taken into account. Processor Utilisation $\mu = \sum_{i=1}^{n} \frac{C_i}{T_i}$ where T_i is total processor time available. $\mu < 1$ will allow all tasks to schedule under certain schedulers.

Real-time Job Types

Figure 2.3 graphically demonstrates the four common types of job within a real-time system. Non real-time jobs must be considered as these may be within some processes. Since these have no deadline, their value ($V(fi)$) cannot be considered to reduce at any point. Soft jobs lose value after they pass their deadline, but continue to have some use until a time t. Therefore, these jobs can tolerate some late execution and even allow a $\mu > 1$ to occur, provided no

Table 2.1: Common Operating Systems and Their Schedulers [107]

OS	Pre-emption	Scheduling Algorithm
Windows 3.1x	No	Cooperative
Windows 95/98	Half	Cooperative/Pre-emptive
Windows NT/7	Yes	Multilevel Feedback
iOS9	Some	Cooperative
iOSX	Yes	Multilevel Feedback
Linux 2.6.23	Yes	Completely Fair
FreeBSD	Yes	Multilevel Feedback

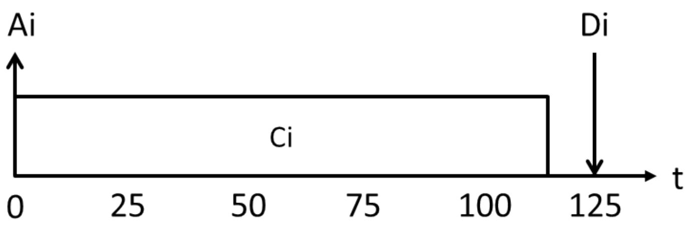


Figure 2.2: Real-time Job Example, showing Activation, Computation and Deadline

Table 2.2: Queuing Types and Their parameters [107]

Algorithm	**Overhead**	**Throughput**	**Turnaround**	**Response**
First Come First Serve	Low	Low	High	Low
Shortest Job First	Medium	High	Medium	Medium
Priority Queue	Medium	Low	High	High
Round Robin	High	Medium	Medium	High
Multilevel	High	High	Medium	Low

firm or hard jobs are missed. Firm jobs have a high level of importance, so receive a value of 0 if they are late. This value means the information within the task can be perceived to have no worth, though it may finish executing. Hard real-time jobs are the most critical to control. A miss of these could lead to hazardous conditions outside of the system that could cause a dangerous outcome. Misses of hard real-time jobs should therefore be avoided at all costs, either through careful scheduler choice or system design.

Scheduling of Real-Time Tasks

Schedulers are of two main types: Time-triggered and event-triggered. Event triggered will activate on arrival of a new task, assess the jobs and alter the schedule accordingly. Time-triggered activates periodically through a real-time clock within the system. Since real-time clocks are now common within modern embedded systems, time-triggered schedulers are more common due to their predictable temporal behaviour, improved scaling and verification testing over event-triggered. Though in a system with resource constraints such as a battery-powered device, event-triggered could be considered superior [54]. The ideal scheduler and triggering method vary depending on items such as job precedence, ratio of periodic to aperiodic tasks, number of cores available, etc. These must be considered by the system designer during the concept phase of the design to ensure high levels of both processor utilisation and system reliability. While many scheduling algorithms exist for real-time systems, the two most commonly used are Earliest Deadline First (EDF) and Rate Monotonic (RM), which will be discussed further in the following sections.

Earliest Deadline First EDF is defined in [18] as follows, *"For n independent tasks with arbitrary arrival times, an algorithm that at any instant releases the task with the earliest absolute deadline amongst all ready tasks."*

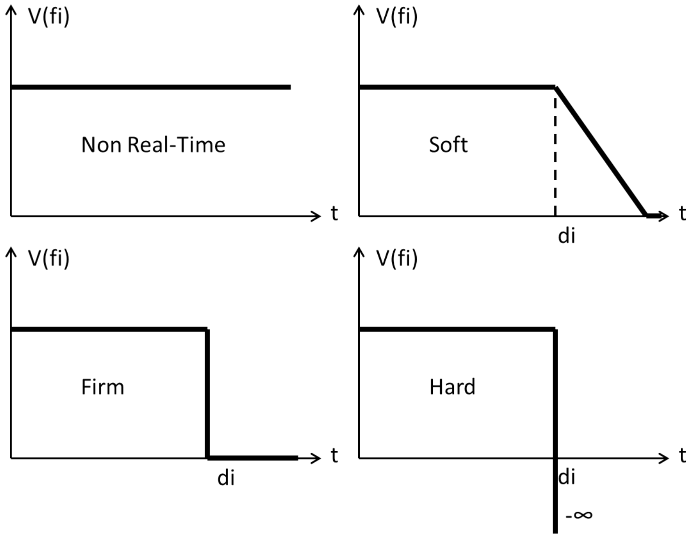


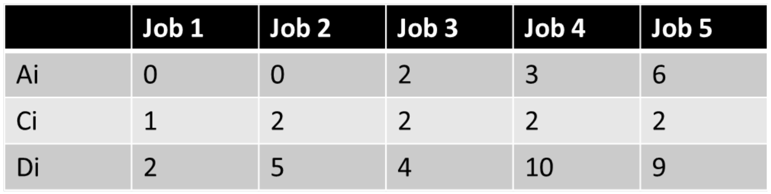
Figure 2.3: Real-time Job Types [18]

This algorithm minimises lateness and can be deemed optimal as it will find a feasible schedule, should one exist. EDF can be pre-emptive, where jobs may be stopped mid execution to allow ones with earlier deadlines to execute. An example of this can be seen in Figure 2.4.

Rate Monotonic RM is defined by [18] as, *"a simple rule that assigns priorities to tasks according to their request rates. Tasks with higher request rates (shorter periods) will have higher priorities."*

These priorities are assigned before execution and are static, giving an intrinsically pre-emptive system where a currently executing task will be pre-empted by one with a shorter period. It is optimal for schedules with fixed priority assignments, though with lower processor utilisation than EDF [18]. However, this means a critical long task may be blocked by Rate Monotonic, so careful consideration must be made when using this scheduler, though [82] comments that Rate Monotonic is optimal for independent tasks on a uniprocessor system and can be improved by a priority altering system such as that in [60].

In addition to EDF and RM, several other algorithms have been developed to control systems and manage unpredictability. A design of note is the Lazy Scheduling Algorithm (LSA) [70]; this has been used in changeable energy environments to determine optimum execution points for jobs by running a job as close to its deadline as possible, outperforming EDF and allowing a reduction in energy storage requirements. For this work, running a job as late as possible is thought of as optimal, a suggestion also noted in [26], where a job gains priority when laxity is zero.



	Job 1	Job 2	Job 3	Job 4	Job 5
Ai	0	0	2	3	6
Ci	1	2	2	2	2
Di	2	5	4	10	9

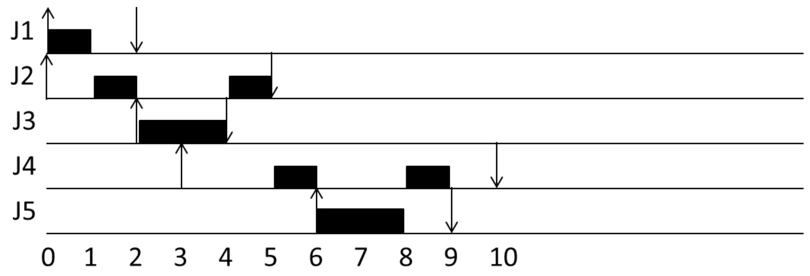


Figure 2.4: Graphical Depiction of a schedule created in EDF with pre-emption

Dynamic Real-Time Scheduling

Fixed schedules or those that only obey basic rules, while suitable for basic systems and theoretical examples, reduce in efficiency for complicated devices where a mixture of real-time and non-real-time tasks are present. To overcome this, work developing from suggestions made in [17] to reserve CPU runtime for Hard Real-Time tasks through a variety of methods has taken place. Many of these techniques look at using established algorithms such as EDF and add flexibility based on heuristics to understand whether resources should be reserved or job scheduling order altered [59] [22]. For systems running mixed tasks and with a high element of aperiodic jobs, it was found that the use of EDF with a preserved area to account for this and allow a secondary temporal deadline to exist gave improvement in performance. This preserves the EDF scheduling methods and allows mixed tasks to execute, giving a hybrid scheduler — similar to the SPRING algorithm in that the scheduler changes to give ideal responses for the situation at hand. [2].

Heuristics have been used in traffic management [19] and energy management [61] within integrated circuits and have been noted as the main feasible solution for high-complexity situations; due to their ability to react iteratively as the environment changes [98]. These works consider all aspects of the real-time taxonomy discussed in Chapter 4 and use approaches such as linear programming and task mapping to resolve a best case solution. In these examples, the algorithms are relatively short (approximately 30 lines) and are often simulated in MATLAB due to its ability to handle large data sets. Methods can use explicit rules to find optimal strategies [76] or more abstract techniques such as Nash Equilibrium discussed in section 2.2 [3]. Heuristics allow for schedules that are clearly infeasible to be ignored through initial grouping, which reduces computation time and can improve performance [21] [23]. This use of heuristics means some scheduling that would originally have been performed offline can now take place in real-time, improving the reliability of operation [121]. As complexity of systems increases, the use of rule-based schedulers will become more commonplace. While these cannot always give the best response, the outcome will often be suitable in the time taken to calculate it that no detriment to the quality of service will perceptibly take place [18]. These works show that for their areas of research, heuristics are a viable tool and can give comparable results compared to significantly more complicated systems.

While a heuristic system adds complexity, it is thought that the savings generated offset this penalty by offering a great improvement in efficiency. However, models must be reduced to their minimum size, only using key parameters to schedule work; otherwise this efficiency will be greatly reduced. Heuristics are discussed further in Chapter 5.

As demands for real-time systems over the internet such as Skype and YouTube have grown, so has the research into scheduling in this area. For this, the use of Multilevel scheduling has become more popular, allowing traffic with a need to be timely to pass quickly at the expense of data types such as downloads. For this, a great deal of job modelling has taken place to determine best response rates for preservation of Quality of Service [101] [33]. These show that as traffic increases, the ability to manage it in a complete manner becomes more difficult. One solution, presented in [58] is once again the use of heuristics to reduce the investigation length at a small penalty of slightly reduced accuracy. The algorithms developed in this paper worked as well as full-solution algorithms; except in a few specific cases, improving quality of service.

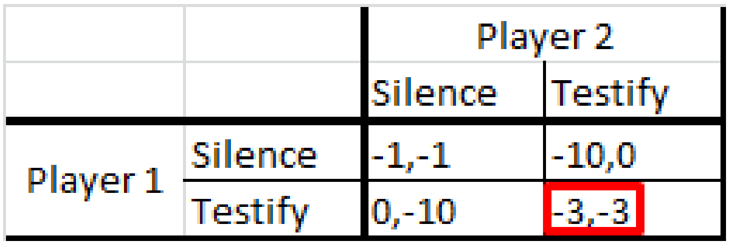
Multi-core and Embedded Systems

Many current real-time algorithms struggle in a multi-core environment due to the added complexity that extra cores gives. A solution has been to dedicate jobs to a specific core (the no migration method), which has allowed current design methods to work in the interim. However, as core numbers increase, this method will not be acceptable and eventually will be impossible for a designer to create [22]. One method discussed has been the ability to change scheduling policy for the algorithm as and when required [57]. These two papers demonstrate that a fixed method is no longer suitable when working with a complex system and architecture, meaning that a scheduler must operate in a hybrid approach, changing when necessary, along with managing a mixture of job types and environments. Therefore, while existing schedule types such as EDF and RM are of great use currently, as systems increase in complexity and microprocessors grow in transistor count and number of cores, it can be expected that statistical and heuristic based schedulers will grow in use and ubiquity.

2.2 Game Theory

2.2.1 Origins

Game Theory can trace its origins back as far at the 18th century, when the first mathematical strategy solution to a problem was recorded by James Waldegrave [11]. However it was not until the late 1940s that true investigation into this branch of mathematics took place. In their work of 1944 [116], John Von Neumann and Oskar Morgernstern concentrated on the sub-games known as two person, zero sum. Within these games, wins represent a gain of some worth,



		Player 2	
		Silence	Testify
Player 1	Silence	-1,-1	-10,0
	Testify	0,-10	-3,-3

Figure 2.5: The Prisoners Dilemma
Nash Equilibrium is marked at $(-3, -3)$

while a loss leads to a deficit of an equal amount. Therefore, when this game is repeated many times, the outcome for both players will be zero [12]. Von Neumann was the first to develop this understanding and give consistent solutions across these game types. A zero sum game is shown in Figure 2.8.

As understanding of the theories behind Game Theory grew, the situations presented became more complex and elaborate. One of the most well-known is the Prisoners Dilemma, originally suggested by Malcolm Tucker, which leads to a non-ideal solution for both players. This optimal outcome is a Nash Equilibrium, made famous by John Forbes Nash in his seminal work [72]. The Prisoners Dilemma can be summarised as follows,

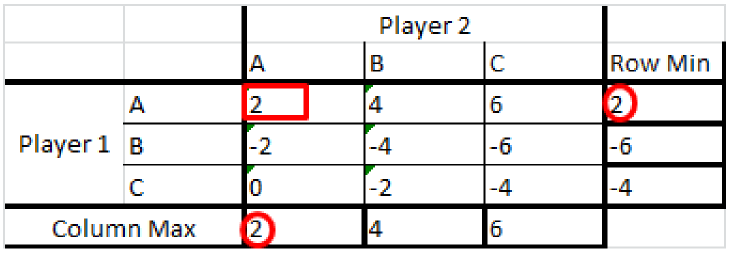
> *After arresting two men, the police place each of them in separate rooms. A deal is offered to both men independently. If they testify against their partner and their partner says nothing, they will go free while the partner will get ten years in prison. If both refuse to testify, they will go to jail for one year. If both testify against one another, they will both be jailed for three years.*

The game can be seen as the matrix-form model in Figure 2.5, where the matrix shows the pay-off for each decision. Matrix-form is commonly used in Game Theory to give a simple summary of outcomes for strategies. Within Figure 2.5, the first number in a cell corresponds to the outcome for Player One; with the second number the outcome for Player Two. For strategy pair (Silence, Testify) in the top-right corner, this gives Player One a ten year sentence (A score of minus ten), with Player Two receiving a zero year sentence and being set free. Nash proved that the optimal action in the Prisoners Dilemma is to testify against your partner. This proof became known as the Nash Equilibrium,

> *The optimal outcome of a game is one where no player has an incentive to deviate from his or her chosen strategy after considering an opponent's choices.*

In this example, this seems counterproductive, as both staying silent leads to a shorter sentence than testifying. However, by remaining silent a prisoner trusts his partner to cooperate and failure to do this will lead to a long sentence.

When this experiment was conducted in the 1960s with candidates playing for small amounts of money, 90% of the pairs suffered from mutual or single-party defection (the equivalent of testifying), even though this lead to



		Player 2			
		A	B	C	Row Min
Player 1	A	2	4	6	2
	B	-2	-4	-6	-6
	C	0	-2	-4	-4
Column Max		2	4	6	

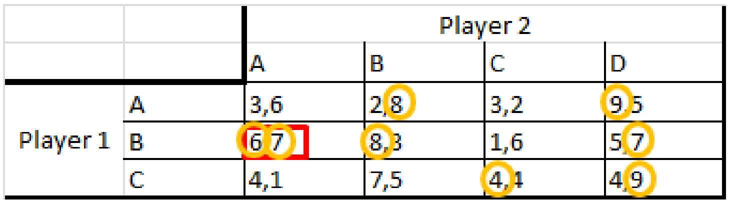
Figure 2.6: Mini-Max Game
This game matrix shows the pay-off values for two players based on choices of strategy A or B. [36]

a reduction in reward over repetitions of the game [99]. Repetition of the experiment with applicants able to discuss their strategies beforehand lead to no statistical difference in results, allowing us to conclude, in an interesting aside, that humans are found to be more tempted by short-term gains than long-term outcomes.

Within these two person games, the recognition of Nash Equilibriums is key in determining the strategy taken. In a deterministic game, it can be assumed that a player will attempt to maximise their reward from the game while minimising their opponents'. This results in a minimax and maximin approach detailed in Figure 2.6. For this, as choice is simultaneous and the pay-offs are known, both players will attempt to minimise their opponents gain while maximising theirs. Therefore the optimal strategy for Player One will be largest minimum value for each row (the Maximin, circled) while Player Two will choose the smallest maximum value (Minimax, circled). In this example, the Strategy A1,B1 is optimum and therefore a Nash Equilibrium. In a matrix where players receive different pay-offs for choosing a strategy, such as that in Figure 2.7, Player One's best choices against Player Two's Strategies A-D are B, B, C and A respectively. Player 2s best choices against Player 1s Strategies A-C are A, A/D (as these both have the same pay-off against Strategy B from Player 1) and D respectively. Therefore, in line with the definitions above, Case B/A gives a Nash Equilibrium and the best choice for each player.

Of course, in a highly non-deterministic system, the Nash Equilibrium at one instant may not remain stable. In [65] the authors make the case that strictly seeking Nash Equilibrium may not be ideal, meaning a compromise must be developed within the algorithm. Despite this, other works such as [53] and [111] show that within a two-player non-competitive game where a Nash Equilibrium exists, this often leads to a better outcome for highest priority jobs and therefore should be used over more complex searching for Pareto Optimal outcomes where all players have their maximum possible return. The Nash Equilibrium is found by searching for each players prudent outcomes (the best outcome for each tactic with respect to your opponents choice of tactic). If both players have a prudent outcome in the same tactic pair, this pair is a Nash Equilibrium. An example of this is illustrated in Figure 2.7.

Formally the Nash Equilibrium is defined on (S, f), a game containing n players and S strategies, where S_i contains the strategy for player i ($S = S_1x, S_2x, ..., S_nx$) and $f = (f_1(x), ..., f_n(x))$ contains the pay-off where $x \epsilon S$.



		Player 2			
		A	B	C	D
Player 1	A	3,6	2,8	3,2	9,5
	B	6,7	8,3	1,6	5,7
	C	4,1	7,5	4,4	4,9

Figure 2.7: Game Matrix showing the prudent outcomes for each system (orange circles). Player 1 and Player 2 both have prudent outcomes at (B,A), therefore this is a Nash Equilibrium (Red Square)

If x_i is Player i strategy and x_{-i} is the strategy for all other players, then when each player selects strategy i, this gives a pay-off of $f_i(x)$ for each player; dependent on the strategy chosen by all players. $x^* \epsilon S$ is a Nash Equilibrium if:

$$\forall i, x_i \epsilon S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*) \tag{2.1}$$

For the Prisoners Dilemma in Figure 2.5, Equation 2.1 can be seen to be true since a player not choosing the Nash Equilibrium x_i^* to testify will receive an outcome of ten years in jail if the other player chooses their Nash Equilibrium. Therefore, the inequality holds true and (Silence,Testify) or (Testify, Silence) are proved to not be Nash Equilibrium.

2.2.2 Game Types

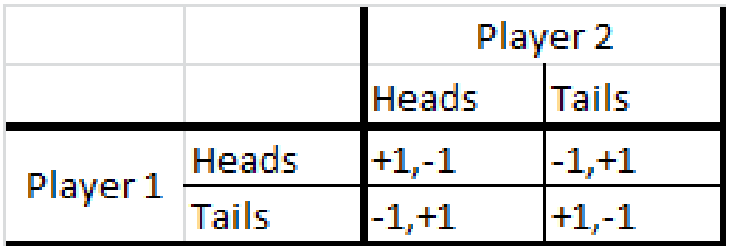
The games played can broadly be split into two types:

- Cooperative: Where players negotiate to ascertain their fair share of a resource
- Non-Cooperative: Where players optimise their strategy to maximise the return from a resource

Games can also be two player or many player (> 2), further adding to the complexity of a situation, and zero-sum or non-zero-sum. In zero-sum games, the expected outcome of each round will be a constant; assumed to be zero, as each player will gain an amount or lose an equal amount. An example of this would be a betting game such as the mixed strategy game matching pennies, shown in Figure 2.8. In this, two players each have a coin and decide whether to show the head side or tail side. After simultaneously uncovering their decision; Player One will win the penny if the faces match while Player Two will win if they are different. Since one player gains a penny while the other loses one, the sum per round is zero. This game has no Nash Equilibrium as no response is ultimately best [12]. This game is formally defined as:

$$\forall x_i \epsilon S_i \sum_{i \epsilon n} f_i(x_i) = constant \tag{2.2}$$

Players within these games can play either mixed or pure strategies. Pure strategies involve playing the best response at all times, which is the Nash



		Player 2	
		Heads	Tails
Player 1	Heads	+1,-1	-1,+1
	Tails	-1,+1	+1,-1

Figure 2.8: A Zero-sum game played between two players.

Equilibrium. In the event of two equilibrium being present, the player will fluctuate between these with a probability $P = 0.5$.

When modelling a situation as a game, it is important to consider several factors [89]:

- Number of players — as complexity increases exponentially with this
- Whether players are monolithic — differences within players sub-goals may compromise outcomes
- Whether the game is repetitive — a one-time play for resources is more likely to result in a free-for-all environment where all players push to maximise their outcomes. However, in repetitive negotiations, a player could find their reputation damaged by overly aggressive tactics; meaning repetitive bargaining is often more cooperative than single-shot events
- Linkage effects — whether a decision has knock-on effects to other sub games
- Existence of multiple issues — these may be mutually exclusive (such as runtime vs. energy consumption)
- Is an agreement required (within a time-frame) — otherwise does a contingency payment or cancellation take place?
- Does the agreement need to be ratified by a higher authority?
- Is a decision binding?
- Are negotiations private or public — can huddles take place between players to agree outside of the primary game, or are all agreements made in sight of all others?
- What are the norms of players — what do they expect or hope to achieve by negotiations?

Of course, when modelling a game theoretic situation for mathematical systems, a great deal of these can be removed. The games can also be designed to control the outcomes and ensure that players choose the best decision based on logic — something that cannot be relied upon with human players.

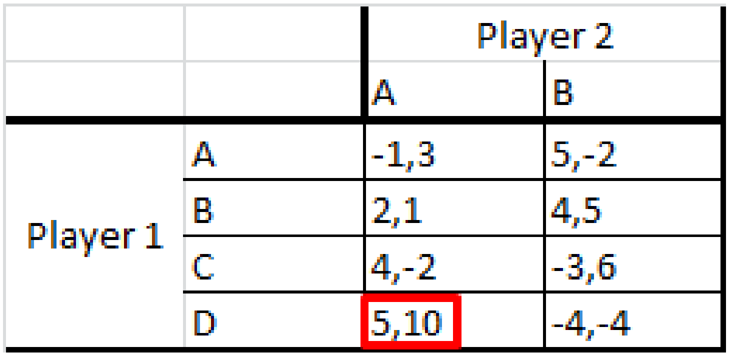
Due to the work by Von Neumann and Nash, the majority of research that has taken place in Game Theory looks at two-person non-cooperative games, especially within finite Automata. Within these two person non-cooperative games, four primary sub games are evident when pure strategies are used [87]:

- **Deadlock Games** (Such as the Prisoners Dilemma) where the Nash Equilibrium is to defect, despite the better outcome from mutual cooperation
- **Stag Hunt Games** where cooperation gives the Nash Equilibrium. Since the Nash Equilibrium also is the best outcome, these games are pareto optimal (No player can improve their outcome without another becoming worse off) [71].
- **Chicken Games** where there are two Nash Equilibriums between which the game will fluctuate
- **Non or Multi Nash Equilibrium games** where no one strategy dominates and the game therefore is played randomly

While a game may have a no Nash Equilibrium using Pure Strategies, Nash proved in 1951 that all games have a Nash Equilibrium in either pure or mixed strategies, building on the work by Von Neumann that proved this was true for zero-sum games [72]. The proof for this is detailed in [37] and is summarised for $2 \times n$ games in Figure 2.9 and Figure 2.10. For the example, Row D Column A is a pure Nash Equilibrium (illustrated by the box). To find any mixed strategy equilibrium, each strategys pay-off curve must be created based on the pay-offs for choosing the particular strategy compared to an opponents choice chosen with probability p. The graphical outcome shows peak strategy pairs and these points can either be solved algebraically or by interpreting the graph to ascertain the optimal value of p to gain this outcome. This is done in Figure 2.10 and proves the pure Nash Equilibrium of Strategy (D, A) at point (0, 5) (Point 1) as this maximises both players return without altering their strategy. Point 4 at (1, 5) is not a Nash Equilibrium as Player 2 could increase his outcome by playing another strategy. Points 2 and 3 require investigation to determine whether they are Nash Equilibriums. Taking the x-value for Point 2, we see it gives a strategy of $(\frac{8}{11}, \frac{3}{11})$ for Player 1 to play Strategy A and Strategy B respectively. Using the same technique for Player 2, we find Player 1 should play Strategy B and D with probability $(\frac{7}{9}, \frac{2}{9})$. This gives us the complete game $(0, \frac{7}{9}, 0, \frac{2}{9})$ $(\frac{8}{11}, \frac{3}{11})$ — which follows the shape of the line made by B and D. Therefore neither player can increase their pay-off by changing strategies; meaning this is a Nash Equilibrium. As mixed Nash Equilibriums are not considered in this research, the full solution for this is omitted from this thesis but can be found in [37].

2.2.3 Limitations of Nash

While a Nash Equilibrium often gives a strong or adequate result, on occasions this is not the case. An example of this can be seen in Figure 2.11 from [41], which is an extended form of the Prisoners Dilemma. However, it can be seen that the Nash Equilibrium is significantly worse for both players than other strategies, but still meets the definition as better outcomes require cooperation between players and is true when applied to Equation 2.1. This idea is confirmed by [65] that shows the stability of a set in a non-ideal state. For this simple two



		Player 2	
		A	B
Player 1	A	-1,3	5,-2
	B	2,1	4,5
	C	4,-2	-3,6
	D	5,10	-4,-4

Figure 2.9: A Mixed Strategy Example

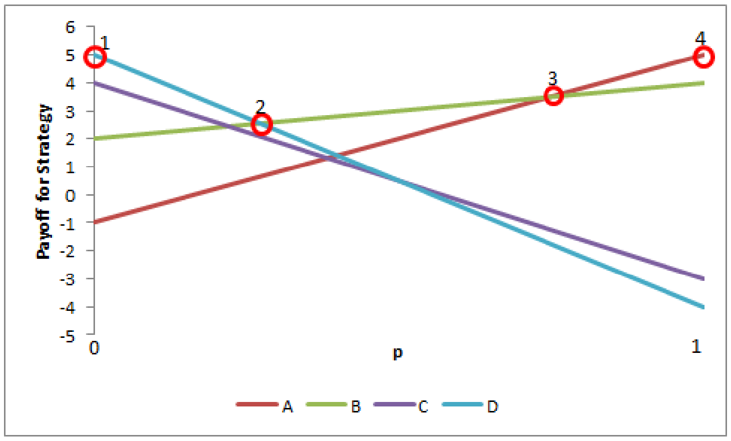
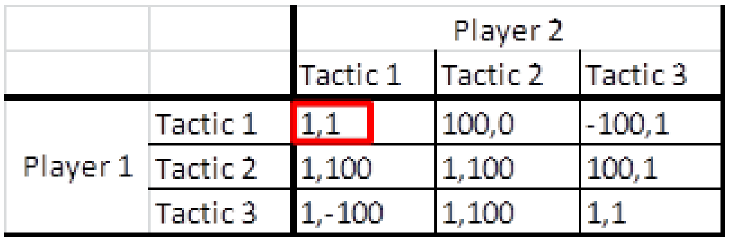


Figure 2.10: Graph of Game Matrix in Figure 2.9

player example, the agreement between players to accept a non-Nash solution may be easy to see; but in more complex environments or many player settings, attempting to convince many players to take the non-Nash solution may be excessively complicated.

Work with Game Theory is widely undertaken in fields as diverse as Biology and Economics, but within Electrical Engineering it is also used, especially within communication theory and traffic routing for items such as bandwidth efficiency and security. It has also been used within automotive embedded controllers to manage Hybrid drive-lines, where items within the cost function have been tuned to give fair weighting and the most ideal outcome [53]. This was modelled as a non-cooperative game where a driver competed with his vehicle drive-train and was found to give good outcomes compared to established algorithms and benchmarks. This gives weight to the idea that non-cooperative games can give improvements over already existing systems, provided they are tuned correctly. To do this, the taxonomy and metrics must be clearly defined at the conceptual stage.

Nash is also limited in effectiveness as the number of players increase. Many



		Player 2		
		Tactic 1	Tactic 2	Tactic 3
Player 1	Tactic 1	1,1	100,0	-100,1
	Tactic 2	1,100	1,100	100,1
	Tactic 3	1,-100	1,100	1,1

Figure 2.11: A Game with a clearly unreasonable (but unique) Nash Equilibrium

player solutions have been developed, but the calculation time for above three players grows until the time for solution calculation becomes excessive when compared to the output time [109]. Cooperative games also have this issue, with attempts to find a solution in Euclidean space becoming overly difficult beyond two-dimensions [3]. A method of resolving this issue is the mixture of both game types in a meta-game, first suggested by Howard in [45], where cooperative sub-games can be used to build coalitions that can be used in a non-cooperative environment; exploiting superadditivity to give stronger outcomes than a purely cooperative or non-cooperative environment [97]. By grouping players into coalitions, a many player game can be reduced as these groups can be deemed a single entity working towards a common goal. Therefore, once multiple coalitions have been formed, the controlling non-cooperative game becomes feasible to calculate and can be created. This method also allows many repetitions through techniques such as swarm optimisation or genetic algorithms, which will identify a local maximum and allow its use even if it is not a Nash Equilibrium [122]. A major concern with this practise is the requirement for several replications of the algorithm to run to identify a non-Nash maximum, which may not be acceptable in a situation where speed is a priority. In these settings, games must be carefully designed to give a clear maximum at the Nash Equilibrium for quick computation of a suitable result, especially in a complicated environment such as a multi-core system [88][110]. Therefore, when optimality of global and dynamic agreement cannot be guaranteed, the use of Nash Equilibrium gives an outcome deemed adequate for most situations.

2.2.4 Computing Nash Equilibrium

Several methods exist for the computation of a Nash Equilibrium, with the Lemke-Howson most commonly used [24]. This works by searching through the matrix space for the best response after beginning at a random strategy. This means there is no guarantee that the first strategy chosen is good and the time for searching through all possible paths increases exponentially [103]. To overcome this, some solutions use the Stackelberg Value, where a leader is appointed, who will always play first and cause other players to base their tactics on this move. Used commonly in economics, this value will equal a pure Nash Equilibrium if one exists in the game space; but will also find a stronger value if one exists. Stackelberg is used widely in communications and traffic management, as these allow not only a leader to be declared, but also all information to be available [106]. For a non-deterministic environment, this means Stackelberg Values can not easily be computed, again meaning that Nash gives a strong result in a significantly shorter space of time.

2.2.5 Game Theory with Real-Time and Availability

Owing to its strength in non-deterministic environments, the exploitation of Game Theory within scheduling has become more common as architectures grow and transistor count rises. This idea of a repetitive game that finds the best operating point for the next period of time has been investigated in several works, notably [3], [25], [122] and [100]. All these works build a schedule based on input factors such as available energy, using non-cooperative games to find solutions. All of these use Nash Equilibriums either as a starting point to search for optimal values from or as the final result. Since Nash is quick to calculate for a two-player game, it gives a local maximum or minimum (the saddle point) and only requires a short search either side of this point to determine whether a better result can be determined. This search is performed through an iterative algorithm and allows improvement over a pure Nash solution in some operating conditions. However, this technique is time consuming and computationally intensive, meaning that a carefully designed game can give comparable results for simply calculating the Nash Equilibrium.

For the highly non-deterministic environments considered in this work, where development of a schedule without *apriori* knowledge takes place and is desirable as soon as possible, the environment must be designed so that the Nash Equilibrium is a maximal solution, thus removing the need for repeated runs of a secondary scoring algorithm. Such an idea has not been found in the literature for practical use and is thought to be an original concept for developing a feasible schedule quickly and accurately.

2.3 Energy Harvesting

2.3.1 Introduction

Though the ancient Greeks understood the concept of electrostatic charge, it was not until the 18th century and the development of the Leyden Jar (a forerunner of the modern capacitor) that scientists began to understand the composition of electricity and how this could be harnessed [91]. With the development of the first rechargeable battery in 1859, combined with the concurrent advancements in dynamo technology, the supply of electrical power through a wind turbine first took place in 1887 through the work of James Blyth [75].

As understanding increased and technology advanced, smaller quantities of energy could successfully be harvested from the environment and stored. Improvements in battery chemistry and efficiency of devices have now led to small systems that can run almost exclusively from scavenged energy as and when needed. This has allowed an increase in inventory turnover and quality of products through the increase in information that can be stored by moving from 1D and 2D bar-codes to passive Radio Frequency Identification (RFID) tags. This has also lead to the "Internet of Things" where self-powered sensors have become ubiquitous and supply terabytes of information on a daily basis.

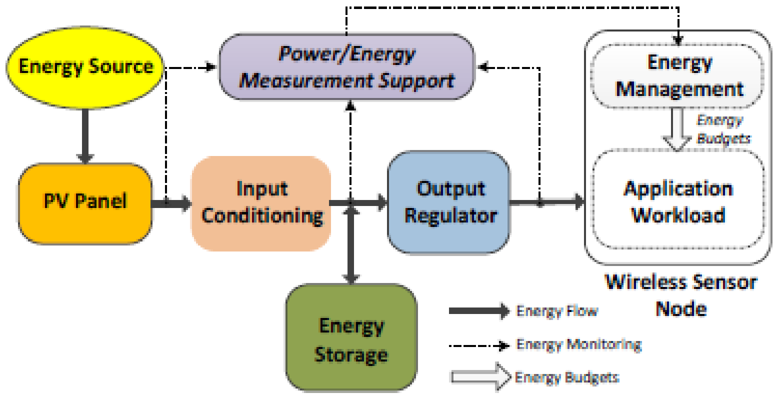


Figure 2.12: An Energy Harvesting System Block Diagram [4]

2.3.2 Harvesting Types

Energy Harvesting systems vary in their design based on the harvesting device used and level of energy delivery required. A common layout for a system can be seen in Figure 2.12, which has conditioning and regulation circuits, along with a storage device such as a battery or capacitor and a management system that can be based in hardware or software.

Harvesters can be of four main types:

- Solar (Photovoltaic)
- Vibration (Piezoelectric and Magnetic)
- Thermoelectric
- Wind

As well as these common types, those taking energy from events such as radioactive decay and pressure differences also exist; but are less common [40].

The power these devices are capable of delivering varies based on a great deal of factors including its location, ambient conditions and suitability of tuning that has taken place. For example, a wind turbine used for micro power generation can deliver between 50 and 210mW depending on wind speed, while a piezoelectric harvester harnessing vibration from a running person may only create 14μW, with this depending on the runner moving at the ideal response rate of the device.

When considered as a statistical distribution, the power delivered by these devices also varies. A Photovoltaic Cell for example will give its maximum output at noon on a sunny day, rising to this and falling to near zero after the sun sets. A vibration harvester mounted in a vehicle by the Holistic Energy Harvesting Group at Southampton University showed a less predictable response, giving peak output when the engine was running at the resonant frequency of the harvester [104]. Therefore, the energy model can vary from a delta function for solar cells, a Gaussian function for vibration devices and a uniform response for

Table 2.3: Average Power Outputs and Models for Energy Harvesting Sources [20]

Harvesting Method	**Power Density**	**Output Model**
Solar Cell	15mW/cm3	Uniform/Delta
Piezoelectric	330uW/cm3	Gaussian
Vibration	116uW/cm3	Gaussian
Thermoelectric	40uW/cm3	Gaussian

a thermoelectric system in a consistent operating environment. This variation makes designing a standard harvesting circuit very difficult, if not impossible.

2.3.3 Deliverable Energy

Photovoltaic Cells

Photovoltaic (PV) cells work through the delivery of electrons from a light source to the exposed p-n junction of a semiconductor diode, typically made of silicon [115]. Normally a PV cell will have its energy delivery quoted in watts per square metre, which can be of the order of $2 - 200mW/m^2$, dependent on the light level and composition of silicon. Cells are designed for use under natural or artificial light and therefore environment for use should be considered when designing a system using PV cells.

Vibration Systems

Vibration energy is harnessed through either Piezoelectric or Electromagnetic devices. Piezoelectric material, made of materials such as polyvinylidene fluoride, can be manufactured in very thin sheets (100μm) and has the internal dipoles aligned to allow rapid transfer of electrons when the sheet is placed under tensile stress. The devices give a predictable response through a long life ($>$ 1000000 cycles). The generated voltage is directly proportional to the strain present, giving 170V and between 5-20$\mu J/cm^2$ for a 1 percent strain. These systems are $<$ 5 percent efficient, meaning a great deal of harvested energy is lost and this technology is still only usable in a small number of energy harvesting cases [64].

Electromagnetic devices work through the passing of a magnet through a coil to induce an electrical current. Unlike Piezoelectric devices, which deliver their energy in bursts, electromagnetic systems are much smoother and easily rectified into usable power for devices and can also be manufactured in compact casing of 70mm with outputs of $100\mu W$ possible.

Both methods of vibration harvesting require tuning to their mounted points resonant frequency and power output. Operating outside this range will give degraded output compared to operating at the system ”sweet spot”. Vibration sources can vary between 0.1 and 10 m/s^2 vibrational power and between 60 and 240 Hz frequency [93].

Thermoelectric Harvesting

Using the Seebeck Effect, where heat presented to dissimilar metals generates a voltage at the junction of around 0.2mV/K, thermoelectric harvesters are used with sources above ambient such as hot water pipes or engine blocks. These devices give good proportionality to the temperature difference present and can be sandwiched together to give suitable voltages to run embedded devices.

The technology used in these devices has been known for many years and used in thermocouples, so is a well exploited system, but requires good temperature differential between the hot and cold sides ($> 25K$) to be effective, which can be a limitation as many hot sources are heavily insulated; thus preventing thermoelectric devices being exploited fully due to the lack of suitable environments.

Wind Turbines

While large scale wind generation for the National Grid has become commonplace, the use of micro-scale generation for small DC devices has grown in popularity as an emergency or alternative charging source [35]. A major issue with wind is the high variability over short periods — giving an almost stochastic response. Small turbines can deliver between 2-200mW, depending on their mounting and ambient conditions. Due to the high variation present, wind can be considered useful as a standby harvester, or to top-up and extend operational life, but is not deemed suitable as the sole source for a device unless its duty cycle is very low [80].

2.3.4 The Energy Gap

Based on the environment the harvester is to be used in, the choice of harvester can be an arduous process. Designers must consider whether the harvester is present to extend operational life or to completely power the system, along with peak power that can be delivered, cost and mounting of the overall device. Currently, PV systems are very popular due to their high power for area and easy integration into an embedded device. Since this also is away from any vibration or heat, the risk of secondary damage to the system being powered is low. As the main cost of an energy harvesting system is the storage device, reduction of this and the ability to run an embedded system from a more unpredictable power supply is of great interest to the energy harvesting community and of high relevance in the near future.

Within an environment such as moving vehicles, PV has been used to add additional energy to electric vehicle batteries for running low voltage devices such as GPS systems without placing strain on the main Lithium Ion Batteries. However, within a vehicle powered by internal combustion, the main source of harvestable energy is vibration from the engine and a secondary lower frequency from the suspension. Therefore, for the purposes of this Thesis and owing to the maturity of research into PV, this Thesis will investigate the methods of managing energy harvested from Gaussian sources such as vibration harvesters.

Mechanical modelling for vibration harvesters has taken place, allowing simulations of expected energy to be completed [5]. All harvester types deliver varying power density and voltages and require application specific design of the end system to deliver maximum efficiency (see Table 2.3). Since these harvesters can only deliver small amounts of energy, embedded systems are being designed with ever smaller core voltages, meaning that DVFS and other techniques that lower the device voltage would not be feasible to embed. Therefore, a system with adaptive duty cycling that rejects jobs would assist in increasing system reliability, with tests found to give positive results [46].

A standard model for energy harvesting [112] can be seen in Figure 3.1 . This contains an energy harvester that provides energy at the power supply level (P_s) supplemented by an initial value of energy (B_0) within the storage device, from which the energy is consumed by the circuit. Consumption can be thought of as dynamic power based on the duty cycle of the processor (P_c) and a near-constant loss due to the manufacturing process and switching effects (P_{leak}). This model can also be thought of as equation (2.3). Energy is then accumulated over a period T and consumed by P_c and P_{leak}. Since a value of energy cannot be negative, a form of ceiling function is used where if the outcome of an equation is less than zero, it is rounded up to zero [46]:

$$\begin{aligned} E(T) &= B_0 + \eta \int_0^T \lceil P_s(t) - P_c(t) \rceil^+ dt \\ &\quad - \int_0^T \lceil P_c(t) - P_s(t) \rceil^+ dt \\ &\quad - \int_0^T P_{leak}(t) dt \end{aligned} \tag{2.3}$$

- B_0^{Max}= Maximum value of Energy the Energy Storage Device can accept
- B_0 = Initial value of Energy in the Energy Storage Device at time T=0; ($0 < B_0 < B_0^{Max}$)
- $E(T)$ = Available Energy; ($0 < E(T) < B_0^{Max}$)
- $P_s(t)$ = Power Harvested
- $P_c(t)$ = Power Consumed
- $P_{leak}(t)$ = Power Lost through Leakage
- η = System Efficiency
- $\lceil x \rceil^+ = x$ if $x > 0,\ else\ 0$

As well as different parameters of operation, these harvesters also deliver their power through varying output profiles [69]. For example; solar cells can be expected to deliver consistent power through the hours of daylight, while vibration harvesting will vary based on the duty cycle of the item the harvester is mounted upon and whether the harvester is correctly tuned to compensate and maximize for this [27]. Though the conversion efficiency of a harvester may be very low, the ability to capitalize on waste energy means a low-cost reliable harvester becomes competitive compared with maintenance costs for other solutions [94]. These techniques have been used in varied sources, from self-winding watches to experiments charging cellular devices through energy expanded by walking [80].

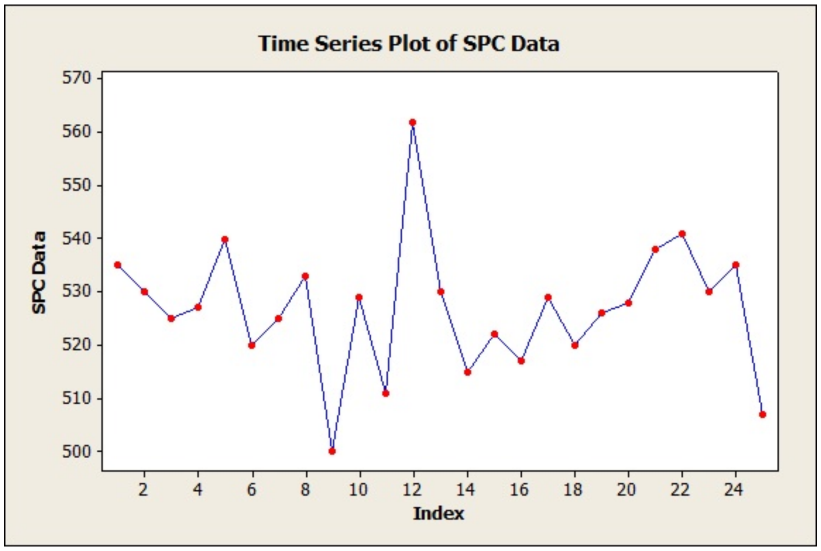


Figure 2.13: Data monitored without SPC

2.4 Statistical Process Control

2.4.1 Background

Control of a process within Statistical Process Control is performed by constant monitoring of the outcome and reacting at critical points. However, the key issue is identifying these critical points. Looking at Figure 2.13, which gives an example of sales figures from [117] without using SPC, a reaction may have taken place after point 13, as this seems to show a dip from the high sales at Point 12. Indeed, in a business situation bonuses may have been paid out due to this peak and a manager could be lulled into the belief that something they did could have caused the rise in a prime example of *Post hoc ergo propter hoc* (after this, therefore because of this). What Figure 2.14 shows is the same data, but with upper and lower control limits identified. These are the plus and minus three-sigma points for the data under the graph, which 99.73% of the data should occupy [56]. As can be seen point 12, despite approaching the upper control limit, is not more than three standard deviations from the mean and therefore can be considered part of normal process. This investigation, along with others can be used to identify key points and allow reactive and in some cases pre-emptive action to take place.

2.4.2 MINITAB

Developed in 1972 from the successful Omnitab system, Minitab is a columnar-based statistical tools package used heavily with the implementation of six sigma and for other statistical tests and experiments [66]. Used widely in business and academia, it is seen as the industry standard for statistics, computation and graphics [96]. Within this Thesis, a great deal of the statistical analysis; from

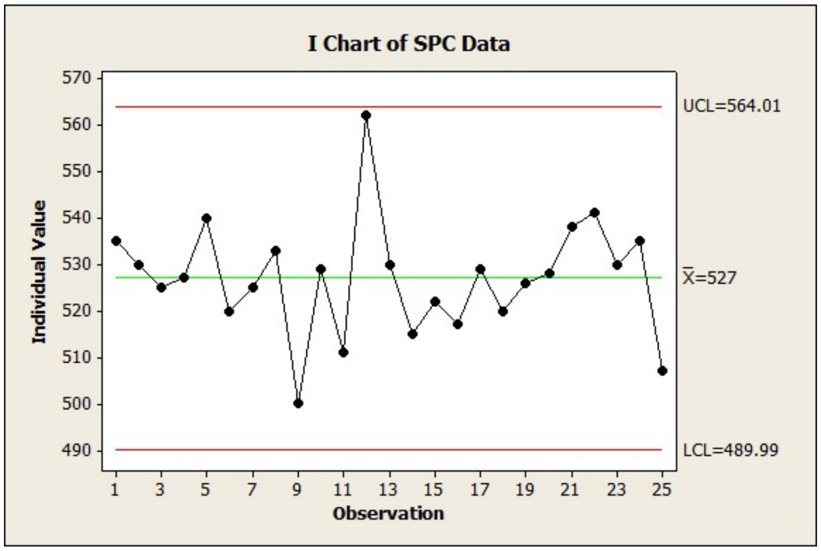


Figure 2.14: Data monitored with SPC
This shows the peak in Figure 2.13 is nothing more than normal for the process

graphs to Design of Experiments (DoE) have been set up, run and analysed using Minitab. For certain areas, such as Capability and DoEs, a brief introduction with reference to the Minitab methods is included within this chapter.

2.4.3 Capability

A process can be seen as a combination of items and processes that produce a quantitative output. Since this output will have natural variation, it can be evaluated.

For a given process, there will be upper and lower limits of specification for suitable parts; known as the upper control limit (UCL) and Lower Control Limit (LCL). Parts that fall within these limits will be classified as conforming, with those outside rejected. When new products or designs are evaluated, a process capability study is often carried out. This will record all instances of the output for a given period, such as a shift of work, and determine the distribution of the output to gain a mean and standard deviation. Once this is complete, the number falling within the UCL and LCL are calculated, giving a process capability index (Cpk). This is calculated as follows:

$$Cpk = min(\frac{(UCL - Mean)}{3\sigma}; \frac{(Mean - LCL)}{3\sigma}) \tag{2.4}$$

A Cpk of 1 implies that the process fits in the specification limits with no space for deviation, meaning that any deviation within the process will cause a percentage of parts to be non-conforming. For a process to be accepted within many industrial scenarios, a $Cpk > 1.67$ is desired, allowing a 2.5 sigma drift before outputs would begin to be out of specification. This is due to the evidence

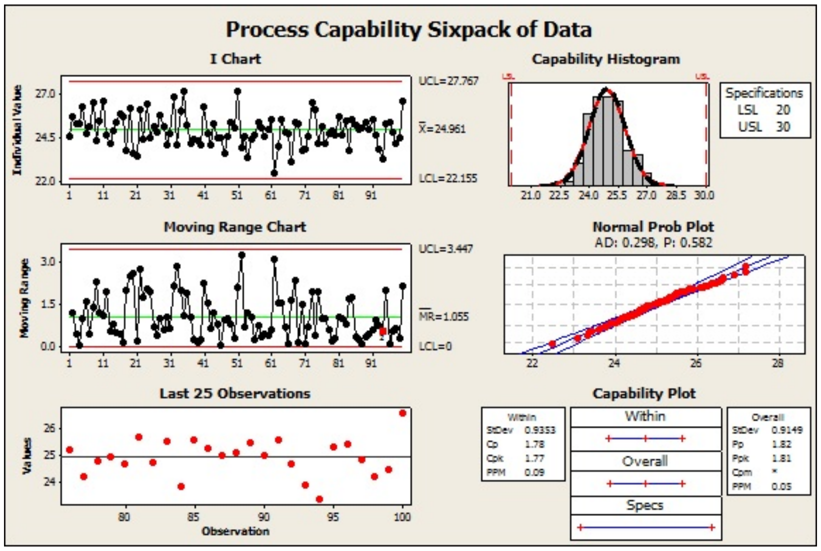


Figure 2.15: A Capability Report from Minitab showing a Cpk of 1.78 for a distribution

that an output can drift up to 1.5 sigma when a process moves from capability to production due to stack up of process deviations and other such phenomena.

To show this in Minitab, data was collected from a normally distributed source with mean of 25 and standard deviation of 1. 100 data points were analysed using the capability six-pack within Minitab, the results of which can be seen in Figure 2.15. The embedded charts show stability plus/minus three sigma for each value and its change from the previously recorded value (I-MR), along with the ability to see its shape through a histogram and Normality through a Normal Probability Plot. Finally, the Cp (shape of the distribution) and Cpk (shape considered with centring). These six graphs allow a simple overview of a process and whether it is stable and requires improvement.

Drift in a process owing to part wear or changes in material composition can cause $Cpk < 1.67$, meaning the risk of items being out of specification may occur. Methods for returning capability to its target include maintenance of the equipment, review of the component design or to alter the UCL or LCL.

2.4.4 Six Sigma

The desire for quality within manufacturing has existed since the dawn of the industrial revolution. While initially trained craftsman would labour at creating a quality piece, with their reputation suffering if an item failed to meet customer expectations; as mass manufacture grew in use, the need to take action to identify and prevent a failure became more prevalent. Building from ideas initially used by the United States Military during World War Two, Motorola looked at improving its product quality in the 1980s through the use of SPC, Capability Studies and a variety of other techniques to manage the key process input variables (KPIVs) to reduce defects [56]. Six Sigma can be thought of as a tool-kit to perform these improvements through what is known as the DMAIC Process (Define, Measure, Analyse, Improve, Control) [16].

Any outcome, be it manufacture of electronic components or the management of consumer complaints over the phone can be viewed as a process. As such, they will have inputs (X), and outputs (Y) with $Y = f(X)$ as the inputs are transformed by the process ($f()$) into the output. All inputs:

- Must be understood
- Have Variation
- Must be controlled
- Have Capability (see section 2.4.3)
- Need Improvement [78]

As there can be many inputs for one end-product, items that are either uncontrollable or of low importance must be removed from the model in order to allow key variables to be identified. This is performed through the use of Six Sigma Tools, creating a funnel effect to reduce from tens of inputs to perhaps five critical items, known as key x's . The funnel effect begins by listing all known factors in the define phase and using techniques such as cause and effects analysis (see Chapter 6 for an example of this) in the measure phase. Once this has taken place, control of the process and preventing drift or faults occurring becomes the primary goal [78]. This is feasible as the number of inputs ideally should be in the region of 20-30, a reduction factor of four.

Once the inputs have been reduced, those left are statistically analysed through a variety of tools. This is deemed passive evaluation, where the current process data is collected, investigated and reviewed to find any discrepancies. A simple example could be one specification of the component manufactured having statistically higher values of failures than others. Further drilling down on this specification could identify a key difference in either the design or manufacture which can then be taken to the next step. Passive evaluation is present in this Thesis in Chapters 3 and 4 and reduces inputs down to five or six, ideal for the next phase.

Improvement of a process is active evaluation, where alterations are made and their consequences on the outcome are recorded. The most common method for this is the Design of Experiments (DoE), which systematically determines the contribution of a factor, along with any interactions that take place with others in the process. Once the KPIVs of a process are identified, a value for their set up can be determined and tested. This means for n KPIVs, 2^n runs will have to be completed for a full factorial run, with repeats if more accuracy is required. If n is large, the number of runs can be reduced through the use of lower resolutions at the penalty of lower confidence in the changes [102]. DoE's are widely used in this Thesis across all technical chapters.

Following collection of data in a DoE, the response is analysed across all levels of interaction. For a three-variable DoE, this will mean initially analysing A, B, C, AB, AC, BC and ABC for any statistically significant effects on the outcome. Figure 2.16 shows the initial outcome of a 2-level, 3-factor DoE with alterations to A and AC identified as significant. This means the third level (ABC) can be removed and the DoE analysis repeated to ensure no other effects become significant. As can be seen in Figure 2.17, B now becomes significant. Graphs of Main Effect and Interaction in Figure 2.18 show contribution of altering variables and whether these alterations cause constructive or destructive interference with another variable.

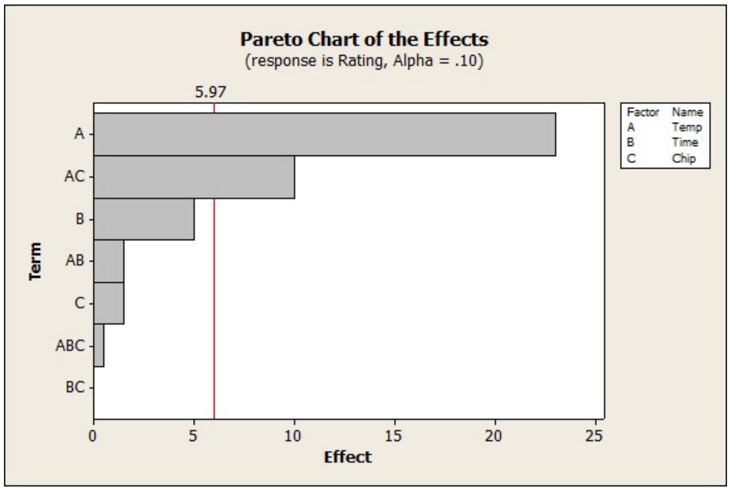


Figure 2.16: DoE Results showing ABC is insignificant

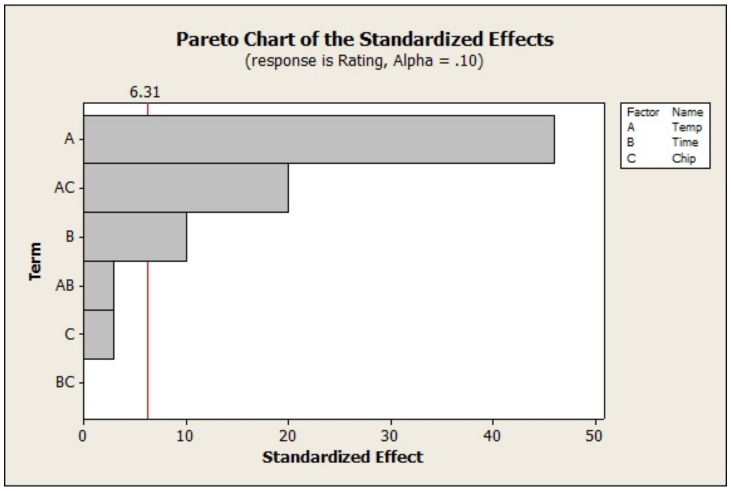


Figure 2.17: Reduced DoE removing ABC now shows A, B and AC are significant

Upon completion of the Factorial Analysis, the contribution given by each variable or combination of variables can take place using an Analysis of Variance (ANOVA). In Six Sigma, often the General Linear Model (GLM) is used, as this allows univariate analysis [16]. From this, the Sum of Squares outcome (sum of squared deviation relative to the mean) can be used to calculate the Epsilon

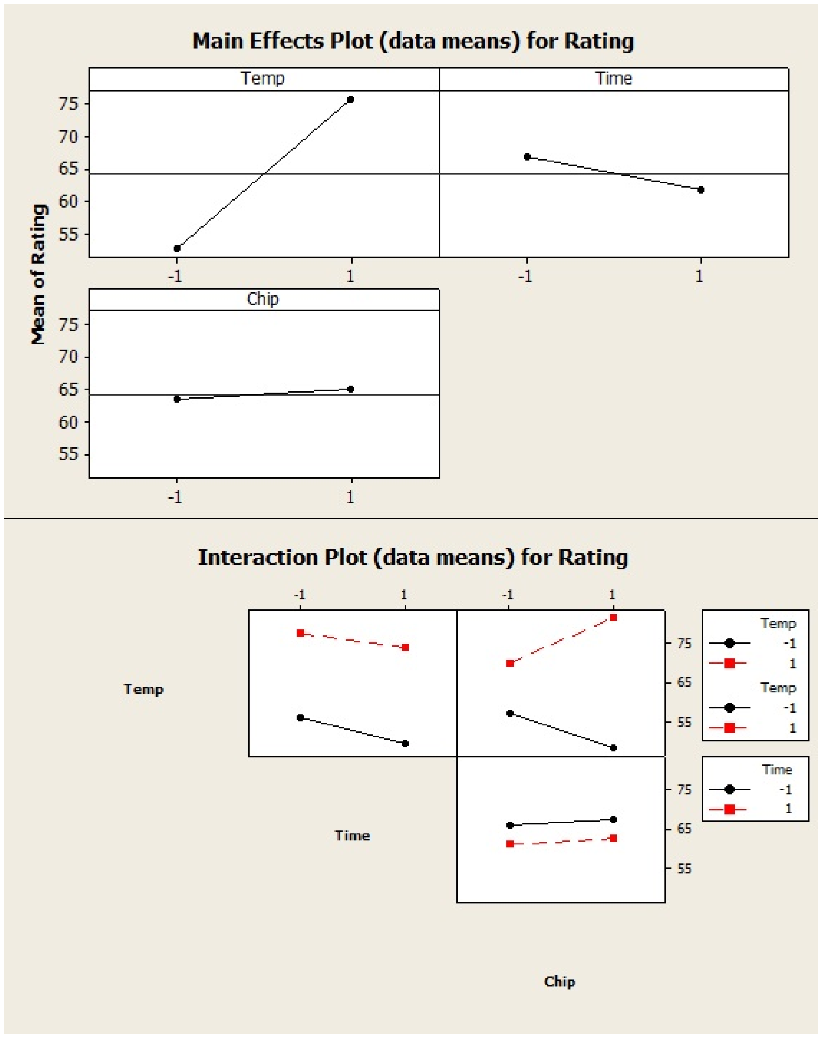


Figure 2.18: Main Effects Plot
This shows Temperature alteration has greatest effect, with interactions plot showing a possible interaction with Chip in the top-right plot. This would be investigated further to ensure process alterations were not causing problems

Squared value, which is the percentage contribution to the outcome the factor, combination of factors, or error gives. Therefore, if the error is high, results of the DoE may not be trusted and a repeated experiment after further analysis could be required.

2.5 Automotive Benchmarks and ISO26262

Within Chapter 6, an embedded device is used to create a fuel Electronic Control Unit (ECU) for use within an automotive environment. However, to allow this to successfully be used outside of a laboratory environment, the micro-controller and other components used require automotive qualification. This would be performed by completion of tests under ISO26262 and the sign-off of parts by an independent testing agency. The Benchmarks used are discussed below, along with the ISO standard.

2.5.1 CAN System

Since the definition of CAN by Bosch in 1984 and the subsequent definition of ISO11898 in 1991, the Controller Area Network has rapidly gained popularity within automotive design owing to its defined protocol, feasible high speeds and high resilience. Data rates of over 1Mbit/s are possible on short bus lengths ($< 30m$) with only one twisted pair wire required. In addition to this, CAN allows for priority, real-time response, resistance to short circuits and non-destructive bus access. A standard topology can be seen in Figure 2.19, with two lines (CAN-H and CAN-L) joined in a twisted pair to reduce EMI, terminated by 120Ω resistors to prevent reflections. This design means a system can operate with reduced capability in the event of a single line failure and also gives the ability to manage simultaneous transmissions through arbitration. This is done by assessing the priority or source address in the CAN Frame, shown in Figure 2.20. This frame design allows for messages to be sent to specific Source Addresses or for Broadcast Announce Messages (BAM) to be sent to all nodes, accomplished through the Destination Address within the Parameter Group Number (PGN). In the event of two nodes attempting to transmit at the same time, the lowest Source Address within the PGN will dominate through an arbitration phase of the PGN. This can be achieved as dominant (low) bits will pull the CAN Bus low, meaning the highest priority signal is preserved and a node sending a recessive (high) bit will cease transmitting. This is illustrated in Figure 2.21, which also shows the non-destructive nature of the arbitration.

2.5.2 Vehicle Speed Sensor System

Knowledge of the engine speed and position relative to top dead centre (TDC) are important factors for complete combustion and reliable running of the engine. The translation of this speed through the transmission into the wheels is also important for a vehicle control unit to know for items as diverse as ride comfort and passenger safety. Engine speed, engine load and wheel speed in modern cars are usually calculated through the use of magnetostatic sensors, such as those making use of the Hall Effect. These give good detection at zero speed and accurate readings in a compact, highly resilient package [28]. These Hall Effect sensors are mounted close to a toothed or notched wheel, giving a digital output corresponding to the proximity of a tooth to the sensor. The wheel can be encoded through the use of gaps or opposing magnetic poles to give a signal that identifies TDC, allowing an ECU to react accordingly. A typical encoding wheel will contain 58 teeth with a two-tooth gap, leading to simple calculation of engine or vehicle speed through elementary mathematics. The

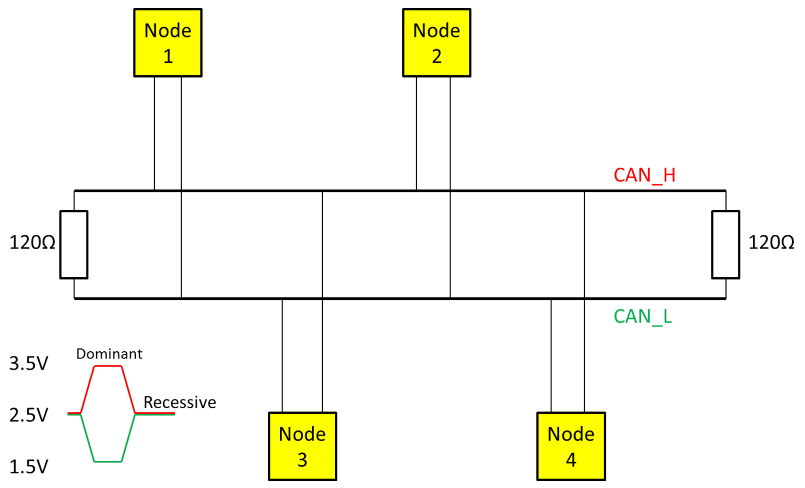


Figure 2.19: A Typical CAN Topology

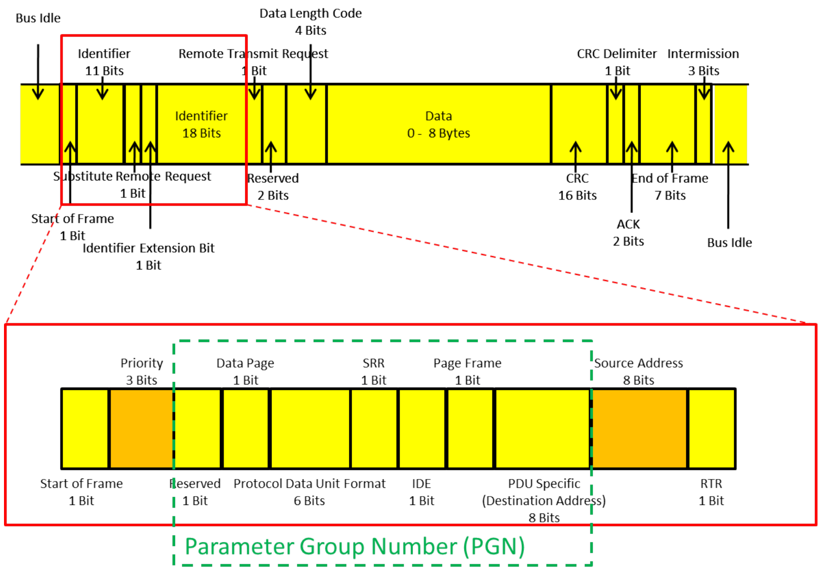


Figure 2.20: An Extended CAN Frame, used in Automotive CAN [34]

algorithms that interpret this signal must also detect possible efforts to tamper the system (such as supplying a constant signal to bypass any speed governor) or any faults that may occur with the sensor.

2.5.3 Fuel Injector and Spark Control System

The control of fuel supplied to a cylinder and the time of combustion is probably the most critical factor in vehicle reliability, efficiency and operational quality.

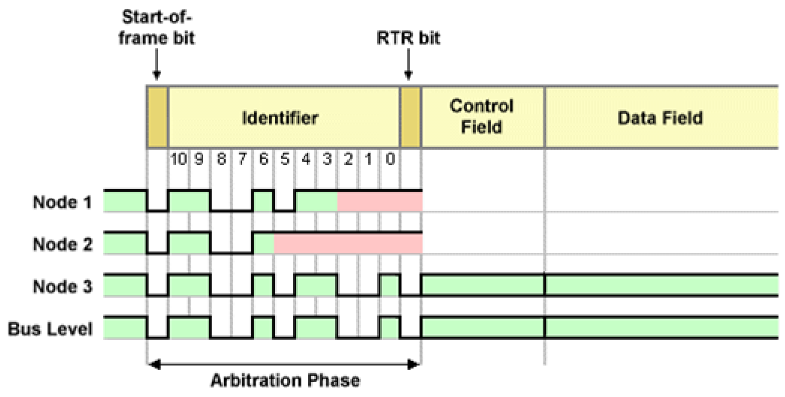


Figure 2.21: Example of Arbitration in a CAN Network [14]
In this example, Node 2 ceases transmission at Bit 5 of the Identifier after sensing a dominant bit from Node 1 and Node 3. Node 1 ceases at Bit 2, allowing Node 3 to win arbitration and control the CAN Bus. The Bus Level is preserved to the data from Node 3.

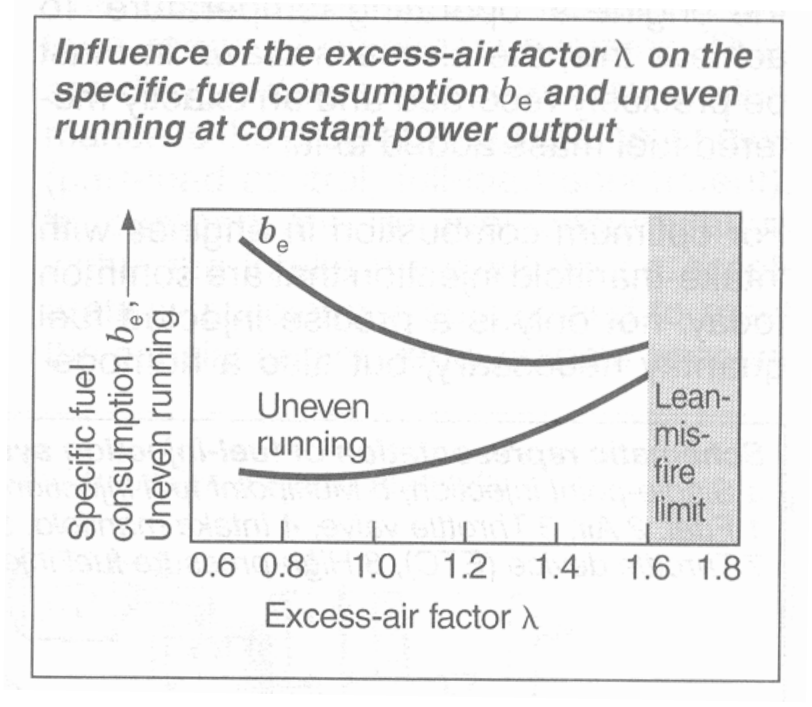


Figure 2.22: Graph showing how alterations in λ change fuel consumption and engine running [28]

Until the introduction of the Corporate Average Fuel Economy (CAFE) regulations in the United States in 1975 [73], most petrol engines were controlled mechanically, with a carburettor mixing fuel and air and spark time controlled by a distributor cap, which energised the spark plug based on camshaft rotation. Since the design of carburettors to meet the new US and European emissions standards were significantly more expensive than an equivalent fuel injection system, these rapidly gained acceptance and are now commonplace in automotive engines.

As systems have become more advanced, higher efficiency and power output have been made possible through relocation of the fuel injector, use of higher pressure within the fuel system (over 10MPa in a modern direct injection system [28]) and the reduction of emissions through the introduction of catalytic converters and particulate filters. Since ideally 14.7kg of air is required to burn 1kg of fuel (the stoichiometric ratio) engines aim to run as close to this as possible, with the Air to Fuel Mixture Ratio (λ) between 0.8 and 1.6 depending whether the engine is running rich (high fuel consumption to even running; used when the engine is cold) or lean (low fuel consumption to some uneven running) as shown in Figure 2.22. Figure 2.23 shows how the variation of λ also affects components of exhaust gases; where stoichiometric conditions maximise the production of Oxides of Nitrogen (NOx) and lean conditions increase Hydrocarbons. Both of these are controlled by the use of Catalytic Converters, which use metals such as platinum and rhodium in reactions that cause less harmful gases to be emitted from the tailpipe. For example $NO_x + O_2 \rightarrow xO_2 + N_2$. Since poor combustion can lead to damage of the catalytic converter, misfires and a reduction in the total operating life of the engine, timing of the ignition spark is vital to preserve the desired value of λ. How soon or late relative to TDC this occurs is relative to the engine speed and load, both of which are determined by sensors on the crankshaft and transmission. As the propagation of combustion is relative to engine speed and delivered fuel, but ideal combustion always occurs just after TDC, Ignition must begin before TDC and adjust based on speed and load changes; as well as the value of λ, as lean mixtures require the ignition time to be brought forward (advanced) further still. With the introduction of electronic control, the spark timing can be thought of as a function of these three inputs, allowing a complex ignition map to be created, such as the one in Figure 2.24. The timing of ignition must also compensate for the time required to propagate current through the primary ignition coil, induce a current in the secondary coil and deliver the spark. The length of time this current is allowed to propagate is known as the dwell period and requires calculation by the fuel system to deliver optimum combustion.

2.5.4 ISO26262

Ratified in 1991, ISO26262 is an automotive standard for the functional safety of road vehicles with a maximum weight below 3500kg [49]. This is designed to give a framework for the development of automotive components from concept to decommissioning and a level of testing that must take place for each of these based on their Automotive Safety Integrity Level (ASIL), which is determined using quality tools such as a Failure Modes Effects Analysis (FMEA). Components themselves do not have ASIL's, rather events occurring due to issues with the component. For example, the spontaneous and unexpected inflation of an

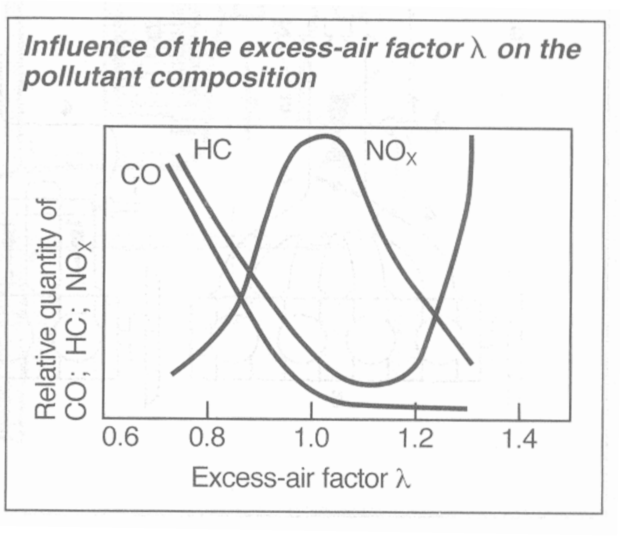


Figure 2.23: Graph showing the change in tailpipe emissions for variations in λ [28]

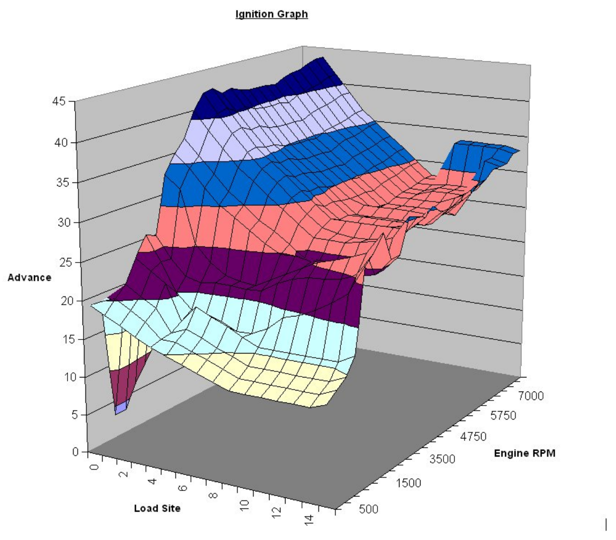


Figure 2.24: A Complex Ignition Map [1]

air-bag would be classified as an ASIL-D event (the most severe) but the air-bag control system would also be required to manage many other events at differing ASIL levels. The standard has been well received by the automotive industry and used on items such as the battery system for the Chevrolet Volt Electric Vehicle [43].

ISO26262 is only a guideline for severity, stating that an event deemed an ASIL-D should have an occurrence rate below $10^{-8}h^{-1}$, but must be used in con-

Table 2.4: Recommended Testing for Differing ASIL Levels [92]

0=Not required, += Recommended, ++=Highly Recommended

Method	**ASIL Standard**			
	A	**B**	**C**	**D**
Environmental with basic functional verification	++	++	++	++
Expanded Function	0	+	+	++
Statistical Test (Common Operating Mode)	0	0	+	++
Worst Case Test	0	0	0	+
Over Limit Test	+	+	+	+
Mechanical Test	++	++	++	++
Accelerated Life Test	+	+	++	++
Mechanical Endurance Test	++	++	++	++
EMC/ESD Tests	++	++	++	++
Chemical Resilience Test	++	++	++	++

junction with engineering knowledge and quality management such as TS:16949 (standard for quality management for design, development, production, installation and servicing of automotive products) to determine the battery of tests that must be used [85]. Though the use of ASIL levels to generically standardise a system is not common within the ISO standard, many manufacturers base their performance testing on these levels for a system [92]. For electronic components, recommended tests for an ASIL-D system include mandatory items such as Electromagnetic Compliance and Immunity (EMC/EMI) and Electrostatic Discharge (ESD), as well as the use of Highly Accelerated Life Testing (HALT), where the system is subjected to extremes of temperature, humidity and vibration to exacerbate any failures. This aims to deliver the $< 10^{-8}h^{-1}$ occurrence rate and common automotive warranty rate of exceeding 160,000km [79]. While achieving this level of mileage or time would not be possible in a normal development life-cycle, the use of reliability estimation combined with HALT or over-limit testing can establish confidence in a system for automotive qualification. When non-permanent faults such as software issues are considered, these models can be factored with the tests in Table 2.4 to give a system wide metric for combined hardware and software failures. These failures are usually grouped into three classes:

- Those that cause danger or sudden non-operation
- Those that cause significant detriment to performance
- Those that reduce functionality while still allowing use

For an entire system; no occurrences of the top class should occur across the life of the vehicle, with no occurrences of the second class during the warranty period, which is now commonly 160,000 km or five years of operation. The third class of issue may occur infrequently (< 1 within the warranty period) and can be rectified at the service interval.

Chapter 3

Energy Harvesting

We demand rigidly defined areas of doubt and uncertainty!
Douglas Adams, The Hitch-hikers Guide to the Galaxy

3.1 Introduction

Modern systems running on stored energy are expected to have a long lifetime (reliability metric) while also maximising useful work and availability [47][51]. Initially mobile systems worked using batteries as their primary source of power, but these either required recharging from a mains power source or monitoring for replacement. As the number of devices increase, the time and labour required for this becomes infeasible. Therefore, energy harvesting systems now work in one of three ways [86]:

- Battery Supplemented: Where the harvester supports the battery to reduce its use
- Autonomous: Where the harvester alone meets the system power needs
- Autonomous Hybrid: Where the battery acts as a reservoir, only charged by the harvesting device

3.2 Methodology

3.2.1 Energy and Scheduling

Energy is a key metric for all next generation microprocessors, with consumption minimisation noted as a key investigation item by the International Roadmap for Semiconductors (ITRS) [50]. Harvesting Aware Power Management (HAPM) can work at all levels of the device, from System to peripherals, with techniques such as sleep modes, sensor management and duty cycling employed in existing systems [86].

Within low core voltage embedded systems common power saving methods such as DVFS are not feasible due to the voltage threshold of materials being approached in the drive to meet transistor count, therefore a requirement for

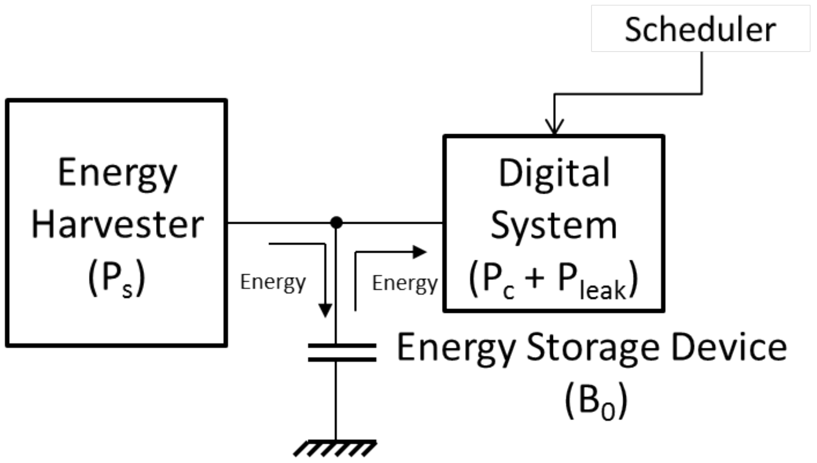

Scheduler
Energy Harvester (P_s)
Digital System ($P_c + P_{leak}$)
Energy
Energy
Energy Storage Device (B_0)

Figure 3.1: Energy Harvesting Model for system

job management and power scheduling across each harvesting cycle is necessary to maximize operational lifetime and successful job execution. An example of duty cycle alteration can be seen in Figure 3.2, where management pauses the processor during a low energy state to prevent potential failures. More on Energy Harvesting can be seen in Chapter 2.

3.2.2 This Work

The research presented in this chapter works at the system level, managing energy through a heuristic-based management scheme. This is based upon work by Koch and Moser [52], [70], where energy is collected and then consumed until a lower limit is reached and the effectiveness of the overall system recorded. For this, equation 2.3 is used. The models from this work identify important points in scheduling, such as when to suspend jobs and save power, or alter duty cycles so important work takes place during times of abundant energy.

The energy management models chosen for investigation were:

- No Management - System will run and take no action based on its energy readings
- Statistical Process Control (SPC) - System will monitor previous values for energy stored in B_0 at the end of a harvesting cycle and reduce the number of jobs executed if this is more than one standard deviation below the mean
- Trivial - A guard band system, where the processor is suspended if the recorded value of B_0 is below a pre-set limit
- Hybrid - A mixture of the SPC and Trivial examples above, where SPC will only be run if the value of B_0 is above the guard band limit

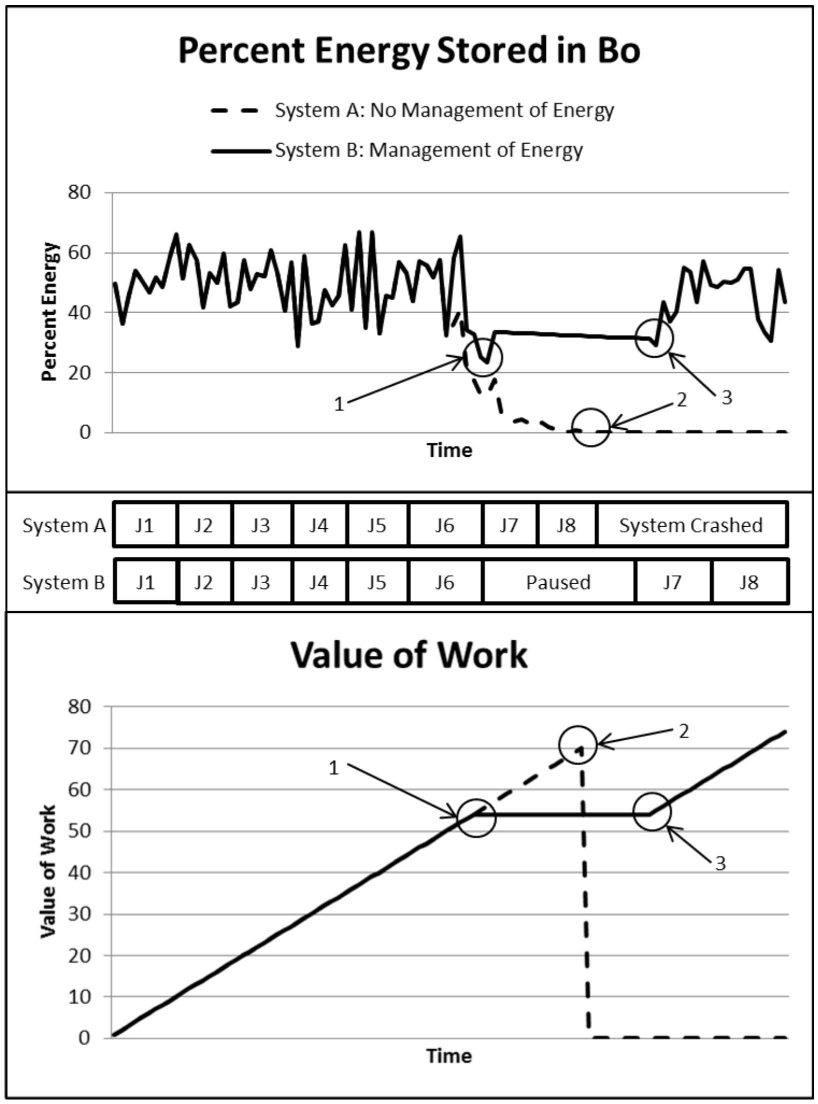


Figure 3.2: Energy Use vs. Value to the System.

Energy stored in an Energy Storage Device enters a famine state at (1), where System B suspends processing of jobs to allow energy to recover. System A continues with execution, causing failure at point (2), and the processor to crash - giving zero value to work. System B recovers at (3), continuing execution from its previous state.

Prior to all simulations undertaken, a power and sample size calculation took place in Minitab to ensure small differences could be identified. With an assumed standard deviation of 30 (based on data from the experiments in section 3.2.3), an α value of 0.05 (95% confidence no significance between values will be correctly identified) and a β of 0.8 (80% confidence significance between values will be correctly identified), 10000 samples was deemed sufficient to identify a Δ difference of 1.24 taking α and β into account. The values used for α and β are standard values used within six sigma for estimating Δ [16]. This means small changes in distributions due to variations in the input could be recognised with a high level of confidence and allow rapid progression through the experiments.

3.2.3 Baseline Simulations

Within this chapter, preliminary experiments look at how simple management affects the probability of successful completion and how this changes when shifting from a single-core to multi-core environment, as the embedded system market is rapidly shifting towards this architecture layout [110].

A key question is whether job or energy management is necessary. Under light loads, it is hypothesised that purely ad-hoc management could perform just as well as dynamic schedulers. Therefore, an initial experiment shall identify the effectiveness of ad-hoc management.

Simple management of jobs can be thought of based on a premise of travelling to a destination through several stages where the management of the journey (purchase of tickets) can be performed either ad-hoc (buying tickets at the point of onward travel) or in advance. Journey time, much like processor execution time, can vary. Eventually all time in the processor will be allocated (equivalent to tickets selling out). To test this, a simulation based on this ticket purchasing allegory was made. The rules of this simulation were as follows:

- Journey is made up of several stages
- The next stage can only take place if a ticket is available upon arrival
- Travel time is a random value
- All tickets are released simultaneously
- Tickets are consumed at a random rate
- If a ticket is not available, simulation ends

The simulation was run with the following variables:

- Tickets per step: 10-100
- Number of steps: 1-10
- Ticket Consumption per Round: 1-50

After 10000 repeats for each combination of inputs, the probability of successful completion was outputted to a CSV file for statistical analysis.

To follow on from this generic model of probability, core use was investigated using a Monte Carlo Simulation, as these have been widely used in design stages of work to verify correct operation and feasible operation [120]. Parameters for this were as follows:

- Jobs arrive at a random rate per second, with only one job arriving at a time
- Jobs are completed at a random rate per second
- If a job arrives and many cores are free, the lowest numbered core with no job will begin processing
- If a job arrives and one core is free, it will begin processing
- If a job arrives and no cores are free, it is placed in a queue

This code simulates a basic first-come-first-served (FCFS) multi-core scheduler with no job priority, similar to the ad-hoc situation in the previous experiment. The simulation was run with the following variables:

- Job arrival time (λ): 10-50
- Job completion rate per core (μ): 10-50
- Number of Cores: 1-10

After 72000 repeats for each combination of inputs to give confidence in the value of Δ, the probability of successful completion was outputted to a CSV file for statistical analysis, storing the average and maximum wait time, queue length and core idle time for each run.

3.2.4 System Level Simulation

Following on from these baseline experiments to determine effectiveness of a simple FCFS model, MATLAB was used to conduct System Level Simulations with the additional consideration of energy. An algorithm was developed, which can be seen in Algorithm 1, to determine key parameters and values through large introductory simulations. Within this algorithm, mean values for P_s, P_c and P_{leak} in equation 2.3 are initially set; along with a value for B_0^{Max}. The values for P_s, P_c and P_{leak} are multiplied by a Gaussian random number generator with Mean=0.5 and Standard Deviation=0.2 to give distributions, which are then applied to equation (2.3) with B_0 not allowed to exceed its value of B_0^{Max}. If the value of energy (E) is at or below zero, the simulation is deemed to have run out of available energy, meaning the system would fail and the simulation ends. However, if energy is above zero, then B_0 stores this value and the simulation repeats for another harvesting cycle.

The simulation repeats for 10000 harvesting cycles and with ten repeats to prove consistency of results. Results for each simulation are outputted to a CSV file for statistical analysis.

To determine the effectiveness of the simulation, tests were run with the following parameters:

- P_s=1-50
- P_c=1-50
- P_{leak}=1-5
- n=0.5-0.9
- B_0^{Max}=100-1000

```
for $repeat$ = $1:10$ do
    for $P_c$ = $5:50$ do
        $P_s$=50, $P_{leak}$=3, n=0.8;
        for $B_0^{Max}$ = $50:50:2000$ do
            while $Clock < 100000$ do
                if $B_0 > B_0^{Max}$ then
                    $B_0$=$B_0^{Max}$;
                end
                $P_s$=random number*Psrate;
                $P_c$=random number*Pcrate;
                $P_{leak}$=random number*Pleakrate;
                SDPs=StDev all $P_s$;
                MeanPs=Mean all $P_s$;
                if $B_0 < MeanPs-(3*SDPs)$ then
                    $P_c$=1;
                end
                if $B_0 < MeanPs-(2*SDPs)$ then
                    $P_c$=0.5*$P_c$;
                end
                if $B_0 < MeanPs-(SDPs)$ then
                    $P_c$=0.8*$P_c$;
                end
                S1=n*($P_s$-$P_c$);
                S2=($P_c$-$P_s$);
                if $S1 < 0$ then
                    S1=0;
                end
                if $S2 < 0$ then
                    S2=0;
                end
                Power=$B_0$-S1-S2-$P_{leak}$;
                if $Power <= 0$ then
                    Break;
                end
                $B_0$=Power;
                Clock++;
            end
            Output Clock;
        end
    end
end
```

Algorithm 1: Algorithm for Energy Harvesting Investigation

3.2.5 Design of Experiments

This preliminary work showed effects of varying all inputs, with changes primarily seen from altering P_s, P_c and B_0^{Max} through passive data observation (see section 2.4.4). Therefore, a Design of Experiments (DoE) was conducted to determine each variables overall contribution and whether any interaction between variables took place. Once insignificant higher-orders were removed, a minimised model was found, which can then be mathematically analysed by the General Linear Model (GLM) to give the percentage contribution (ϵ) that each parameter, including error, delivers to the overall system [67]. The DoE was set up in Minitab as a two-level, five input, full factorial DoE with five repeats, giving 160 data points (5x2^5). Parameters were as follows:

- P_s=20/50
- P_c=20/40
- P_{leak}=2/5
- n=0.5/0.8
- B_0^{Max}=100/1000

Results from the DoE were analysed using Minitab and reduced to allow investigation using the General Linear Model, giving contributory effects and suitable ratios of P_s, P_c and B_0^{Max} for successful operation in a harvesting environment that has a Gaussian output profile. In these, energy is centred around a mean (μ) with the distribution of points set by the standard deviation (σ). A larger standard deviation means points will be further from the mean on either side, which could cause energy supply issues for a circuit powered by the harvester.

3.2.6 Statistical Process Control of Energy

Once the DoE was complete, simulations using Statistical Process Control was carried out to manage jobs based on how the current energy value relates to previous levels. For a Gaussian model, the output should never move more than three standard deviations from the mean. If this occurs, then it can be assumed that the process has changed either through alteration in the harvesters' ability to generate energy or from the embedded system increasing its energy demands, meaning action must be taken to return the output to control. This is a key criterion of Statistical Process Control (SPC) key to monitoring an output [117]. To conduct this, the algorithm was run three further times:

- With no energy management, for a baseline
- With P_s monitored and actions taken off this
- With B_0 monitored and actions taken off this

When Energy Management was activated, the value of P_{leak} was increased from 5 to 12, simulating the increased energy requirements of the energy management circuit. The goal of these tests was to identify if a simple check on a 32 point mean and standard deviation could identify when a system was approaching a low energy state and take action accordingly. A 32 point mean was chosen due to being able to use a five-bit counter within a hardware design as well as being well above the minimum sample size needed to identify a one sigma drift in a distribution. For this, the following rules were applied:

- If point was more than one standard deviation below the mean, 20% power reduction took place
- If point was more than two standard deviations below the mean, 50% power reduction took place
- If point was more than three standard deviations below the mean, all jobs were suspended to allow the famine state to recover

Following collection of the data for these tests, statistical analysis took place to determine whether an improvement in the number of successful clock cycles had taken place. Since the majority of this data was found to be non-Gaussian, the Kruskal-Wallis test was used for 3-parameter analysis, with Mann-Whitney for 2-parameters. These tests are accepted within six-sigma analysis as suitable for these kinds of data, providing variance is found to be equal for all parameters [10]. Capability Studies were also used to give understanding of how well a method spotted a low energy state to maximise execution.

From these tests, a concern raised was the possibility of a trivial management system performing at parity or superseding the effects of the SPC system for a greater saving in power and area. Therefore, the above experiment was repeated with all lines in algorithm 1 activated to compare results from having no management, SPC Management or Trivial Management. The trivial system merely looks at the value of B_0 and takes action based on this as a guard band system. If B_0 is below a level, the processor is suspended to allow recovery, replicating the three standard deviation example above. It was also theorised that a Hybrid mixture of these two methods would give better results than either SPC or a Trivial Management system alone, even accounting for the extra energy required. Therefore, a further experiment was conducted where the Trivial system was employed first, followed by the SPC system. If the Trivial system found an issue, it would suspend operation before the high consumption SPC controller was activated.

3.3 Results

This section gives the results of tests details in the Methodology regarding Budgeting, Core Use and Energy Management. These results are discussed along with the execution of further experiments from their findings. Within the Energy Harvesting sections, "Clock" refers to the number of successful harvesting cycles taking place before system failure.

3.3.1 Energy Budgeting

Figure 3.3 shows the probability of successful completion decreases as the number of steps in the journey or consumption per round increases. This gives evidence that for complicated systems, ad-hoc management is not suitable

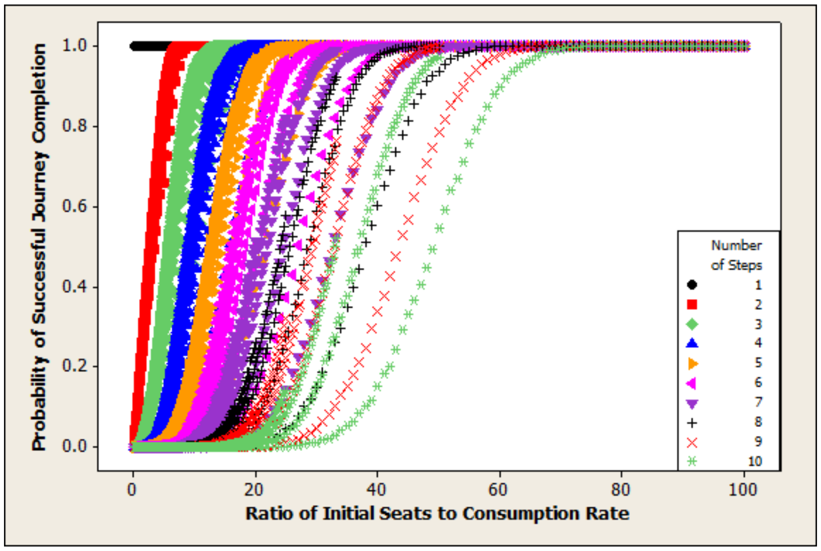


Figure 3.3: Probability a simulation will complete successfully based on the number of steps and the ticket consumption ratio

and managing power will give better performance, even accounting for the area/power overhead required for a circuit to perform this analysis. When the data in Figure. 3.3 is broken down further, a strong correlation to a linear increase (R-squared of 94%) can be drawn when the simulation was run for a fixed number of steps. Therefore, a system that knows the number of jobs in its schedule queue could interpolate a possible consumption for the next round of operation, compare this to the stored energy available and take appropriate action if there is found to be deficient energy in the storage device *apriori*. This proves the need for schedule management in complex multi-core systems, and will be explored further in Chapters 4 and 5.

3.3.2 Job/Core Use Simulations

Figure 3.5 shows that the addition of a pause when a core has completed execution balances out the workload across all cores. Without this step, even for high arrival rates, higher-numbered cores do not have a comparable usage percentage to lower-numbered cores, as illustrated in Figure 3.4. Over the life of a microprocessor, this could lead to reliability issues; especially with the lower numbered cores [10]. This method could give a simple solution to the management of core usage, by the simple addition of a wait command at the end of execution. However, this is only a preliminary example; therefore these tests were repeated with jobs having precedence constraints to ensure related jobs are not dispatched to cores with large separation. This work also does not consider the bus loading and issues with items such as L1 and L2 Cache accesses, which are a main consumer of time and energy within a microprocessor architecture [95].

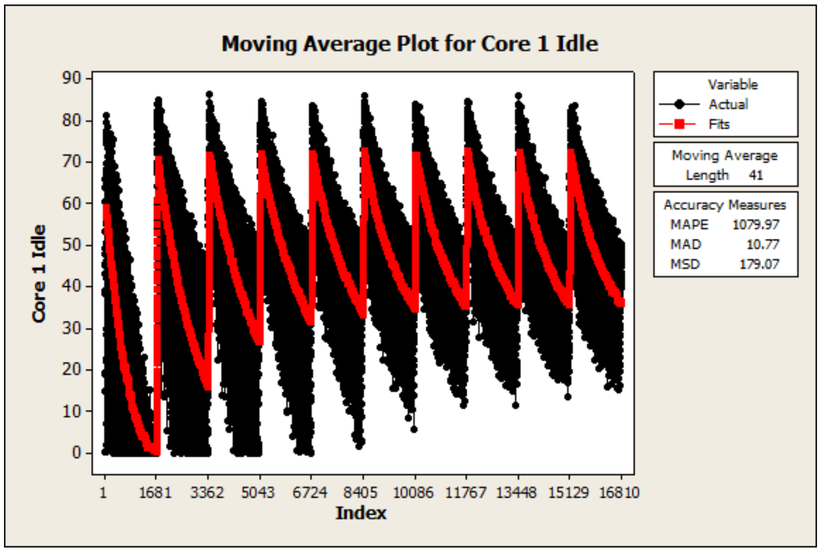


Figure 3.4: Idle percentage for Core 1 of a 10 core system as the Job Arrival and Completion Rate Increase

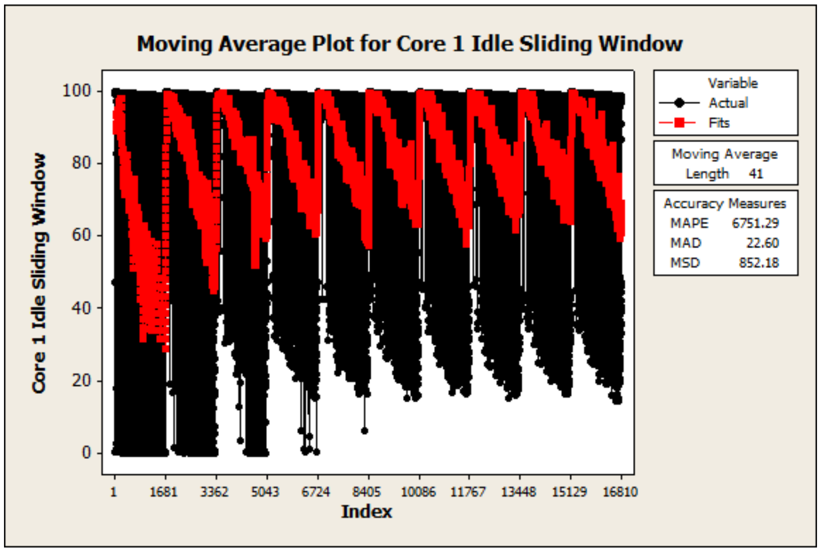


Figure 3.5: Idle Percentage, but with a Pause after Completion Added to balance out overall consumption rates

A simulation using models presented in [38] with 10% of jobs having priority over all other jobs by entering a fast track showed no statistical increase in job execution time for non-priority jobs, with a significant decrease for priority jobs. However, this is still modelled from a FCFS design, meaning lower numbered cores have greater workloads. A possible method presented by [38] of having a lead token that passes between cores on a regular basis could give a solution to this issue. However, this reiterates the importance of schedule management, and alternate methods of scheduling jobs will be investigated in the subsequent chapters.

3.3.3 Energy Harvesting Model

A key finding of the graphs in Figure 3.6 - Figure 3.9 is the prevalence B_0^{Max} has on the number of successfully completed cycles. For low values of storage, it is highly unlikely that a system will complete the simulation successfully. Experimental data shows that a value approximately 30-times greater than the generation rate per round is needed to have any chance of successfully running for an extended period. These tests also show that even when this is the case, due to the effect of harvester efficiency and leakage, consumption must only be approximately half of generation, otherwise it has a marked effect on the reduction of availability. Though it must be noted that the effects of leakage and efficiency taken on their own seem slight compared to the consumption rate.

Capability Studies

As can be seen in Figure 3.10 to Figure 3.11, the use of B_0 for monitoring is significantly more effective than the use of P_s. As the energy arriving at the system can be thought of as stochastic, the observations for P_s show no trend in the Last 25 Observations and the overall data is left skewed; whereas the data for B_0 displays an overall trend in the Last 25 Observations and an element of normality within the normal probability plot, skewed by the maximum value for B_0 being capped at 1000. Cp and Cpk for the systems varies also, with B_0 improving Cp by a factor of 1.61 and Cpk by 5.86. Therefore, the monitoring of the energy stored within the storage device, as well as being easier to practically measure than the energy delivered by the device, also gives significantly better capability and a longer lifetime of operation ($n = 9104$ for B_0 and 1859 for P_s).

3.3.4 Design of Experiments

As can be seen from Figure. 3.12 — Figure. 3.14, the expected availability of the system decreases consistently when the energy consumption rate increases. The DoE identifies several levels of significance, including a fourth order between Generation, Consumption, Efficiency and Bo. This means that all permutations must be considered from first to fourth order when run through the GLM. Table 3.1 shows clearly that for the design of experiments, key contributors to the outcome were generation and consumption rates, along with the amount of stored energy available. These alone give a total contribution score of 45.33%. Also noteworthy is the second and third order interactions of generation with consumption, generation with storage, consumption with storage and generation with consumption and storage. When all these factors are considered, they

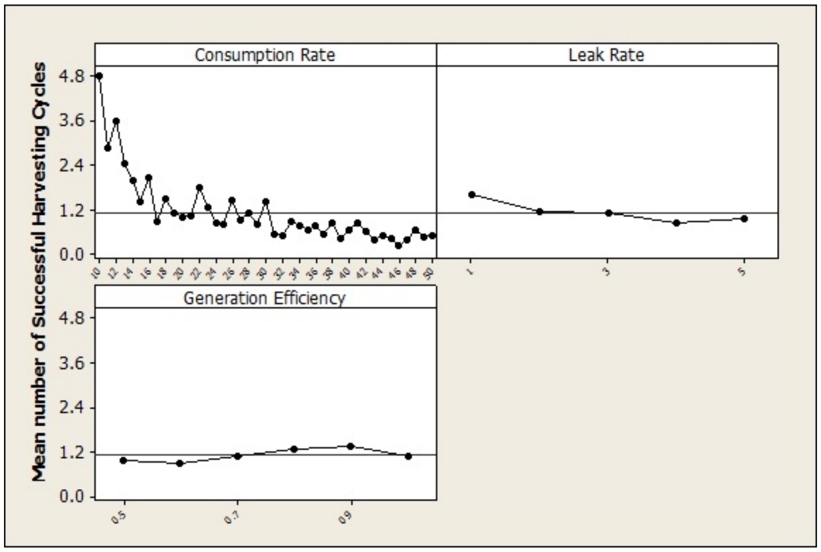


Figure 3.6: Number of Clock Cycles successfully completed for a Generation Rate of 10 and a B_0^{Max} of 100 as Consumption Rate increases

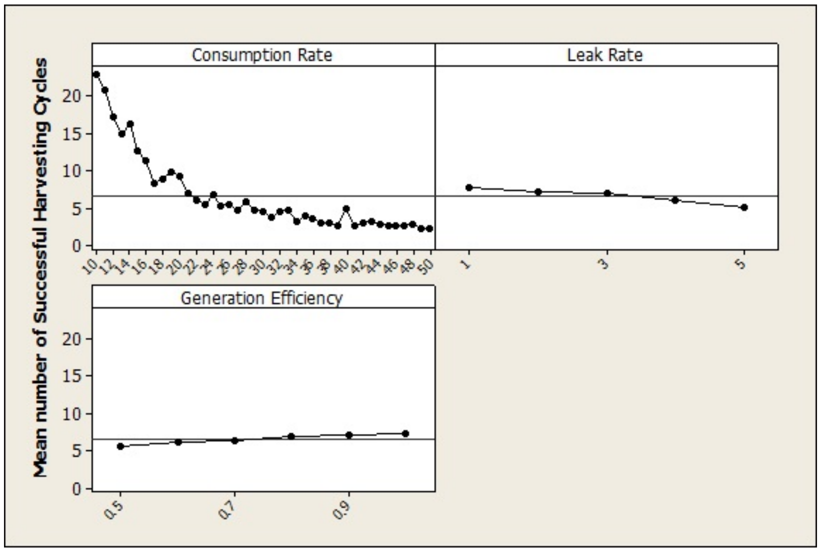


Figure 3.7: Number of Clock Cycles successfully completed for a Generation Rate of 10 and a B_0^{Max} of 1000 as Consumption Rate increases

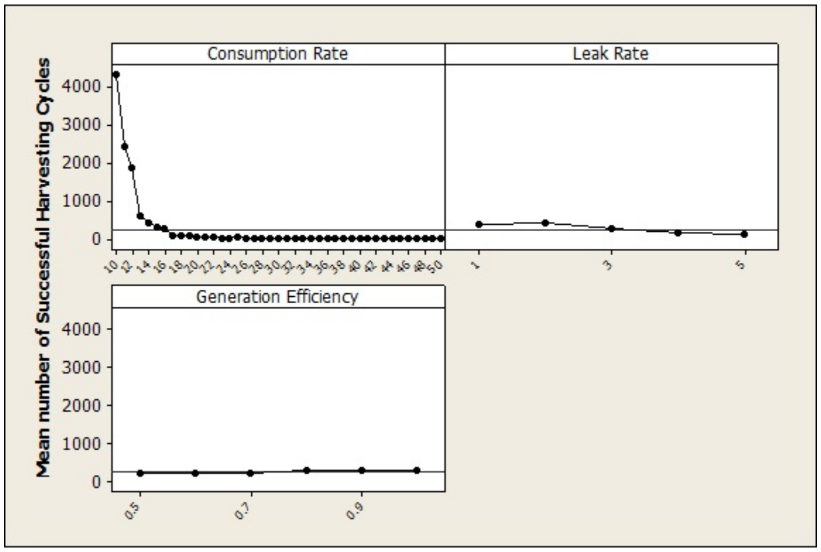


Figure 3.8: Number of Clock Cycles successfully completed for a Generation Rate of 50 and a B_0^{Max} of 100 as Consumption Rate increases

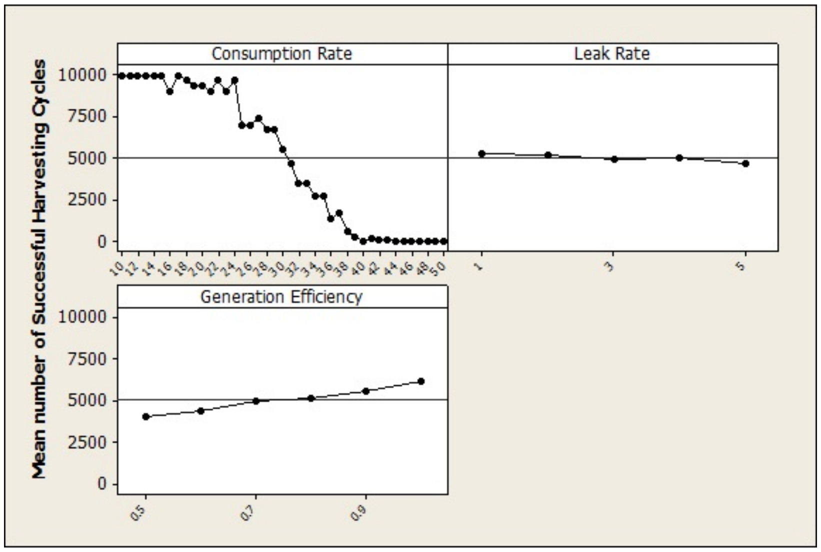


Figure 3.9: Number of Clock Cycles successfully completed for a Generation Rate of 50 and a B_0^{Max} of 1000 as Consumption Rate increases

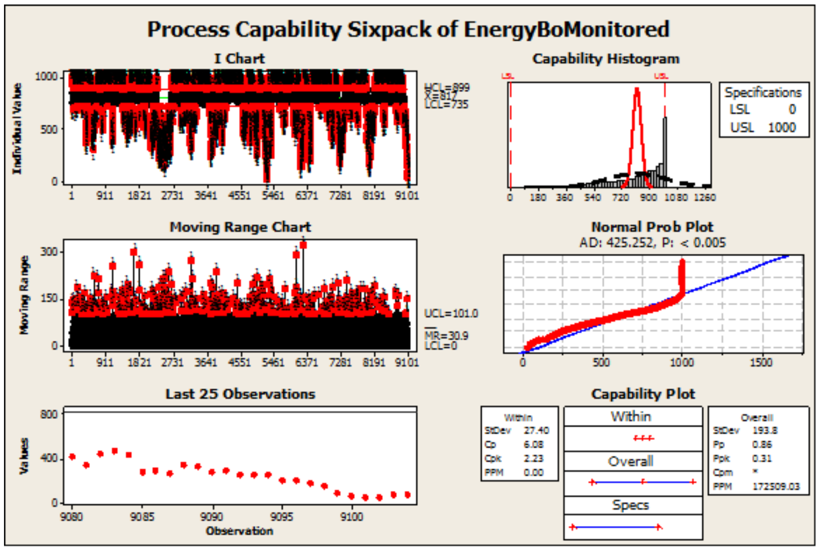


Figure 3.10: Capability of SPC System when B_0 is monitored

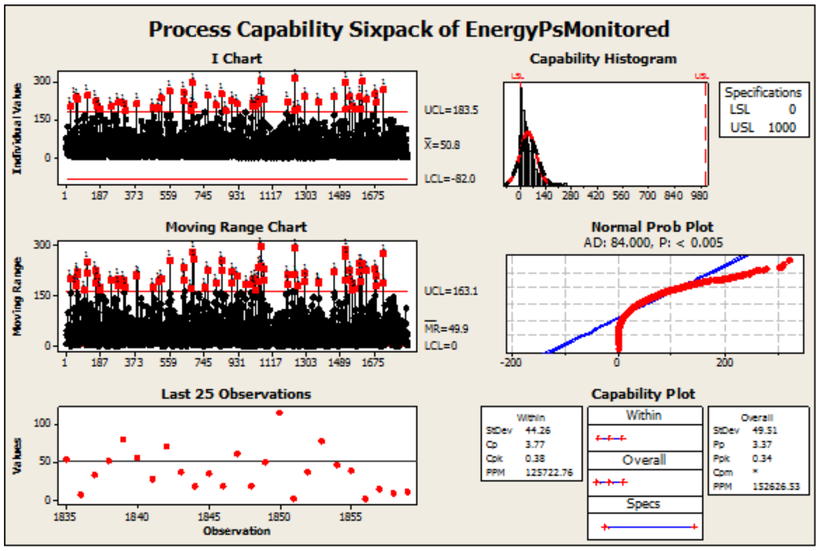


Figure 3.11: Capability of SPC System when P_s is monitored

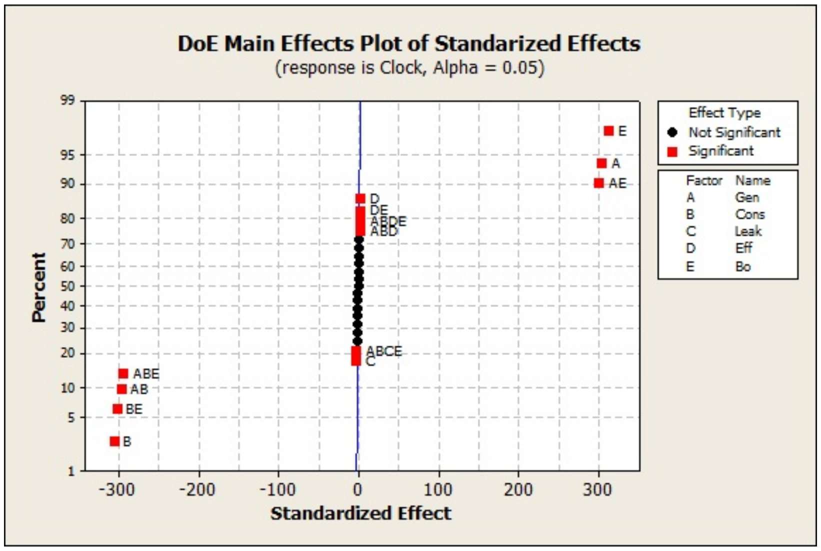


Figure 3.12: Main Effects Plot of DoE
This shows A, B, C, D, AB, AE, BE, ABE, ABE and ABDE have a Statistically Significant Effect

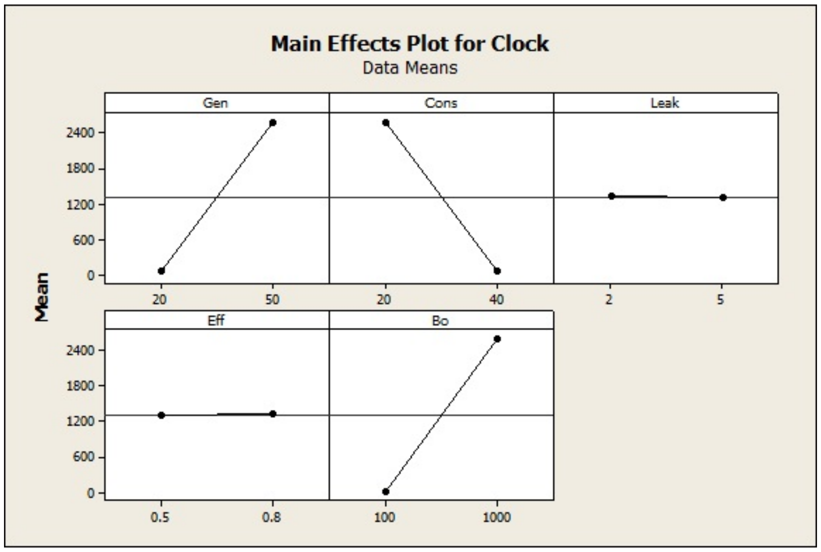


Figure 3.13: Main Effects Plot for DoE
This shows Generation Rate, Consumption Rate and B_0^{Max} have main effects on Mean Clock Cycle

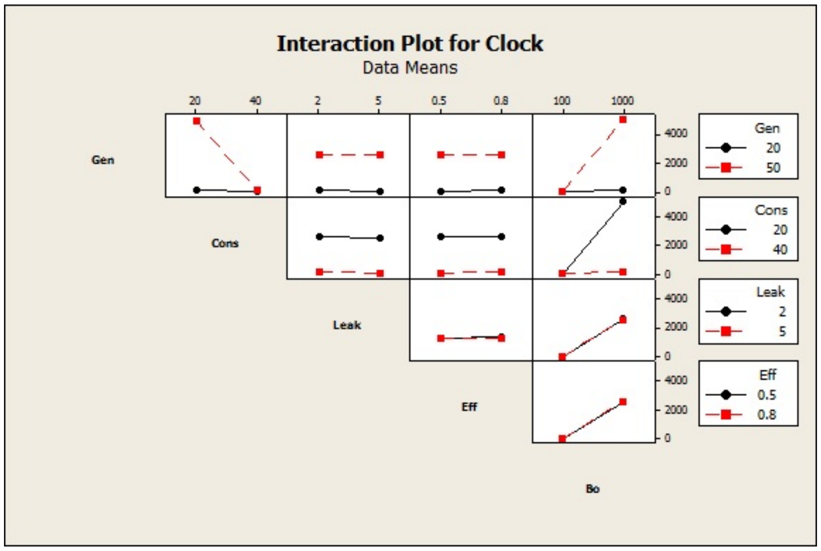


Figure 3.14: Interactions Plot for DoE, showing no interactions

account for 99.97% of the available contribution. The remaining 0.03% contains all other factors and interactions, with error only accounting for 0.02% . However, it must be noted that this experiment is a simulation, which would account for these strong contributions. Tests with a real-life harvester and embedded system will give a greater percentage error, but with main contributors to ϵ still being P_s, P_c and B_0^{Max}.

3.3.5 Statistical Process Control of Energy

When these tests were initially run, no significance was found. However, by altering the algorithm to only run the SPC tests when the value of energy in the storage device was below the mean value, power consumed by the SPC module was considered to be reduced and several results became statistically significant. This proves the feasibility of using SPC within the power management module and its beneficial effects to system availability.

As can be seen in Table 3.5, SPC Management gives a 2.5x improvement in number of completed harvesting cycles compared to no management. Its B10 life (time at which 10% of units under test would be expected to fail) increases from 2119 cycles to 5244, giving a statistically significant improvement in unit lifetime. As P_{leak} values are lowered further, the number of successful cycles increases further. Therefore, if the SPC system can be designed efficiently and minimising its power consumption relative to the main processor, it can be concluded that significant increases in Availability could be achieved.

Table 3.1: Epsilon Scores for General Linear Model Following DoE

Source	**P-Value**	ϵ
Gen	0.000	14.475
Cons	0.000	14.533
Leak	0.004	0.001
Eff	0.007	0.001
B_0^{Max}	0.000	15.328
Gen*Cons	0.000	13.738
Gen*Leak	0.949	0.000
Gen*Eff	0.709	0.000
Gen*B_0^{Max}	0.000	14.196
Cons*Leak	0.368	0.000
Cons*Eff	0.750	0.000
Cons*B_0^{Max}	0.000	14.204
Leak*Eff	0.360	0.000
Leak*B_0^{Max}	0.096	0.000
Eff*B_0^{Max}	0.009	0.001
Gen*Cons*Leak	0.055	0.001
Gen*Cons*Eff	0.016	0.001
Gen*Cons*B_0^{Max}	0.000	13.498
Gen*Leak*Eff	0.651	0.000
Gen*Leak*B_0^{Max}	0.534	0.000
Gen*Eff*B_0^{Max}	0.583	0.000
Cons*Leak*Eff	0.450	0.000
Cons*Leak*B_0^{Max}	0.796	0.000
Cons*Eff*B_0^{Max}	0.796	0.000
Leak*Eff*B_0^{Max}	0.497	0.000
Gen*Cons*Eff*B_0^{Max}	0.011	0.001
Gen*Cons*Leak*B_0^{Max}	0.010	0.001
Error		0.021
Total		100.0

Table 3.2: Mann-Whitney Test Results for Monitoring B_0 or P_s for SPC Control

B_0^{Max}	P_c	**Monitoring Method**	**Median**	**P-Value**	**Conclusion**
2000	40	B_0	4867	0.0376	B_0 Outperforms
	40	P_s	1496		

3.3.6 Trivial Example

As can be seen from Table 3.4, the use of SPC gives better results than trivial management when B_0^{Max} is large. In cases where B_0^{Max} is low (below 500), trivial management gives better results due to the regularity with which B_0

Table 3.3: Mann-Whitney Test Results for Use of SPC Within Power Management

B_0^{Max}	P_c	**Management Method**	**Median**	**P-Value**	**Conclusion**
1000	35	None	762	0.00	SPC Outperforms
	35	SPC	1728		
1500	35	None	3460	0.00	SPC Outperforms
	35	SPC	9329		
	40	None	440.5	0.00	SPC Outperforms
	40	SPC	699.5		
	45	None	169.5	0.00	SPC Outperforms
	45	SPC	253.5		
	50	None	120.5	0.027	SPC Outperforms
	50	SPC	135.5		
2000	35	None	13886	0.00	SPC Outperforms
	35	SPC	38982		
	40	None	534	0.01	SPC Outperforms
	40	SPC	667.5		
	45	None	231	0.00	SPC Outperforms
	45	SPC	285.5		
	50	None	163	0.00	SPC Outperforms
	50	SPC	217.5		

will be below the limits set in the algorithm, therefore meaning management of power is more frequent.

3.3.7 Hybrid Example

To compensate for these cases, a hybrid approach that manages both was tested, but found to not give better results. This is most likely due to the extra power required to run both tests, which is irrelevant at low or high energy storage. This section of the system will be revisited at a later date to further improve the possible design feasibility, as this management technique has potential for systems with a small energy reserve and predictable energy delivery, such as solar cells.

Table 3.4: Mann-Whitney Test Results for Use of SPC and Trivial Systems Within Power Management

B_0^{Max}	P_c	**Management Method**	**Median**	**P-Value**	**Conclusion**
500	45	None	38	0.01	Trivial Outperforms
	45	SPC	32.5		
	45	Trivial	55.0		
	50	None	38	0.00	Trivial Outperforms
	50	SPC	24		
	50	Trivial	55.0		
1000	35	None	762	0.00	SPC Outperforms
	35	SPC	1729		
	35	Trivial	697		
1500	35	None	3460	0.00	SPC Outperforms
	35	SPC	9329		
	35	Trivial	2770		
	40	None	441	0.00	SPC Outperforms
	40	SPC	700		
	40	Trivial	424		
	45	None	170	0.00	SPC Outperforms
	45	SPC	254		
	45	Trivial	182		
	50	None	120.5	0.02	SPC Outperforms
	50	SPC	135.5		
	50	Trivial	118.5		
2000	35	None	13886	0.00	SPC Outperforms
	35	SPC	38982		
	35	Trivial	6984		
	40	None	769	0.01	SPC Outperforms
	40	SPC	1020		
	40	Trivial	676		
	45	None	231	0.01	SPC Outperforms
	45	SPC	286		
	45	Trivial	243		
	50	None	163	0.00	SPC Outperforms
	50	SPC	217.5		
	50	Trivial	172.5		

3.4 Conclusion

This work shows that a model of a purely Gaussian energy harvester could be created and managed successfully, with an increase in availability evident for a suitable harvester. The experiments prove that a system can run for an extended period without management when the following conditions are met:

- Mean energy generation rate $P_s > 2P_c$ mean energy consumption rate (sufficient powerful harvester)

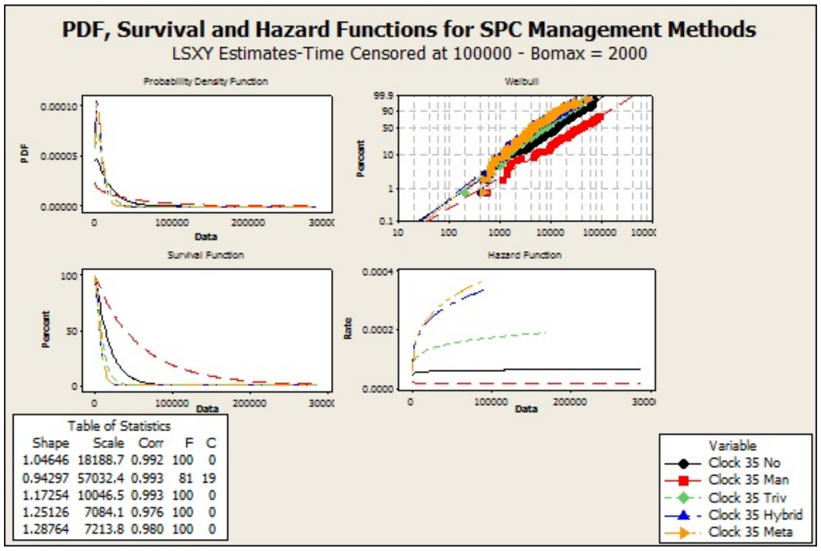


Figure 3.15: Reliability Analysis of No Management, Trivial Management, SPC and Hybrid Systems, showing predicted Failure Rate

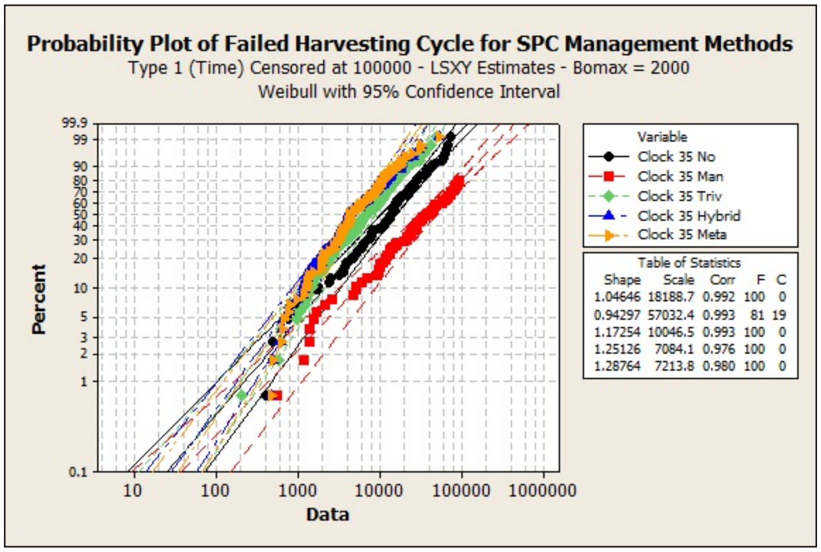


Figure 3.16: Probability Plot for various management techniques, showing predicted number of clock cycles before failure

- Maximum value of storage $B_0^{Max} > 20P_s$ mean energy generation rate (suitably large energy storage device)

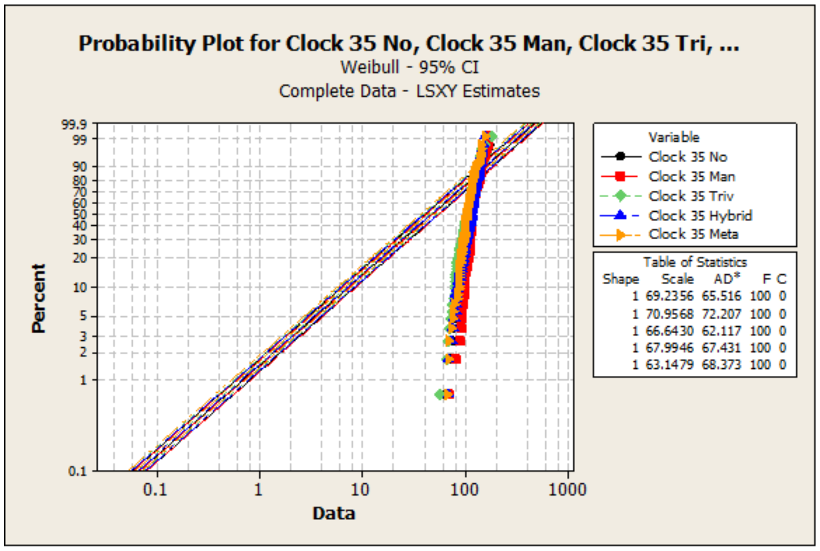


Table of Statistics

Shape	Scale	AD*	F	C
1	69.2356	65.516	100	0
1	70.9568	72.207	100	0
1	66.6430	62.117	100	0
1	67.9946	67.431	100	0
1	63.1479	68.373	100	0

Figure 3.17: Probability Plot for various management techniques with a Uniform Distribution, showing predicted number of clock cycles before failure

Table 3.5: Failure Times for Various Management Techniques with B_0^{Max}=2000, P_s=50, P_c=35

Management Method	B10 Life	B50 Life	B90 Life
None	2118	12814	40359
SPC	5244	38665	138116
Trivial	1474	7350	20461
Full Hybrid	1173	5285	13796
Meta Hybrid	1257	5427	13787

Tests with alternate harvesting environments such as uniform input distributions, which are common with harvesters such as a thermoelectric system [4], also show similar results to the Gaussian environment, with $P_s > 2P_c$ and a large value of B_0 required to give significant operating times as can be seen in Figure 3.17.

The use of SPC as an energy management scheme has a beneficial effect on successful runtime within an Autonomous Hybrid Embedded System, giving up to 2.5 times the successful life time compared to no management. Trivial systems do outperform for a low value of B_0^{Max} ($< 20P_s$ in these experiments) and would be useful in Autonomous Systems where no battery is present to supplement the harvester.

While these increases do not mean the system will run indefinitely, the increase in operational lifetime is beneficial for items that cannot easily be maintained, either through inaccessibility or volume of devices. Improved operation for minimal area penalty is highly desirable in these systems, therefore giving this simple system significant impact on the embedded market.

Work in Chapter 6 revisits this work with more complicated jobs and schedules, including those with precedence and priority, such as the EDF scheduler and other examples from [18], as well as tests using the Arduino embeded device. While construction of the system in VHDL and placement into an FPGA or other device was considered, it was deemed unnecessary for preliminary investigations as use of an established device could give rapid implementation and allow the concept to be proved as feasible.

Chapter 4

Use of Game Theory as a Real-Time Scheduler

Who in the world am I? Ah, that's the great puzzle.
Lewis Carroll, Alice's Adventures in Wonderland

4.1 Introduction

Though embedded systems have existed for many years, only recently with the advent of low dollar per square-inch silicon costs and high processing power has their use has become ubiquitous in our day to day lives. The failure of an embedded device can have outcomes varying from a minor inconvenience such as a slow internet connection to life-threatening issues including failure to apply anti-lock controls to wheel brakes on a car or other vehicle.

While the increase in processing power has led to more utilisation, this comes hand in hand with higher processor utilisation. Systems take on extraneous jobs outside of their core remit, which can cause key jobs to be missed while the kernel deals with these lower priority jobs. Several methods of scheduling exist to cope with this, but could be thought of as inadequate for high-complexity systems. Therefore, a new paradigm in scheduling must be created that can adequately cope with unpredictable systems and the sources of non-determinism that drive these. One possible method for this, which has already been shown to have success within fields such as communication, is Game Theory. Through work identified in Section 2.2, this has been shown to have effective outcomes within complex systems. As such, investigation for real-time systems, both single and multi-core, is of great interest and thought of as critical for future system development.

4.1.1 Real-Time System Scheduling

For this chapter, we consider the scheduling of jobs within real-time systems, the background for which can be found in Section 2.1.2. A design such as the one proposed, with varying levels of criticality for jobs is common in embedded system design. In the unit under test, there are two main jobs with parameters as follows:

- Job 1: Hard Real Time, Variable Activation and Computation Time, Variable Deadline, Aperiodic activation, $100\mu s$ WCET
- Job 2: Soft Real Time, Variable Activation Time, Set Computation Time and Deadline, Aperiodic Activation, $10\mu s$ WCET

With two such contrasting jobs, several already existing algorithms are not capable of delivering a feasible schedule. Under light loads, First Come First Served (FCFS) will manage acceptably, but will fail if the arrival rate of either job increases beyond a functional limit. This is also the case for Earliest Deadline First (EDF) and Rate Monotonic (RM), which has already been shown in [30] .

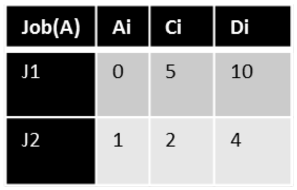
Iideally a system should be deterministic, but in this case jobs can be considered non-deterministic with arrivals not know *apriori*. Algorithms that use exhaustive search such as Bratley's Algorithm for EDF [18] therefore cannot deliver a feasible solution as a job tree cannot be created. Due to this, a more flexible system that can deliver a schedule from a full tree if one exists but can also manage a dynamic online schedule is thought to be of great use in the current and future embedded system market.

This Work

While Heuristic Algorithms have been found to give improvements for simple systems with a small number of job types, as the number of categories increases the effort required to create this heuristic would proliferate exponentially, eventually reaching a point where creating a heuristic for all possibilities would be infeasible.

Cooperative vs. Non-cooperative The work in this chapter is based on a non-cooperative game, though the use of cooperative games has been discussed as a possible method of getting more ideal results [62] [81]. This is not without issue, as the implementation of cooperative games is significantly more complicated than the non-cooperative types and is not considered during this work, which is in the early phases of its development. Wooldridge points out in [119] that a system must be designed with sufficient incentive for players and be simple enough to allow rational choices to be made, providing assumptions set in the design have been adequately developed.

The algorithm in this chapter uses a heuristic based system, similar to the Spring Kernel [105] . This contains a function H that sets to an input parameter to give flexible scheduling. When $H = a_i \equiv FCFS$ and when $H = D_i \equiv EDF$, making Spring a useful paradigm. However this also requires knowledge of each jobs resource requirements to deliver a determined arrival time. The Game Theoretic Algorithm differs as this bases execution on perceived value, which can be fixed in design or dynamically altered by the system. Currently the system is designed as a $1|mixed, nopreem|L_{max}$, meaning it is capable of managing aperiodic and periodic jobs together on a uniprocessor machine to minimise lateness. Figure 4.1 shows a schedule that cannot be created under EDF as it reacts to the arrival of J1 at t=0, blocking J2 from completion and causing an overrun at t=4. This figure also shows the Game Theory solution to the problem, which waits to execute J1 based on the lazy scheduling theory, allowing J2 to arrive and be scheduled.



Job(A)	Ai	Ci	Di
J1	0	5	10
J2	1	2	4

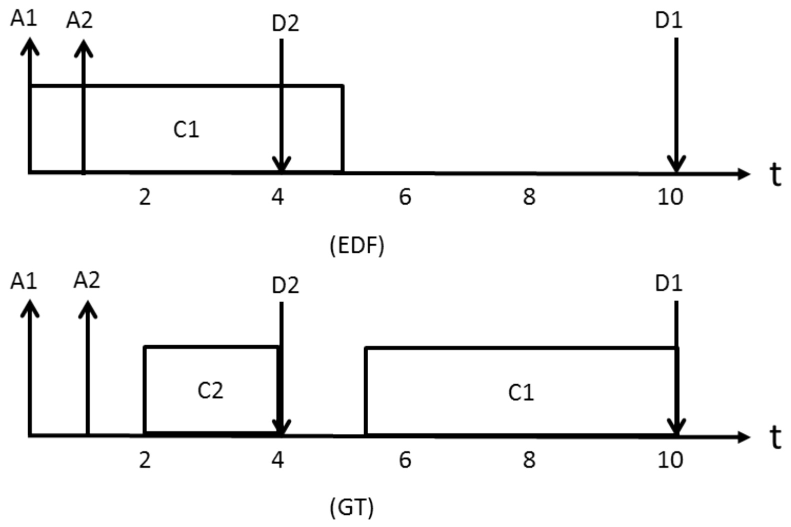
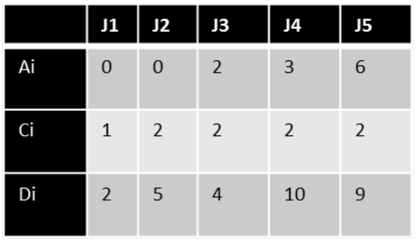


Figure 4.1: A Schedule Feasible under Game Theory but Infeasible under EDF



	J1	J2	J3	J4	J5
Ai	0	0	2	3	6
Ci	1	2	2	2	2
Di	2	5	4	10	9

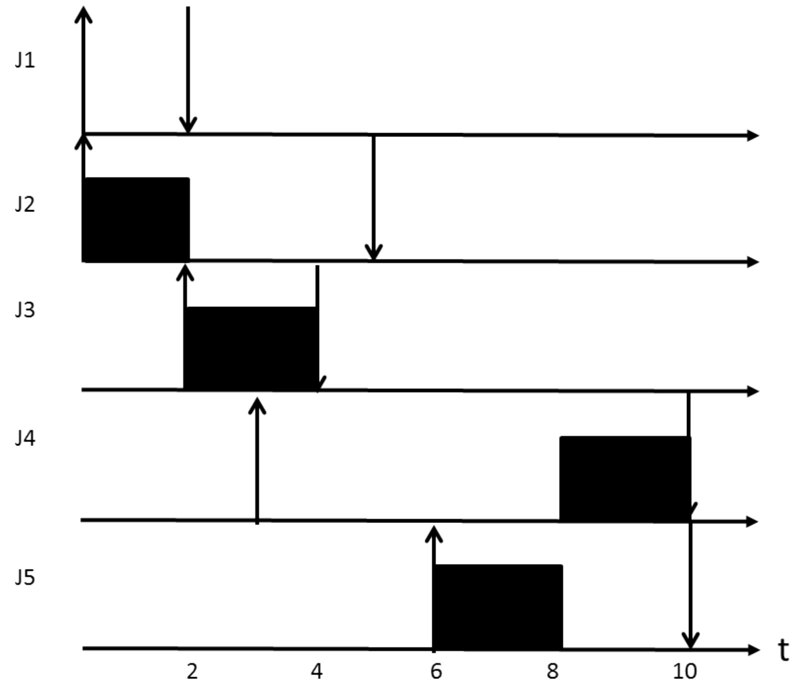


Figure 4.2: Example of Figure 2.4 without pre-emption using the Game Theory Algorithm with $J1 < J2$

For this, J1 is missed as it is deemed of lower importance than J2. The algorithm also differs at t=5 as J4 does not execute until after J5 as this is possible under the Lazy Scheduling caused by the algorithm.

Figure 4.2 is based on Figure 2.4, but with no pre-emption. This cannot be feasibly scheduled as $U_i > 1$ at time t=2, meaning that a conflict between J1 and J2 arises. Depending on the system design, two outcomes can take place:

1. At t=0, if $Priority\ J1 > Priority\ J2$ then J1 will execute, leaving J2 to wait until t=2 to begin, which in turn will cause J3 to be missed as no pre-emption is present
2. At t=0, if $Priority\ J1 < Priority\ J2$ then J2 will execute, causing J1 to be missed

The goal of the Game Theory scheduler is as follows: *In a mixed periodic and aperiodic system, the Game Theory Algorithm will successfully execute all hard jobs, including where Processor Utilisation is above 1; providing Utilisation of the subset of hard jobs is below 1.*

In addition, for a system of equal priorities, the algorithm will create a feasible schedule dynamically, should one exist.

4.2 Methodology

The Game Theoretic Scheduler overcomes issues caused by Heuristic Algorithms by scoring jobs based on their remaining execution time multiplied by a coefficient for choosing to either run the job at the particular instant or to wait. These coefficients can be either constant and set in the system design phase, or a variable that the algorithm can choose to alter based on factors such as processor load — giving a dynamic scheduler in a concise package. These scores are then placed in matrices, giving sets of two-player non-cooperative games. These can then be tested using Algorithm 2 and Algorithm 3 to find the most prudent outcomes for both jobs. Once this is complete, the Nash Equilibrium with the highest utility score for the highest priority job will execute. If no Nash Equilibrium is found, the most prudent decision for the highest priority job in the queue will be chosen. Job priority is set by the run/wait coefficients; thus removing the need for complicated systems such as multi-level scheduling.

An example of the execution can be seen in Figure 4.3 – Figure 4.4. Figure 4.3 shows a feasible schedule, where jobs activate and have enough time to complete execution prior to their deadline, meaning no issues occur. Figure 4.4 however shows a schedule that is infeasible due to the processor utilisation > 1. In this case, an abundance of short jobs have caused the hard real-time job to miss its deadline at 125 time units. The scheduler in this case is not aware of priority levels and therefore does not discern the difference between the hard and soft real-time jobs. Other scheduling algorithms such as multi-level queuing would prevent this miss by running the highest priority job first. Though it should be noted that doing this would cause all three of the short lower priority jobs to be missed, giving a significant detriment to the Quality of Service.

Figure 4.5 gives a compromised solution based on Game Theory. At 25 time units, the system scores whether to run Job 1 or Job 2 based on their benefit, calculated by Algorithm 2. The outcome significantly favours Job 1 executing, even though this means Job 2 will miss its deadline. This is similar to the work in [26] where a job increases its priority once the slack remaining in its execution time is zero, but with no need for priority levels. The level of laxity or importance of a job type is set during the system design by the establishment of values for the constants used to create the utility values. Algorithm 2 performs this through checking each job in the queue relative to its laxity $(Deadline(D_i) - (ComputationTime(C_i) + CurrentTime) > 0)$ and its laxity in the next time period $(CurrentTime + 1)$. This builds a two-person game matrix where each player has two options: to run or to wait. The constants chosen during construction of the matrix such as those in lines 8 and 28 force the matrix to favour Job 1 over Job 2 (Hard Real-Time over Soft Real-Time), allowing decisions regarding the future to be made *apriori* and prevent a missed critical job due to injection of a new lower priority job, thus avoiding issues such as the domino effect — where the introduction of a job causes all other job to miss their deadline by the cascade increase in C_i. This is depicted graphically in Figure 4.6, which is a Game Theory depiction of Figure 4.3 at times $t = 0$ and $t = 10$, with Prudence and Nash Equilibrium marked as in Figure 2.7. At $t = 10$ the Nash Equilibrium favours running Job 1 over Job 2, therefore this will take place. After this, there is only one job in the queue, meaning all Job 2 scores will be zero and the job will execute at the next available opportunity.

4.2.1 Initial Experiments

Following completion of the system designs in Algorithms 3 and 2, tests were conducted in MATLAB. This was used due to its ability to handle large matrices and mathematical manipulations, allowing rapid testing and debugging of the process under test. Monte Carlo simulations at a variety of Job 2 activation rates were run to determine the number of missed hard real-time jobs and the success rate of the system in finding Nash Equilibrium. Both Job 1 and Job 2 were determined as random arrival times under a Poisson process, with jobs computation times set by separate Poisson processes for Job 1 and Job 2. As can be seen in the job definitions above, Job 2 completes execution ten times faster than Job 1 and therefore can have a significantly higher activation rate. The runs from these Monte Carlo Simulations allowed the effectiveness of the algorithm to be assessed, as well as comparing an approach that always used the Game Theoretic Approach to one that uses FCFS in light load, similar to previous work in [30].

4.2.2 Design of Experiments

Following the initial tests and satisfaction that the algorithms ran successfully, a Design of Experiments was used to determine the main effects for the Game Theoretic Scheduler and optimise these. For this system, a two-level DoE was used, where each input has two possible values. Five inputs were selected based on knowledge of the system and these were tested through the DoE three times, giving 96 total runs. Again, MATLAB was used for this, completing the DoE quickly and in a CSV format, able to be analysed in Minitab. Parameters for the DoE were as follows:

- Job 2 Activation Rate: 300/500
- Job 2 Game Theory Score Coefficient: 10/20

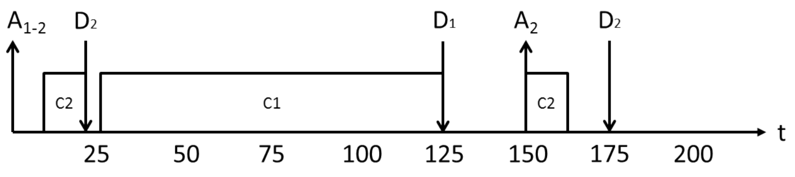


Figure 4.3: A Feasible Schedule under FCFS or EDF

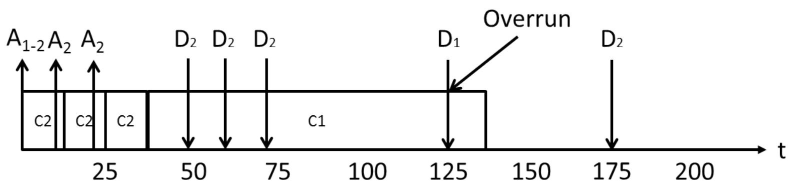


Figure 4.4: Due to several occurrences of Job 2, the schedule is now infeasible by EDF or RM

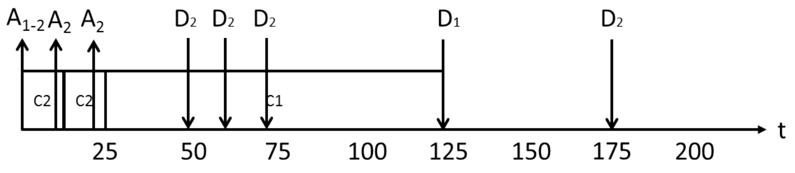
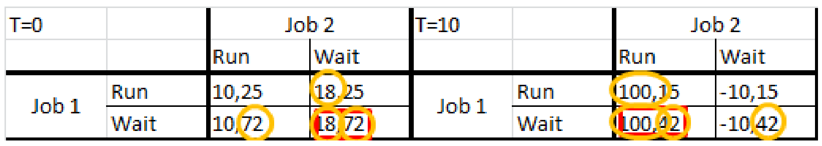


Figure 4.5: The System now runs successfully through removal of one Job 2, preserving Quality of Service



T=0		Job 2		T=10		Job 2	
		Run	Wait			Run	Wait
Job 1	Run	10,25	18,25	Job 1	Run	100,15	-10,15
	Wait	10,72	18,72		Wait	100,42	-10,42

Figure 4.6: Game Matrices for Figure 4.3 at $t = 0$ and $t = 10$, clearly showing the Nash Equilibrium as (Wait,Wait) and (Run,Wait) respectively.

```
for all $JobsinQueue$ do
    $Util1[Job] = D_i - (Clock + C_i)$
    $Util2[Job] = D_i - ((Clock + 1) + C_i)$
end for
for all $JobsinQueue$ do
    if $JobType == Job1$ then
        if $Util1[Job] > 0$ then
            $RunUtil[Job] = Util1[Job]$
        else if $Util1[Job] == 0$ then
            $RunUtil[Job] = 100000$
        else
            $RunUtil[Job] = -100000$
        end if
        if $Util2[Job] > 0$ then
            $WaitUtil[Job] = 3 * Util2[Job]$
        else if $Util2[Job] == 0$ then
            $WaitUtil[Job] = 100$
        else
            $WaitUtil[Job] = -100000$
        end if
    end if
    if $JobType == Job2$ then
        if $Util1[Job] > 0$ then
            $RunUtil[Job] = Util1[Job]$
        else if $Util1[Job] == 0$ then
            $RunUtil[Job] = 100$
        else
            $RunUtil[Job] = Util1[Job]$
        end if
        if $Util2[Job] > 0$ then
            $WaitUtil[Job] = 2 * Util2[Job]$
        else if $Util2[Job] == 0$ then
            $WaitUtil[Job] = 1$
        else
            $WaitUtil[Job] = -10$
        end if
    end if
end for
```

Algorithm 2: Scoring Algorithm to create Game Matrices

```
 1: $Temp1 = 1$
 2: $Temp2 = 1$
 3: $Prudence1[NumberofPlayer1Strategies]$
 4: $Prudence2[NumberofPlayer2Strategies]$
 5: **for all** $Rows$ **do**
 6:   **for all** $Column$ **do**
 7:     **if** $Player1Utility(Rows, Temp1) \leq Player1Utility(Rows, Columns)$
        **then**
 8:       $Prudence1[Rows] = Temp1$
 9:     **end if**
10:   **end for**
11: **end for**
12: **for all** $Columns$ **do**
13:   **for all** $Rows$ **do**
14:     **if** $Player2Utility(Prudence2, Columns) \leq$
        $Player1Utility(Row, Columns)$ **then**
15:       $Prudence2[Columns] = Temp2$
16:     **end if**
17:   **end for**
18: **end for**
19: **for all** Strategies **do**
20:   **if** $Prudence1 == Prudence2$ **then**
21:     **return** $Nash$
22:   **end if**
23: **end for**
```
Algorithm 3: Prudence (Lines 1 to 18) and Nash (Lines 19 to 23) Algorithm

- Job 1 Game Theory Score Coefficient: 10/20
- Job 2 Score Coefficient Accumulate on non-run: On/Off
- Queue Length to activate Game Theory Scheduler over FCFS: 20/40

The results were analysed to identify epsilon levels of contribution to the outcome and detect any interactions between factors.

4.2.3 Optimisation and Reliability Runs

Following the successful completion of the DoE, further simulation runs were undertaken to determine the optimum values for inputs identified as critical. Since Job 1 Execution is most critical, the runs concentrate on finding the ideal operating point for this to be maximised at the expense of other metrics and system run-time. The finalised algorithm was then tested with a variety of CAN Load rates to give an expected lifetime, which allows comparison to established and previously developed algorithms and delivery of metrics to determine effectiveness of this algorithm over others.

4.3 Results

4.3.1 Comparison of Results for Full Game Theoretic and Hybrid Runs

Execution time for 100000 Runs

- Full Game Theory: 121458 seconds
- Hybrid Game Theory: 118472 seconds
- Hybrid (Optimised value from [30]): 105171s

The use of a hybrid approach, when optimised, reduces runtime by 15 percent through the non-activation of Nash Equilibrium calculation in light loading conditions such as when one job is present. As calculation time is a key metric to be considered when designing a scheduler, this reduction can be considered significant if and only if the Quality of Service is preserved.

When compared using a two-sample T-Test, only Figure 4.9, Figure 4.10 and Figure 4.12 gave statistically significant differences between the full Game Theory implementation and the Hybrid approach. More activations of Job 2 by Nash or arrival with core idle occurred for the Full Game Theory implementation and more runs of FCFS for the Hybrid system.

For Figure 4.9 running the system in Hybrid gives a higher Median and Inter-Quartile Range. Therefore the use of a test to run FCFS over GT can be concluded to significantly increase the running of this aspect of the scheduler. While the missed jobs do not significantly change, the runtime is reduced by ten percent.

For Figure 4.10 Less CAN Jobs Nashes are found with the Hybrid solution, most are being taken up by the FCFS Scheduler.

For Figure 4.11 the outputs are statistically and graphically similar — therefore no increase in risk of missing Job 1 when running FCFS.

For Figure 4.12 Inter-Quartile Range of the Hybrid Simulation is larger, but with less outliers. Therefore, as with Figure 4.10, the FCFS Scheduler is reducing the number of CAN Jobs needing to queue.

These results from preliminary tests show that a hybrid approach gives some improvements over the full Game Theory implementation. As Figure 4.11 clearly shows, there is little difference between missed hard real-time jobs between approaches. This is confirmed by Table 4.1, which demonstrates no statistical difference between distributions or centres for full Game Theory and the Hybrid Approach ($P = 0.078$). Therefore, the goal of reducing runtime while preserving QoS has been achieved.

4.3.2 Design of Experiments

Job 1 Optimisation

A high level of interaction took place within this run, with a fourth order found to be significant, meaning all first, second and third orders must also be considered. Following placement into the General Linear Model in Table 4.2, the score applied to the utility calculations for Job 2 and Job 1 gives 21 percent of the contribution towards missed Job 1, with interactions involving Job 2 Load

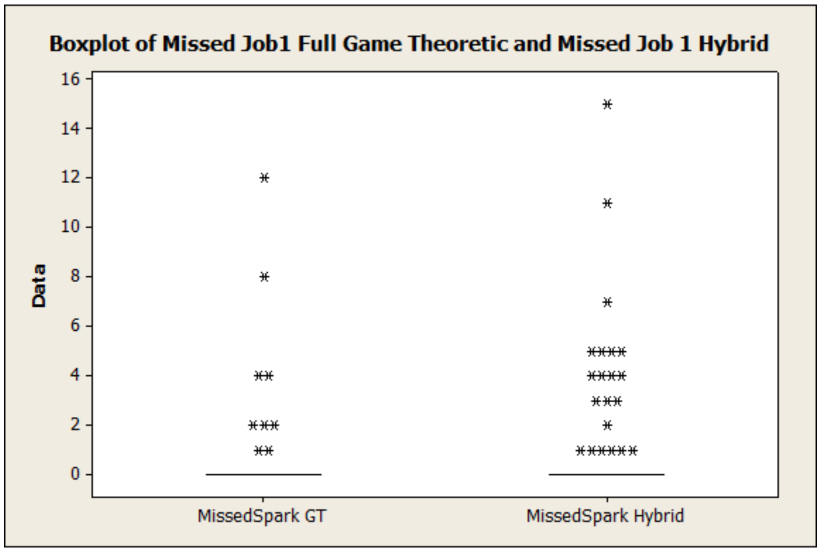


Figure 4.7: Box-plot showing rate of Missed Job 1 across Simulation Runs

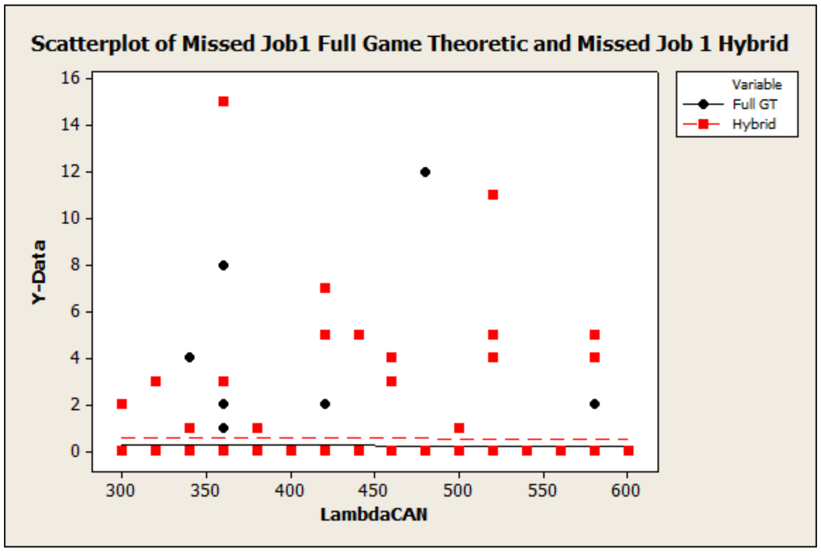


Figure 4.8: Scatter-plot showing missed Job 1 as Job 2 Load increases

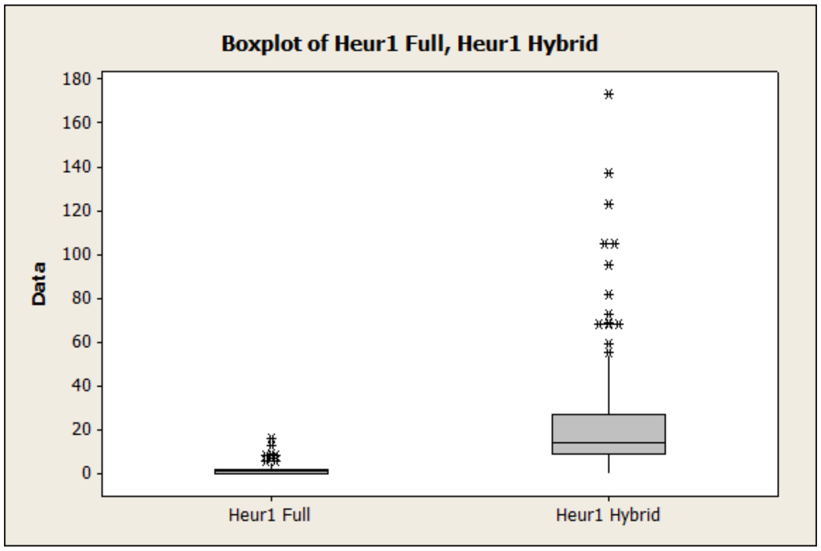


Figure 4.9: Box-plot showing rate of Activation for First Come First Served Scheduler

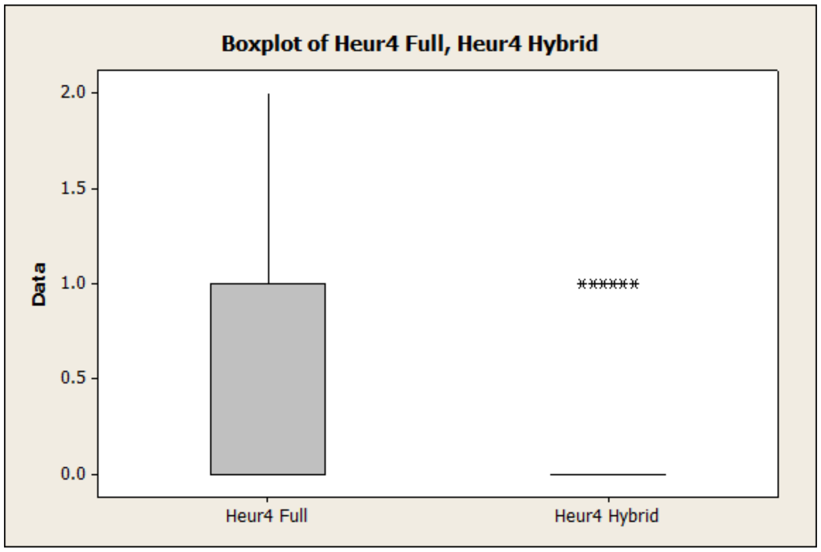


Figure 4.10: Box-plot showing rate of Activation for Job 2 Nash Equilibrium

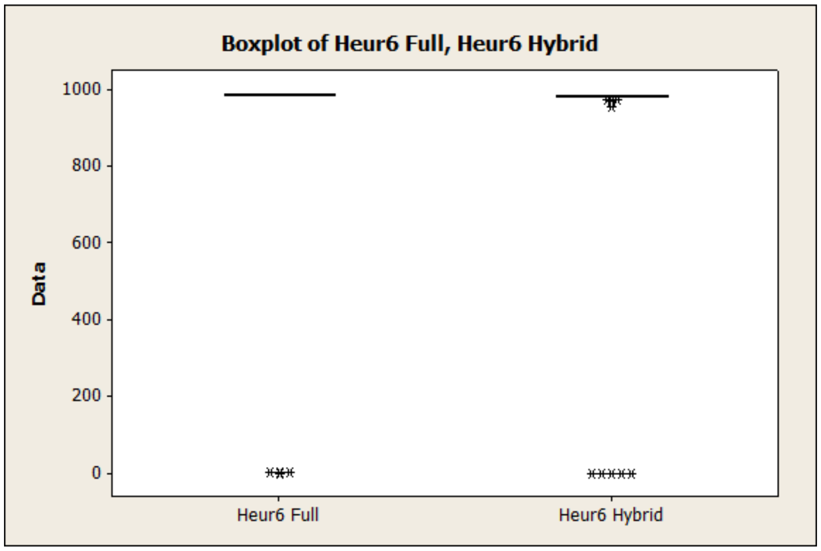


Figure 4.11: Box-plot showing rate of Activation for Prudent Job 1 Activation when no Nash Equilibrium can be found, but Job 1 must execute

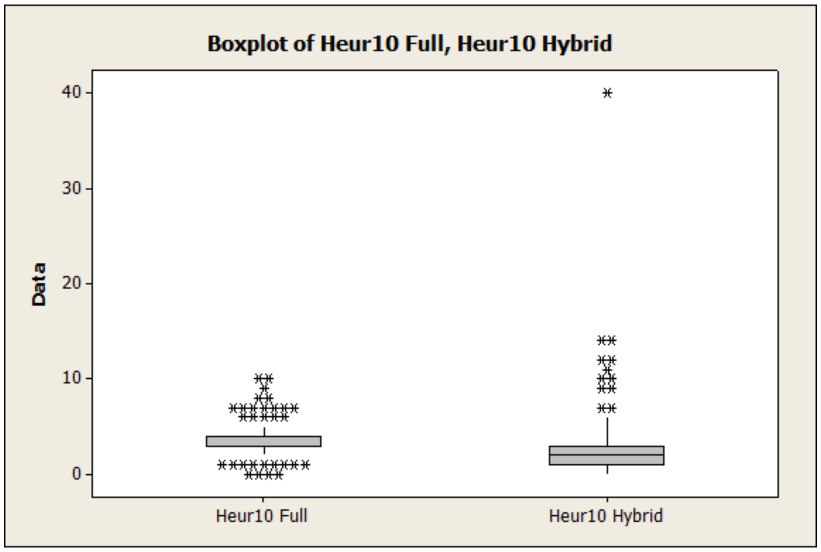


Figure 4.12: Box-plot showing rate of Activation for Job 2 Arrival when Core is Idle

Table 4.1: Two-Sample T-Test Results for rate of Missed Job 1 for Full Game Theory Implementation and Hybrid System

Source	**Mean**	**Standard Deviation**
Game Theory	0.23	124
Hybrid	0.54	1.86
P-Value		0.078

giving a further 15 percent. Figure 4.13 shows the score for Job 2 (CANScore) causes large changes in values, as does the value for Job 1 score (SparkScore).

When $Job2Score > Job1Score$, more Spark Jobs are missed, meaning $Job1Score > Job2Score$ wil prevent this. Some interactions are present in Figure 4.14, showing that increasing the Job 2 Score if no Nash Equilibriums are found causes more Job 1 to be missed if the Queue Length to run the Game Theoretic Algorithm is small. This is due to more Job 2 jobs being tested using Game Theory, which is also the case for the interaction between Queue Length and Job 2 Load and Job 2 Accumulation and Lambda.

From these results, we can determine that the design of the game is critical to its success. This was noted in Section 2.2.2. In the situation where $Job1Score \gg Job2Score$, Job 1 misses drop to near zero, meaning the fundamental design of the scheduler must be carefully considered at the concept phase of the system creation. By ensuring all jobs have appropriate scoring coefficients assigned, a multilevel equivalent system can be created with only one job queue. Accumulation factors may also be added to increase job priority, in a similar way that jobs in multilevel queues can be moved up by the scheduler if a number of rounds take place without service. Though to have this operate successfully, the size of the queue for activating the Game Theory Scheduler must be set accurately.

Table 4.2: Significant Epsilon Scores for General Linear Model Following DoE for Missed Job 1 Events

Source	**P-Value**	ϵ
Job 2 Score	0.004	7.222
Job 1 Score	0.005	7.062
Job 2 Score * Job 1 Score	0.004	7.193
Job 2 Accumulator * Job 2 Rate	0.037	3.744
Job 2 Score * Job 2 Accumulator * Job 2 Rate	0.039	3.683
Job 1 Score * Job 2 Accumulator * Job 2 Rate	0.038	3.706
Job 1 Score * Job 2 Score * Job 2 Accumulator * Job 2 Rate	0.038	3.712
Error		57.244
Total		93.566

Table 4.3: Significant Epsilon Scores for General Linear Model Following DoE for Prudent Job 1 Activation

Source	**P-Value**	ϵ
Queue Length to Activate Game Theory	0.000	14.045
Job 2 Rate	0.000	27.155
Queue Length * Job 2 Rate	0.000	11.225
Job 2 Score * Job 2 Accumulator	0.042	2.010
Queue Length * Job 2 Score * Job 1 Score	0.031	2.279
Error		35.843
Total		92.557

FCFS Optimisation

One third order interaction was found to be significant on this system. When tested in the General Linear Model, 52 percent of contribution to this section of the algorithm running came from Queue Length or Job 2 Load as can be seen in Table 4.3. Since this section was optimised in the work presented in Chapter 5, this experiment shows that the settings carry over from simple heuristics to the Game Theoretic Algorithm. The absence of interactions in Figure 4.16 and major contribution from Job 2 Loading seen in Figure 4.15 clearly demonstrate this.

To successfully use the hybrid algorithm, system designers must therefore consider the types of job and their loading to the device under worst-case settings. Statistical modelling could be used for this, with the possibility of automating this work through the creation of a design tool for creation of systems with the Game Theory Algorithm. For a commercial version of this algorithm,

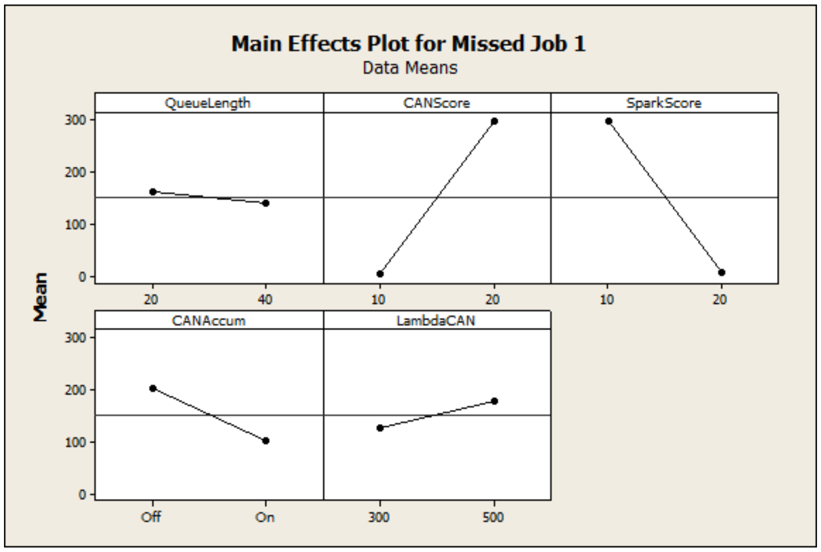


Figure 4.13: Main Effects Plot for inputs to DoE with respect to Missed Job 1

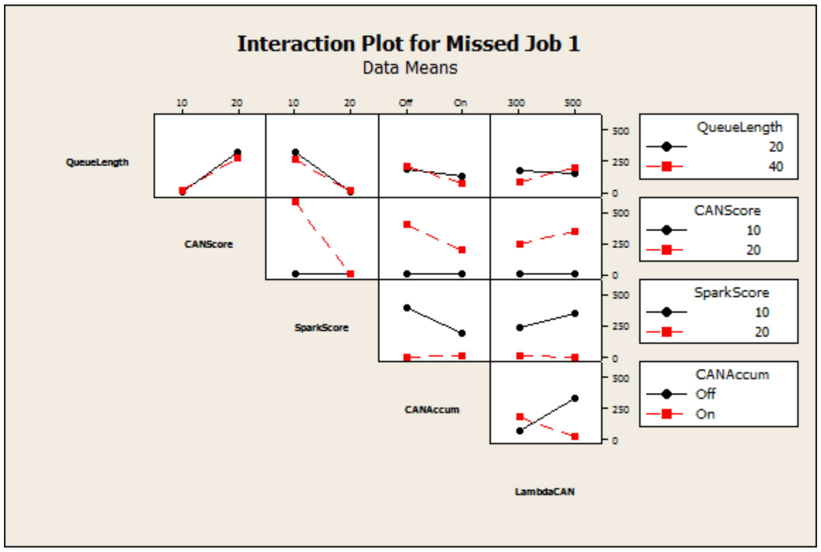


Figure 4.14: Interactions Plot for Missed Job 1

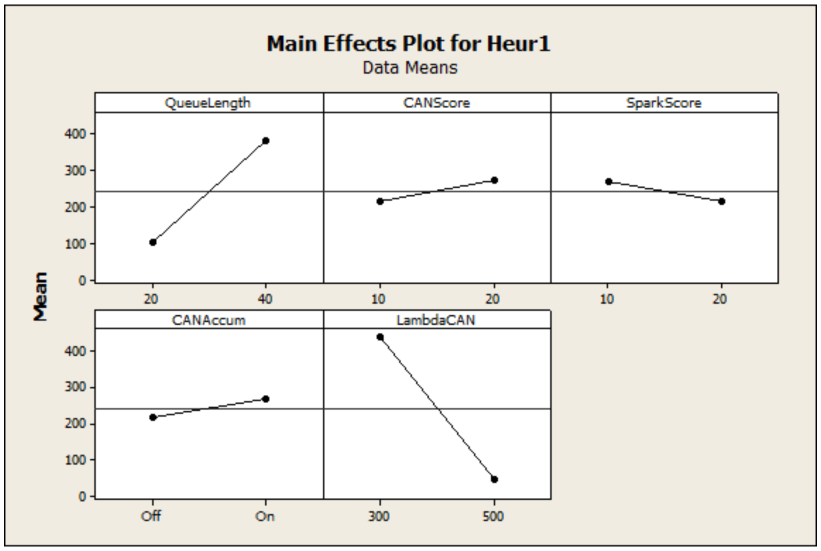


Figure 4.15: Main Effects Plot for inputs to DoE with respect to FCFS Activation

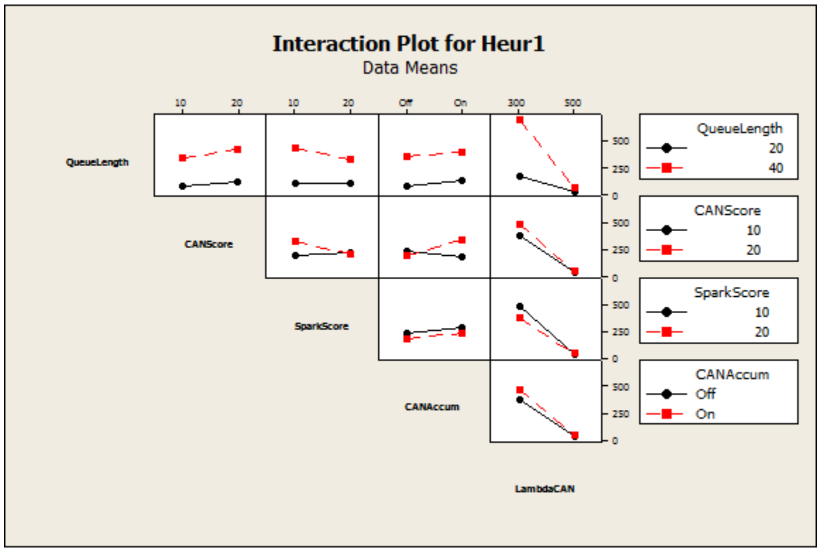


Figure 4.16: Interactions Plot for FCFS Activation

Table 4.4: Significant Epsilon Scores for General Linear Model Following DoE for Prudent Job 1 Activation

Source	**P-Value**	ϵ
Queue Length to Activate Game Theory	0.029	2.368
Job 2 Score	0.000	10.012
Job 1 Score	0.000	9.327
Job 2 Rate	0.000	14.075
Queue Length * Job 2 Rate	0.032	2.293
Job 2 Score * Job 1 Score	0.000	12.686
Queue Length * Job 2 Score * Job 1 Score	0.010	3.363
Error		38.027
Total		92.151

such a design feature is considered highly desirable and a possible area of future research.

Wait, Job 2 Nash and Job 2 when Core is Idle optimisation

All three of these sections within the Game Theoretic Algorithm had the same common factors. As with the Spark Optimisation above, these were the scores for Job 2 and Job 1 utility, giving over 36 percent of the contribution for all three algorithms within the General Linear Model. Again as with Spark, if $Job2Score > Job1Score$, more Spark Jobs are missed, proving that $Job1Score > Job2Score$ for the system to work successfully. Therefore, these areas of the algorithm can be considered to be a corollary of the optimisation work undertaken to minimise missed Job 1 and the queue length to activate the Game Theory algorithm.

Prudent Job 1 Optimisation

Again a third order interaction was present, with Job 2 Load and Job 2 and Job 1 utility delivering 46 percent of the contribution to changes in this output. As with the above heuristics, $Job1Score > Job2Score$ for successful system operation, as the interaction between these can be seen in Figure 4.18, once again meaning optimisation of Job 1 and FCFS also optimise this area of the algorithm.

4.3.3 Reliability Assessment

As the graphs in Figure 4.19 to Figure 4.22 show, there is little difference in the results for FCFS and the Hybrid Game Theory Algorithm at light Job 2 Load. This is due to the Hybrid Approach using FCFS at light loads and therefore only occasionally activating the Game Theoretic System. As Load increased, shown in Figure 4.20, the difference becomes more significant, with an improvement in B50 life (the point at which 50 percent of tests fail) of a factor of 32 percent

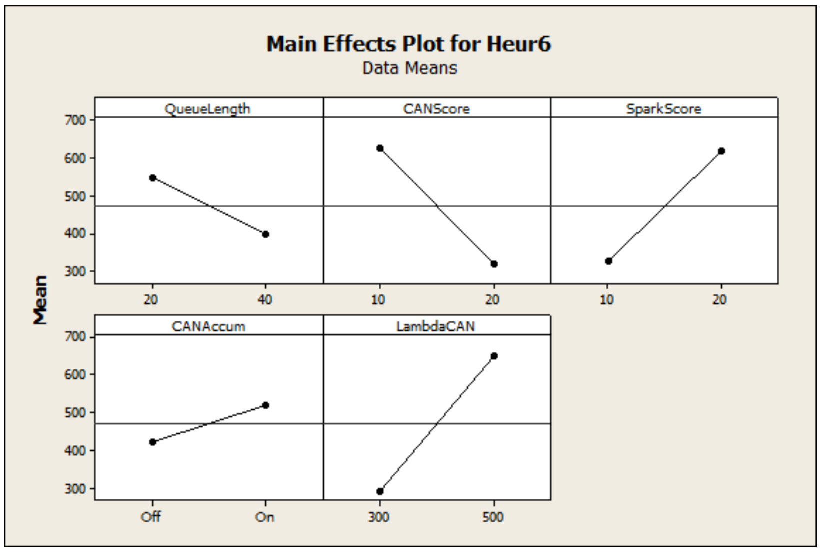


Figure 4.17: Main Effects Plot for inputs to DoE with respect to running Job 1 Prudently

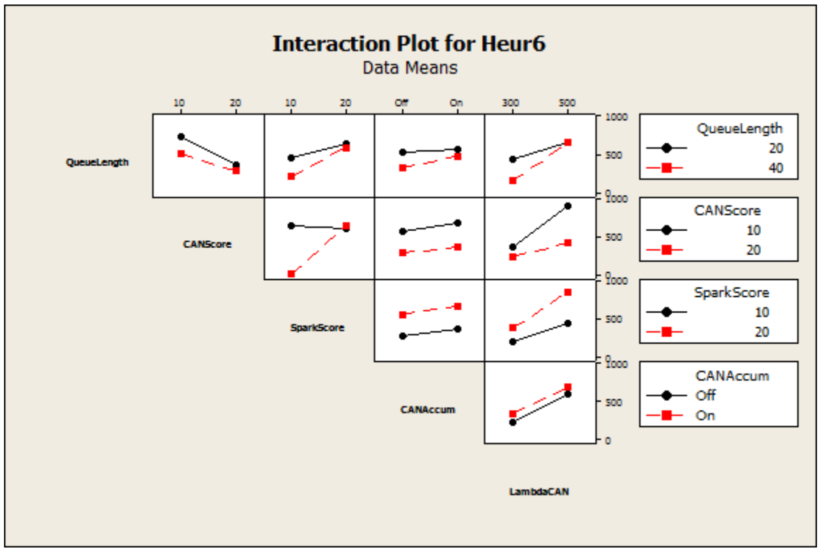


Figure 4.18: Interactions Plot for Prudent Job 1 Activation

Table 4.5: Mann-Whitney Results for number of successful cycles completed before a missed Job 1 occurs for FCFS and the Hybrid Game Theory Algorithm

Low Job 2 Load	
Source	**Median**
Game Theory	2784.5
FCFS	2941.5
P-Value	0.8738
High Job 2 Load	
Source	**Median**
Game Theory	1078.0
FCFS	1106.5
P-Value	0.2061

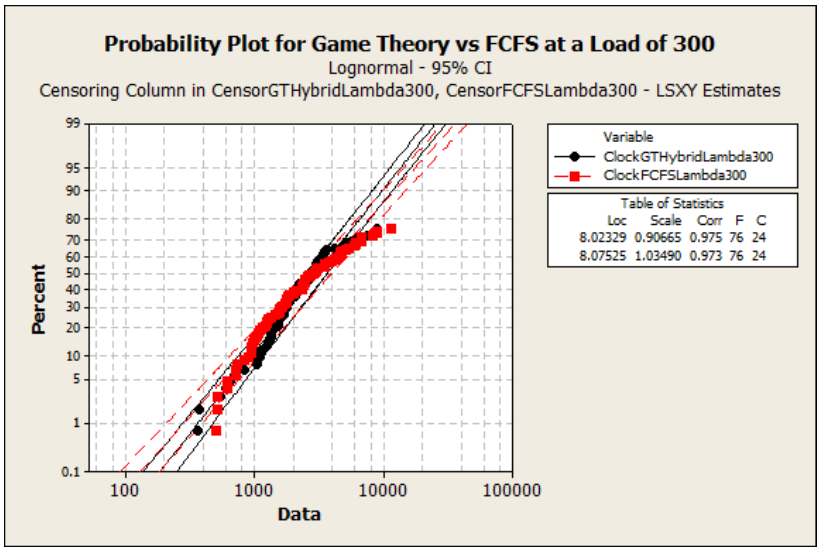


Table of Statistics

	Loc	Scale	Corr	F	C
●	8.02329	0.90665	0.975	76	24
🟥	8.07525	1.03490	0.973	76	24

Figure 4.19: Reliability Plot for Low Job 2 Load

and a B90 improvement of 92 percent. A run using solely the Game Theory Algorithm gave an improvement over the Hybrid Approach at high loads, but a detriment to performance at low loads. When the lifetimes were placed into a probability plot in Figure 4.19, a consistent 27 percent improvement of B10, B50 and B90 lives were observed against a 15 percent increase in run time; meaning the trade-off of run time against availability was considered fair.

Figs 4.21 and 4.22 also shows this increase in life, but include Probability Density Function, which shows the higher clustering of jobs at an early life for FCFS at high Job 2 load; meaning the system effective lifetime is shorter with

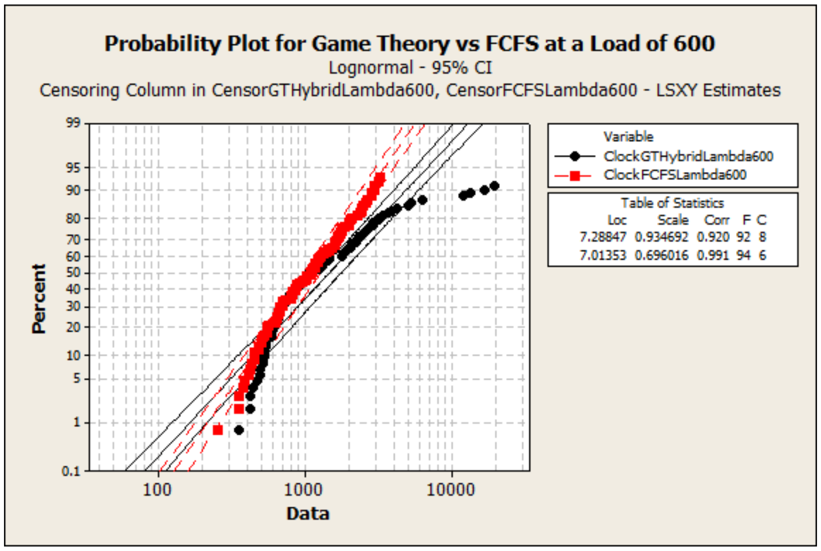


Table of Statistics

Loc	Scale	Corr	F	C
7.28847	0.934692	0.920	92	8
7.01353	0.696016	0.991	94	6

Figure 4.20: Reliability Plot for High Job 2 Load

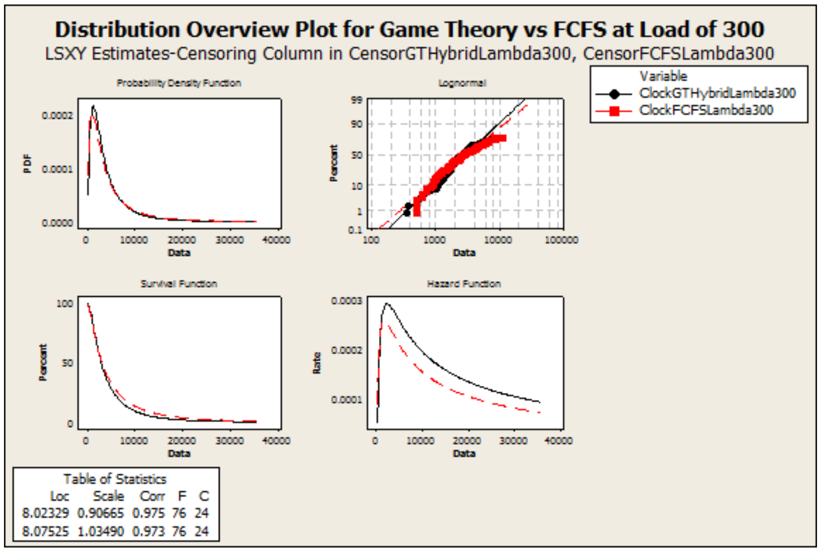


Table of Statistics

Loc	Scale	Corr	F	C
8.02329	0.90665	0.975	76	24
8.07525	1.03490	0.973	76	24

Figure 4.21: Graphs of Probability Density Function, Hazard Function and Survival Function for Low Job 2 Load

FCFS over the Game Theory Scheduler. This is also reflected in the Survival and Hazard functions, which demonstrate the Game Theory Scheduler is capable of reducing the peak failure rate and allowing higher job loads to be adequately managed.

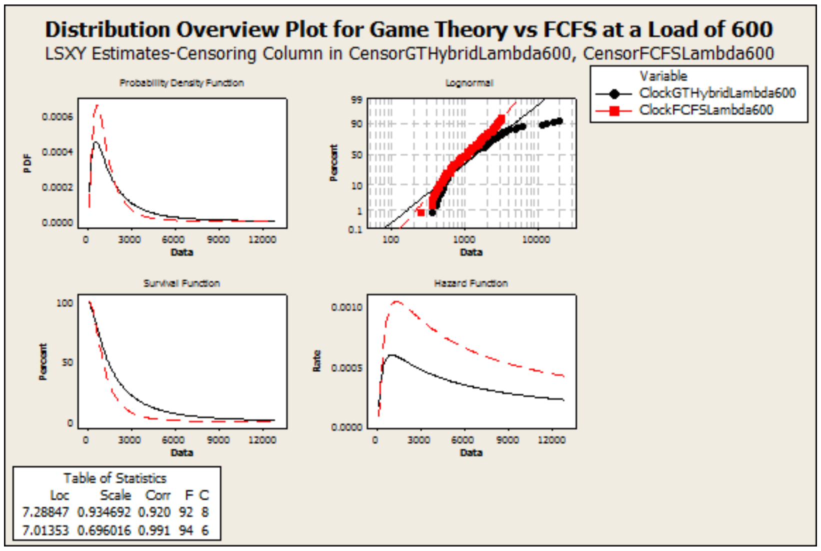


Table of Statistics

Loc	Scale	Corr	F	C
7.28847	0.934692	0.920	92	8
7.01353	0.696016	0.991	94	6

Figure 4.22: Graphs of Probability Density Function, Hazard Function and Survival Function for High Job 2 Load

4.4 Conclusion

The Game Theoretic Algorithm significantly reduces the misses of Job 1 in the two benchmark tests described. This improvement in system availability comes at the penalty of increased run time. By using a first come first served system during periods of light load, the availability is conserved while appreciably reducing the execution time. By the use of a design of experiments, the system is able to be optimised quickly and deliver high processor availability during simulation runs. In this current form, the system does not use pre-emption, but such an addition would simply require an addition to Algorithm 2 that executes whenever a new job arrives and the processor is occupied.

Chapter 5

Heuristic and Real-Time Scheduling

There is no greater folly than to be very inquisitive and laborious to find out the causes of such a phenomenon as never had an existence, and therefore men ought to be cautions and to be fully assured of the truth of the effect before they venture to explicate the cause.
John Webster, The Displaying of Supposed Witchcraft

5.1 Introduction

Chapters 3 and 4 prove the viability of using Statistical Methods and rule-based systems to improve availability and Quality of Service for embedded systems. However, the work in these chapters is based on high level simulated models. To investigate whether these techniques are viable for real-life systems, further investigation with established benchmarks and commercially available architectures is required. Within this chapter, the development of such benchmarks and experiments takes place.

5.1.1 Motivation

The Automotive Industry is a major consumer of embedded systems and a strong driver of development and innovation. The average car now has more than 40 Electronic Control Units (ECUs), with some vehicles possessing up to 80 [31]. These control units have jobs varying from climatic control to safety critical roles such as air-bag deployment and emission regulation. As with most industries, minimization of cost is a primary motivator, meaning that a small reduction in the overall dollar cost per vehicle will lead to a significant saving for the consumer and increased profitability. Therefore, using lower specification and cost embedded controllers within the on-board ECUs could be a significant driver of development over the next five to ten years [84].

5.1.2 Real-Time Systems

In this chapter, we develop a method of scheduling where tasks of different levels of criticality within a system, including those which must meet real-time criteria, are combined in the same system. This is common within an automotive environment, and places additional requirements on the design of such devices. This, along with the differences in scheduling compared to standard paradigms is discussed in Section 2.1.2.

Section 5.2 covers the benchmarks used, with the algorithm in Algorithm 4 managing the jobs within this taxonomy.

5.1.3 Scheduling

The complexity of some mixed-mode algorithms can be thought of as detrimental when working with minimal overhead and could be counterproductive to system availability. Therefore, for working with high processor utilisation, a new method of finding optimum schedules without adversely adding to the overall computation time would be highly beneficial and a strong driver of cost reduction.

One solution that may meet this criteria is the use of Heuristic Algorithms, which obey simple rules based on deterministic factors within the system, meaning the uncertainty can be managed and an optimal schedule delivered within a usable time-frame. Designers can use these known facts to manage the non-deterministic aspects of the environment and deliver an optimal schedule in a timely manner. Rule based schedulers are attracting interest, especially with the increase in gate and core count on an integrated circuit causing complexity to increase exponentially.

The models developed tend to take on simple rules, which over time lead to a stabilized outcome. Ideally this outcome will maximize for all players, making the game pareto optimal. This way, if a sudden change occurs, the game will become unstable and eventually settle over a set of repetitions. Within a system, this scheduler would have many players and input variables, therefore a method must be developed that can reduce these into an optimal model and manage scheduling based on key parameters only.

5.2 Methodology

5.2.1 Development of Heuristic-Based Schedulers

To develop a heuristic based algorithm, a simplified system was modelled in MATLAB. This took the form of Figure 5.1, with jobs arriving and being sent to cores by a scheduler. The cores would manage these jobs and inform the scheduler when they are available to accept new tasks.

Three models were chosen for testing. These were:

- First Come First Served (FCFS)
- Priority Queuing
- Heuristic Algorithm under investigation.

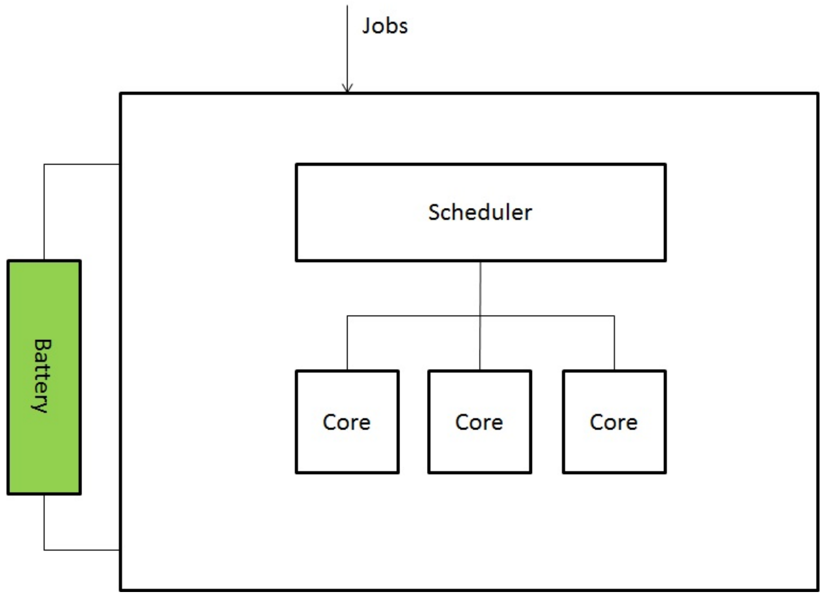


Figure 5.1: Simplified Microprocessor Design for Simulations

An issue with FCFS, Priority Queue and other scheduling algorithms is the failure to dynamically manage cores to maximize system availability. Many also lack the ability to manage precedence and are unaware of energy as an input. The proposed solution in this chapter provides these important items, while also managing job direction through simple heuristics, rather than a complex management program.

System Level Simulation

An algorithm, which can be seen in Algorithm 4, was created to allow System Level Simulations to take place. This let a series of jobs be run through a Monte Carlo Simulation. Jobs are created based on a rate (λ) and processed based on another rate (μ), similar to the work conducted in Chapter 3. One to three cores can also be implemented, allowing the difference in queue times, queue length and processor utilization to be observed. For this jobs are assumed to be arriving from activation (t_0) at a rate of $\lambda e^{-\lambda t}$ and processed at a rate of $\mu e^{-\mu t}$. The simulation is run over 72000 cycles with 10 repeats for each case and λ/μ values of 10, 30 and 50. One, two and three cores are active in versions of the simulation and all simulations are repeated ten times to give confidence in the consistency and capability of the simulation model.

Once job arrival and service rates are determined in lines 4-5, the system enters the 72000 operational cycles. The algorithm considers available energy prior to scheduling and adjusts the level to activate the Heuristic section through variable $MaxQueueLength$. Cores are only activated to service jobs if the current queue length is greater than this variable, preventing the excessive consumption of energy but increasing total operation time for jobs. If all cores are active

```
Set $\lambda$ and $\mu$
Set Flags and Clock to zero, Energy to 100
for $ArrivalTimeandServiceTime = 1$ to 72000 do
    $ArrivalTime(n) = \lambda e^{-\lambda t}$
    $ServiceTime(n) = \mu e^{-\mu t}$
end for
while $Clock < 72000$ do
    if $Energy < 50$ then
        $MaxQueueLength = 2$
    else
        $MaxQueueLength = 4$
    end if
    for $AllActiveCores$ do
        $ServiceTime - -$
    end for
    if $QueueLength > 0$ then
        if $CoreisEmpty$ & $Coreisactive$ then
            $PlaceJobonCore$
        else if $CoreEmpty$ & $CoreInactive$ & $QueueLength > Max$
        then
            $ActivateCore, PlaceJob$
        end if
        if $Clock = ArrivalofNewJob$ then
            if $CoreisEmpty$& $Coreisactive$ then
                $PlaceJobonCore$
            end if
        else if $CoreisEmpty$ & $Coreisinactive$ &
        $QueueLength > MaxQueueLength$ then
            $ActivateCoreandPlaceJob$
        else if $Allcoresarebusy$ &
        $Jobisnon-priority$ then
            $PlaceJobinQueue$
            $QueueLength + +$
        else if $Allcoresarebusy$ & $Jobispriority$ then
            $PlaceJobinPriorityQueue$
        end if
    end if
    for Each Active Core $1 - x$ do
        Core($x$)EnergyConsumed=$\mathcal{U}(0.002, 0.005)$
    end for
    Energy=Energy-AllCore($x$)EnergyConsumed-$\mathcal{N}(0.01, 0.005)$
    if $Energy <= 0$ then
        Break
    end if
    $UpdateWaitTime, IdleTime, QueueLengths$
    $Clock + +$
end while
print Value of Energy, Cores, $\lambda$, $\mu$, Average Wait Time, Max Wait Time,
Average Queue Length, Max Queue Length, Clock, Cores 1-3 Idle
Percentage
```

Algorithm 4: Simulation for Heuristic Scheduler

and a job is low priority, it will be placed into the queue at the tail, replicating FCFS. However, priority jobs will be placed into the priority queue, which will be serviced by the next available core. All data was outputted to a CSV file on completion of 72000 cycles.

Data from the simulations was collected and analysed using Minitab to look for statistical differences between scheduler types and key parameters to be used in any improvement exercises.

Addition of Energy

Once these experiments were complete, giving a baseline of algorithm effectiveness, the extra uncertainty of energy was added to the simulations. This was done by placing a battery into the simulation with a percentage of charge. The FIFO and Priority Queues were unaware of this and therefore continued executing jobs at the same rate until failure. However, as the state of charge decreased, the aggression of the heuristic algorithm in activating cores decreased, meaning the execution time for jobs increased. While this would seem to decrease the quality of service, the goal is to maximise lifetime of the system. Since less cores are activated, both the static and dynamic energy consumption are decreased thereby increasing the operational life of the system. Priority queuing is still active and priority jobs will be fast tracked to the core, with jobs currently occupying it halted. Once all priority tasks are cleared from the system, the non-priority jobs may resume executing. In extreme cases, the scheduler may activate a core to push through more priority jobs, thus maintaining quality of service for a small energy penalty. Though this will shorten the operational life of the entire system, missing priority jobs could be hazardous, especially in real-time or safety critical systems — meaning that this reduction in system lifetime is justified by keeping key systems active. For this simulation, the data was again outputted to a CSV file on completion of the 72000 cycles or if system energy reached zero during simulation.

Design of Experiments

Following this preliminary work, a Design of Experiments was conducted to determine each variables overall contribution and whether any interaction between variables took place. This model was then analysed by the General Linear Model (GLM) to give the percentage contribution (ϵ) that each parameter, including error, delivers to the overall system. The DoE was set up in Minitab as a two-level, four input, full factorial DoE with five repeats, giving 80 data points (5x2^4). Parameters were as follows:

- Level of Battery to increase Queue Length for Core Activation = 30/60
- Kick Out of non-priority jobs for priority jobs = Off/On
- λ=10/50
- μ=10/50

These results were analysed to determine key parameters for the GLM to determine the contributions each factor gives to the overall effect. These factors can allow further testing to give ideal results and an optimal scheduler design for the Heuristic Environment.

```
1: $LoadSpeedandLoadLookup$
2: $CheckEngineState$
3: if $EngineState == Stall$ then
4:     $Anti - StallFueling$
5: else if $EngineState == Crank$ then
6:     $CrankFueling$
7: else if $EngineState == Decel$ then
8:     $DecelerationFueling$
9: else if $EngineState == Run$ then
10:    $ReadEngineSpeed$
11:    $ReadEngineLoad$
12:    $InjectionDuration = LookupfromSpeedandLoad$
13:    $InjectionTime = LookupfromSpeedandLoad$
14:    $Advance = LookupfromSpeedandLoad$
15:    $Dwell = LookupfromSpeedandLoad$
16: else
17:    $EngineNotRunning$
18: end if
```

Algorithm 5: Tooth to Spark Algorithm

5.2.2 Development of a Real-Time Algorithm using Automotive Benchmarks

To simulate the scheduler, a platform and benchmarks were first chosen. For the hardware, the Atmel SAM3X8E was selected. This is a single core chip based on the ARM Cortex M3 architecture, widely used in embedded control [9]. This is also fitted to the Arduino Due [8], an open source prototype platform, allowing easy crossover of this work to hardware testing, which can be seen in Chapter 6. For benchmarks, an assortment of jobs from the Embedded Microprocessor Benchmark Consortium [32] were reviewed and the following were chosen:

Tooth to Spark

Correct timing of the ignition spark within a gasoline engine is vital, not only for good delivery of power, but also to ensure tailpipe emissions are within legal limits. If the spark is too early (overly advanced), combustion will be similar to the top line of Figure 5.2, rising quickly and giving a great deal of later detonations (known as "knock") that are harmful to an engine piston and cylinder (see Figure 5.3). If the spark is too late (overly retarded), combustion occurs too late in the crank angle, reducing efficiency. Correct timing (second line of Figure 5.2) should give peak pressure just after TDC, maximizing the conversion of thermal energy to mechanical energy. Within the EEMBC Benchmarks, the algorithm for calculating Tooth to Spark time can be seen in Algorithm 5 and must calculate the correct air/fuel mix and ignition timing, outputting these based on the engine conditions. This benchmark must also recognize when the engine is out of run state or in a zero fuel condition and adjust injection timing accordingly. This algorithm assumes stoichiometric conditions, where the ratio of available air/fuel and required air/fuel for full combustion is 1.

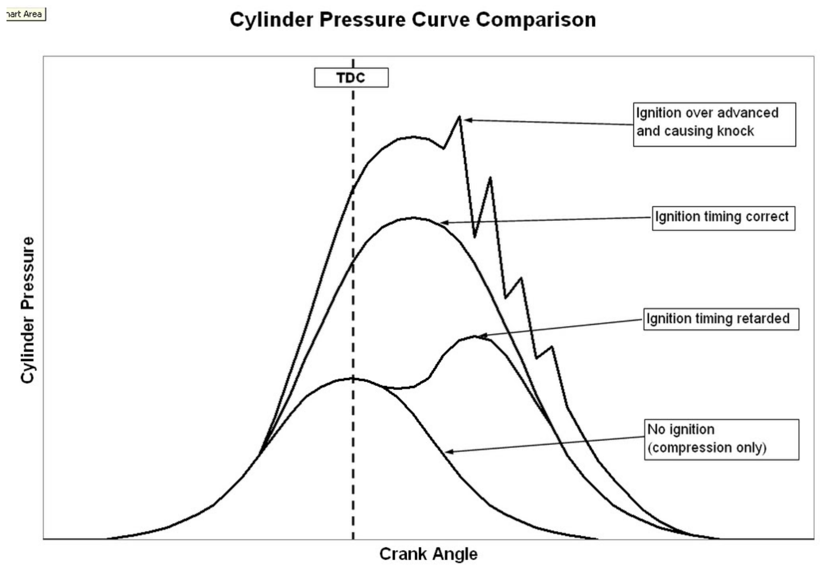


Figure 5.2: Graph showing Cylinder Pressure with respect to Ignition Timing

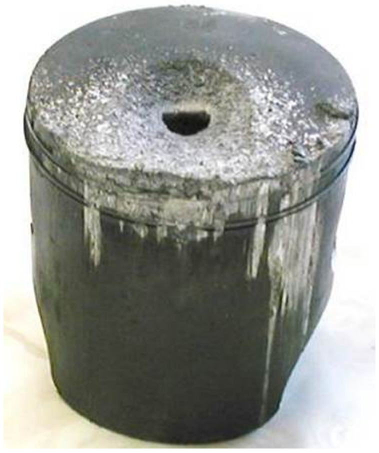


Figure 5.3: Piston Damage caused by late ignition

```
1: $ReceiveCANMessage$
2: if $ReceivedAddress! = NodeAddress$ then
3:     $Ignore$
4: else
5:     if $MessageType == RDR$ then
6:         $SendRDRAck$
7:         $SendData$
8:     else
9:         $StoreData$
10:    end if
11: end if
```

Algorithm 6: Algorithm for CAN RDR Benchmark

```
1: $TakeCounterReading$
2: $TakeTimeSinceLastReading$
3: $Revs = \frac{Counter-PrevCounter}{TeethPerRev}$
4: $Distance = \frac{Revs}{SampleTime}$
5: $Distance = Distance * TyreSize$
6: $KMH = \frac{Distance*3600}{1000}$
7: $PrevCounter = Counter$
8: $OutputKMH$
```

Algorithm 7: Algorithm for Road Speed Calculation Benchmark

CAN Remote Data Request

Since the development of Automotive CAN by Bosch in 1990 [28], it has become the main standard for in-vehicle communication. Since all systems (nodes) share the same CAN Bus, it is vital that traffic is managed to prevent Bus overloading. Messages within CAN can be sent to all nodes (Broadcast Announce Messages: BAM) or specifically to one Source Address. For this Benchmark, shown in Algorithm 6, the controller must recognize messages sent to it; ignoring those for other nodes, store the data if needed or reply if the message sent is a Remote Data Request (RDR). This gives a good test of system memory management and use of communication protocols, ensuring the processor is capable of handling these types of request.

Road Speed Calculation

A vehicle must be capable of calculating road speed for many reasons: It must know if the sensor is operating correctly, it must identify tampering and give a valid output back to the vehicle speedometer so the driver can accurately know their speed. Road speed is normally calculated from the number of rotations made by a toothed wheel, counted by a transducer such as a Hall Effect sensor. Since the number of teeth per revolution is known, along with the circumference of the wheel, deducing road speed is a case of simple mathematics based on the number of pulses recorded over a sampling period. The algorithm for this Benchmark is shown in Algorithm 7.

For this simulation, the jobs were assumed to have the following parameters for Activation, Computation, Deadline, Period and Worst Case Execution Time

(WCET):

1. Tooth to Spark: Hard Real-Time; Variable A_i, Variable C_i; Variable D_i; Aperiodic; $100\mu s$ WCET
2. CAN Remote Data Request: Soft Real-Time; Variable A_i; Set C_i; Set D_i; Aperiodic; $10\mu s$ WCET
3. Road Speed Calculation: Firm Real-Time; Set A_i, Set C_i; Set D_i; Periodic (10ms); $25\mu s$ WCET

Due to sources of non-determinism within these benchmarks, there are situations where EDF and RM are both sub-optimal and cannot deliver 100% of jobs before their respective deadlines. Therefore, the scheduler must decide which jobs can be ignored to ensure optimal operation of the system and therefore vehicle.

Within these tests, it can be seen that Tooth to Spark is not only the most critical, but also the most difficult to control.Missed jobs could lead to incorrect combustion; meaning the engine emissions are not compliant with the law and could also cause physical damage to areas such as the piston. For this benchmark, A_i and D_i will alter with respect to engine speed and C_i will vary with respect to fuel time and how advanced/retarded ignition time must be compared to engine top dead centre (TDC). Given that a spark ignition engine runs at a maximum of 8000rpm, has four cylinders and a standard spark coil takes 1ms to charge; the minimum time available for calculation of a spark is 500us, assuming a consistent firing of 30 degrees before TDC.

Figure 5.4 shows a flexible system based on Algorithm 4. This adds the value of priority into the system, allowing hard or firm real-time jobs to run at the expense of soft jobs. In this example, a missed CAN message or vehicle speed sensor reading is less serious than a missed spark time: the CAN Protocol automatically retransmits a non-acknowledged message after a period of time [28] and the speed sensor algorithm allows for occasional missed transmissions; since noise disrupting the sensor reading can cause inaccuracies in their calculation. If this occurs, then the algorithm will reject the calculated result [42], though continual missed readings would lead to a fault being flagged within the engine management ECU.

5.2.3 Plan of Testing

To test the viability of different schedule types, these three jobs were built in MATLAB, with an overall time-triggered scheduler running at $1\mu s$ monitoring their operation. This schedule model was then run in First Come First Served (FCFS), EDF and the algorithm in Algorithm 4 in cases where all jobs could execute successfully, along with scenarios where overruns would occur.

5.2.4 Design of Experiments

Following validation of the three scheduling types, a Design of Experiments was used to optimize the Heuristic Scheduler. The DoE was set up in Minitab as a two-level, three input, full factorial DoE with three repeats, giving 24 data points ($3\text{x}2^3$). Parameters were as follows:

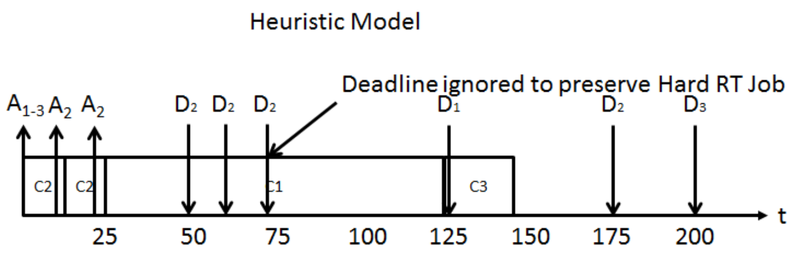


Figure 5.4: Heuristic Scheduling Example
In this example, the system is aware Job 1 has more importance than Job 2 and therefore runs Job 1 at the latest possible interval based on its WCET to successfully execute while also killing one instance of Job 2 to ensure system availability

- Queue Length to activate Heuristic = 2/10
- Number of CAN Jobs to supersede VSS in $V(fi)$ = 5/20
- CAN Job Arrival Rate=150/400

These results were analysed to determine key parameters for the GLM and determine the contributions each factor gives to the overall effect.

5.2.5 Final Optimization Tests

Following on from the DoE, which gives a level of contribution to each input factor, final runs to determine optimal conditions for use were performed. These runs generate data that when analysed in a regression plot give maximum and minimum operating points for the system. From these, static settings for the schedule can be derived. It may also be possible to feed these values back into the algorithm, allowing for automatic fine tuning of the schedule parameters; altering items such as Queue Limits during bursts of high intensity to preserve Hard Real-Time jobs. Finally, a long term simulation was run, simulating the operating cycle across the Modified New European Driving Cycle (MNEDC) shown in Figure 5.5, used in Europe for proving new engines meet emission legislation by simulating four urban driving cycles and one extra-urban cycle over an 1180s run. Within the automotive industry this is used to determine fuel economy and emissions such as CO_2, NO_x (Oxides of Nitrogen) and Particulate Matter below $10\mu m$ ($PM10$), but is also used by certification agencies to test software compliance [28]. Therefore it is deemed acceptable for simulation testing of this new scheduling system.

5.3 Results

5.3.1 Heuristic Algorithm

This short experiment shows that within an FCFS environment, the addition of a second core drastically reduces the wait time for jobs, especially when $\lambda > \mu$.

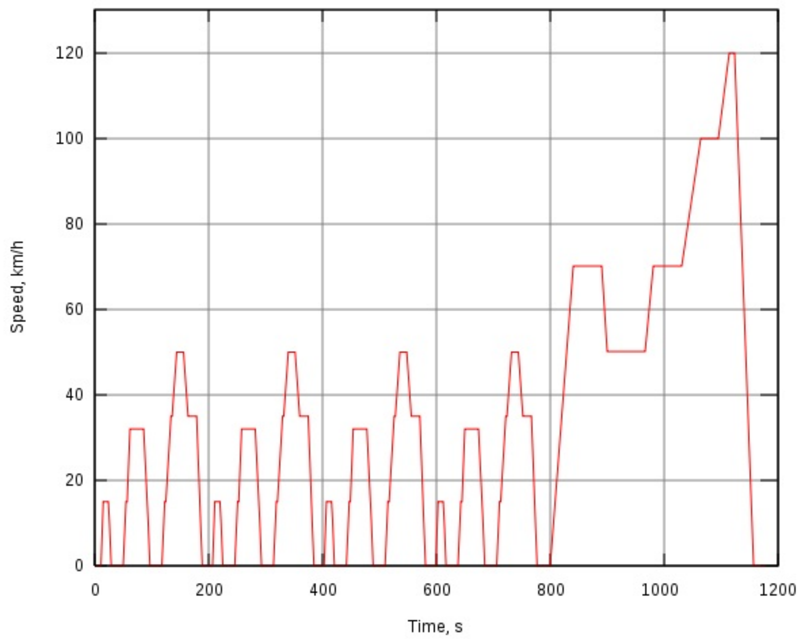


Figure 5.5: The MNEDC
This gives a mixture of urban and extra-urban driving and is used for type approval of new vehicles [74]

Even when $\lambda < \mu$, maximum waiting times for execution can be excessive in a single-core environment. Therefore, while simple to implement and good in many common cases, FCFS must be considered sub optimal for the schedule design. However, we will continue to run tests and include it due to the use it gives us as a baseline measurement.

For the simulations undertaken, Table 5.2 shows the outcome for tests conducted where $\lambda = \mu$ and energy was considered. For this cycle, the Heuristic Algorithm outperforms both FCFS and Priority Queuing by a factor of 3, giving a statistically significant result. In cases where $\lambda \geq \mu$, the Heuristic Algorithm consistently outperforms its rivals and also gives significantly longer operating life in situations where $\lambda < \mu$ due to the dynamic deactivation of superfluous cores. This result is shown for other values of $\frac{\lambda}{\mu}$ in Figure 5.6, demonstrating that the Heuristic Scheduler outperforms both FCFS and Priority Scheduling for a range of values and no statistical change in results as load rate increases found in a regression test (P=0.507, therefore no correlation between job rate and clock cycles completed).

When tests were conducted with single and dual core architectures the results, seen in Figure 5.7, show the Heuristic Algorithm continued to outperform both FCFS and Priority Queuing. With only one core active, Heuristic methods give a factor of two increase in operational lifetime due to the dynamic management of cores and their deactivation during light loading — similar to the sleep mode present in many modern microprocessors. As the core count in-

```
if $\Sigma C_i < \min D_i$ then
    $Continue$
else
    if $\Sigma C_i < SparkD_i$ then
        if $\Sigma C_i - CANC_i < SparkD_i$ then
            $KillCANJob$
            $CANKill++$
        else if $\Sigma C_i - VSSC_i - CANC_i < SparkD_i$ then
            $KillVSSJob$
            $KillCANJob$
            $CANKill++$
        else if $SparkC_i < SparkD_i$ then
            $KillCurrentJob$
            $KillVSSJob$
            $KillCANJob$
            $CANKill++$
        else
            $SetInjectionParameters = 0$
        end if
    else if $\Sigma C_i < VSSD_i$ then
        if $(\Sigma C_i - CANC_i < SparkD_i) \& CANKill < 10$ then
            $KillCANJob$
            $CANKill++$
        else if $\Sigma C_i - CANC_i < SparkD_i) \& CANKill >= 10$ then
            $CANKill = 0$
            $KillVSSJob$
        end if
    end if
end if
```
Algorithm 8: Algorithm for Automotive Heuristic Scheduler

Table 5.1: Change in Queue Times and Lengths for a One-Core and Two-Core System When Job Arrival (λ and Job Service (μ) Rates are altered

λ	μ	Average Queue Time	Maximum Queue Time	Average Queue Length	Maximum Queue Length	Core 1 Idle Percentage	Core 2 Idle Percentage
50	40	5640	9345	73.73	129	0.04	N/A
50	40	154.70	814	0.91	11	30.29	37.46
50	60	233.25	757	2.30	12	18.28	N/A
50	60	69.51	571	0.17	5	49.37	69.55

creases, the effectiveness of the Heuristic Management can still be seen, giving improvements of 61 % for two cores and 35 % for three.

Following analysis of the DoE, Figure 5.8 shows an increase in Lambda, as well as deactivation of the Kick Out function have a significant effect on the number of successful clock cycles completed. Further analysis, shown in Figure 5.9, reveals that increasing the value of $MaxQueueLength$ in Algorithm 4 gives some increase to operational life and concurs with Figure 5.8 that being able to remove tasks from processors to allow execution of priority jobs (the "kick-out" function) reduces operational life of the system – as this will increase the

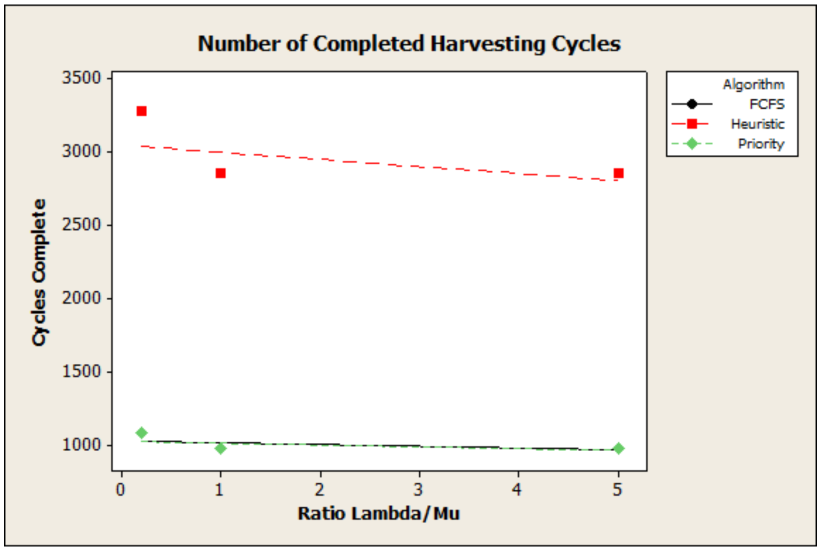


Figure 5.6: Scatter-plot showing Clock Cycles Complete for increasing values of lambda over mu

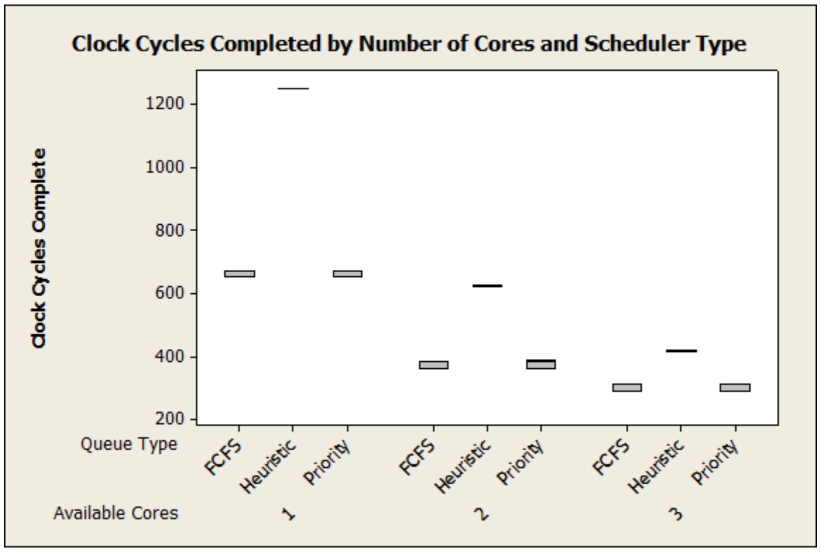


Figure 5.7: Box-plot of Microprocessor Life for varying number of cores

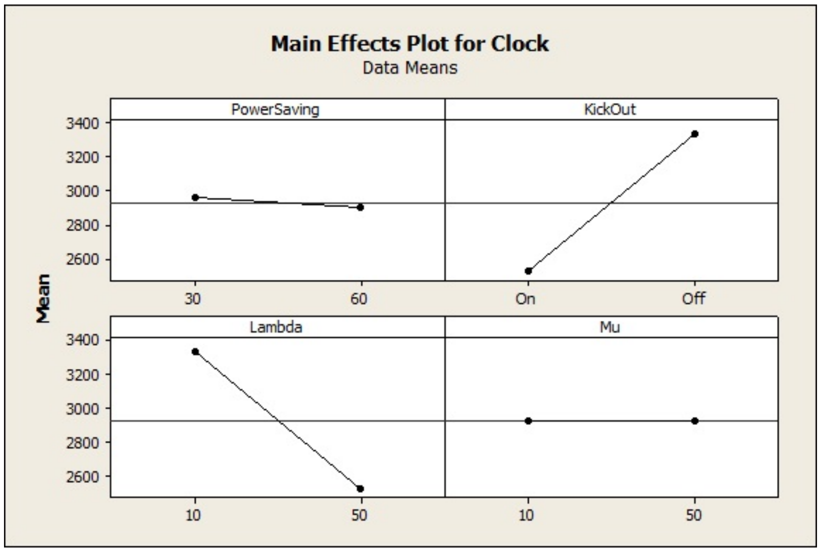


Figure 5.8: Main Effects Plot of DoE
This shows an increase in Lambda, as well as deactivation of the Kick Out function have a significant effect on the number of successful clock cycles completed.

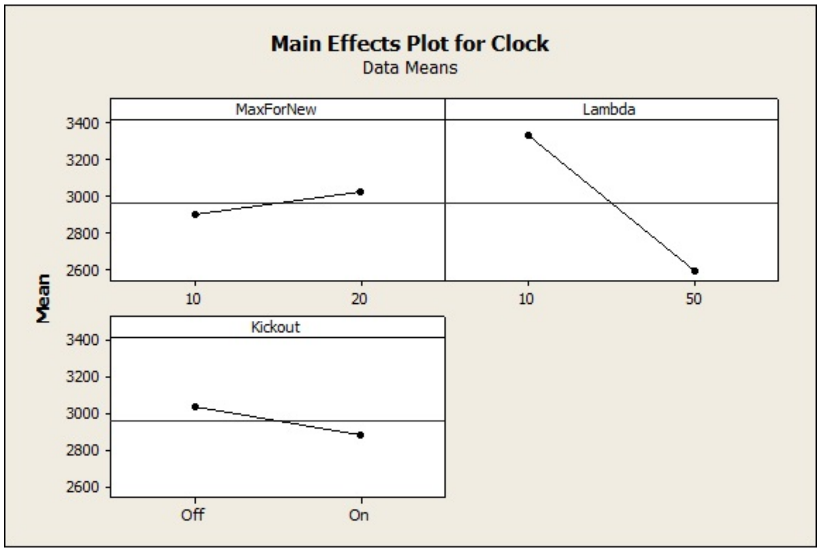


Figure 5.9: Revised Main Effects Plot
This shows further inputs to those in Figure 5.8. As well as Lambda and Kick Out still having an effect, increasing the maximum queue length before another core is activated also can be seen to increase successful clock cycles.

Table 5.2: One-way ANOVA Results for Number of Clock Cycles Completed Compared to Queuing Method

Queueing Method	**Mean**	**Standard Deviation**	**P-Value**	**Conclusion**
FCFS	971.00	0.25	0.00	Heuristic Outperforms
Priority Queue	970.96	0.32		
Heuristic	2848.79	0.54		

Table 5.3: Epsilon Scores for General Linear Model Following DoE

Source	**P-Value**	ϵ
Power Saving	0.000	0.152
Kick Out	0.000	33.162
Lambda	0.000	33.097
Mu	0.756	0.000
Power Saving*Kick Out	0.000	0.153
Power Saving*Lambda	0.000	0.156
Power Saving*Mu	1.00	0.000
Kick Out*Lambda	0.000	33.125
Kick Out*Mu	0.022	0.000
Lambda*Mu	0.165	0.000
Power Saving*Kick Out*Lambda	0.000	0.155
Error		0.000
Total		100.0

utilisation of processors and thus affect energy consumption. Due to this, a second analysis only looking at priority jobs was undertaken and can be seen in Figure 5.10. For this, deactivating Kick Out can be seen to have a large effect on missed priority jobs as these are now made to wait until a core has completed execution, rather than allowing the job immediate access.

The results from these experiments were placed into a GLM within Minitab to see determine main contributions to the microprocessor lifetime. The results for this, shown in Table 5.3, clearly demonstrates the contribution the kick-out function and lambda have on the system, accounting for 99 percent of the effect seen. For these runs, error can be seen to be zero, due to the level of non-determinism afforded by the simulation being low. For repeat runs on a real microprocessor, this value would be expected to rise significantly.

Table 5.4: Missed Spark Jobs based on an increase of CAN Job rate with Schedulers

CAN Job Rate	Missed Spark FCFS	Missed Spark EDF	Missed Spark Heuristic (Proposed Work)
10	0	20	0
20	0	0	0
30	0	585	0
40	0	241	0
50-90	0	0	0
100	0	350	0
110	0	3	0
120	0	12	0
130	0	0	0
140	0	8	0
150	0	885	0
160	0	419	0
170	0	213	0
180	0	3	0
190	0	1483	0
200	0	0	0
210	0	10	0
220	0	769	0
230	0	389	0
240	0	3	0
250	9	128	0
260	3	3	2
270	0	148	10
280	0	223	4
290	1	70	4
300	6	286	22
310	3	236	27
320	0	0	145
330	89	208	0
340	43	156	182
350	179	370	17
360	431	7	143
370	1	2142	182
380	148	1710	85
390	213	1	250
400	630	282	0
410	171	1944	0
420	295	860	181
430	320	1596	326
440	0	532	1102
450	360	4683	52
460	413	5	762
470	408	3	40
480	1347	943	316
490	1389	2494	0
500	271	5	76

5.3.2 Real-Time Automotive Heuristic Scheduler

Time for Execution of 100000 cycles

- FCFS: 2776 seconds
- EDF: 13152 seconds
- Heuristic Scheduler (Algorithm 8): 2790 seconds

FCFS Scheduler

Figure 5.11 and Tables 5.4 and 5.5 show that the number of missed Spark and VSS jobs increases as the arrival rate of CAN jobs increase. When checked on a regression plot, the value for Spark has a high R^2 value of 56% and is statistically significant ($P = 0.000$). This means a change in y (missed spark jobs) can be correlated to a change in x (CAN Job arrival rate), with 56% of points matching this interdependence. Therefore, this shows that for a high CAN Load, FCFS is not suitable as the regular arrival of CAN jobs monopolizes the processor, preventing critical jobs from successfully running.

Table 5.5: Missed VSS Jobs based on an increase of CAN Job rate with Schedulers

CAN Job Rate	Missed VSS FCFS	Missed VSS EDF	Missed VSS Heuristic (Proposed Work)
10	0	30	0
20	0	0	0
30	0	371	0
40	0	20	0
50	0	18	0
60	0	6	0
70	0	12	0
80-110	0	0	0
120	0	9	0
130	0	0	0
140	0	4	0
150	0	27	0
160	0	146	0
170	0	177	0
180	0	0	0
190	0	26	0
200	0	65	0
210	0	0	0
220	0	11	0
230	0	22	0
240	0	7	0
250	0	0	0
260	0	0	0
270	0	218	0
280	0	922	12
290	0	83	0
300	0	333	36
310	0	33	7
320	0	0	0
330	51	30	0
340	0	158	87
350	72	377	0
360	129	0	99
370	164	48	93
380	189	49	149
390	166	0	134
400	189	111	0
410	166	36	0
420	223	12	236
430	257	16	143
440	0	188	253
450	256	73	237
460	310	0	286
470	347	0	351
480	300	2426	365
490	0	0	0
500	431	0	0

EDF Scheduler

Figure 5.12 and Table 5.4 shows a sporadic increase in Spark and VSS jobs misses as the arrival rate of CAN jobs increase. The value for Spark has a low R^2 value of 20.5% and is statistically significant ($P = 0.005$). Since this is EDF, the low C_i and short D_i of the CAN Jobs causes misses for both Spark and VSS. Adding pre-emption would not solve the issue, as the CAN A_i is so frequent that it is absorbing all available processor runtime and causing the other jobs to miss their deadlines. Therefore, in this case, EDF is not optimal. Rate Monotonic would also not be suitable, since CAN has the most regular activation time, this means it would receive the highest priority and continue to consume all processor availability.

5.3.3 Proposed Heuristic Scheduler

Figure 5.13 and Table 5.4 show a reduction in missed jobs as the CAN Job rate increases. However, this has a comparable outcome to the EDF Algorithm, with R^2 of 26.9% and $P = 0.001$ for missed Spark Jobs, meaning that high

Table 5.6: Analysis of Variance (ANOVA) Results for Missed Spark jobs with Respect to Scheduling Algorithm Used

Source	**Mean**	**Standard Deviation**
FCFS	134.6	295.8
EDF	488.6	868.4
Heuristic	78.6	197.7
P-Value		0.000

Table 5.7: Analysis of Variance (ANOVA) Results for Missed VSS jobs with Respect to Scheduling Algorithm Used

Source	**Mean**	**Standard Deviation**
FCFS	60.5	112.7
EDF	121.3	366.8
Heuristic	49.8	98.8
P-Value		0.245

CAN Loads can still cause issues. While these numbers are similar, the equation driving the regression is significantly different, with a lower gradient and intercept than the EDF example. However, Figure 5.14 shows that not only is the peak value of missed jobs lower, but also the overall distribution. Due to the low number occurring across the simulation, Figure 5.15 shows a similar distribution for FCFS and the Heuristic Algorithm, which was expected.

As the algorithm runs in near equivalent time to the FCFS and significantly faster than EDF, this is deemed to be a substantial improvement. Since this is a hybrid approach, working in FCFS under light loading, it gives a compromise between the short runtime of FCFS and complex management of jobs only when this is required.

Statistical Analysis of Results

These results in Table 5.6 show that for missed Spark Jobs, FCFS and the Heuristic Scheduler significantly outperform EDF. When analysed separately, they give a non-statistically significant difference ($P = 0.268$), but higher values for CAN Load could cause a significant outcome. However, it is thought that these values would lie outside of normal operating conditions; therefore these will not be tested.

For VSS in Table 5.7, no statistical difference can be seen at a 95% Confidence interval. Again, this could occur for higher values of CAN Load, but was not investigated as part of this work.

5.3.4 Design of Experiments

Missed Spark Events The Design of Experiments shows only one statistically significant outcome for missed Spark Jobs. Figure 5.17 clearly shows the increase in CAN arrival rate gives a direct proportionality to the increase in missed Spark jobs, which is to be expected as these are dominating the processor. However, it also shows that the point of switching between FCFS and the heuristic scheduler can also be important. By increasing the number of jobs within the queue from 2-10 before the heuristic scheduler activates, the average number of missed spark jobs dropped by 20%. This shows that the hybrid approach adopted by this Heuristic is of critical importance in reducing the number of missed jobs, by only activating the slower priority-based scheduler when it is needed.

Table 5.8 shows the General Linear Model output for the DoE. This shows the CAN Arrival Rate is dominant in missed spark jobs, contributing 24% to this event. Queue length on its own and combined with this gives an 8% contribution, with error accounting for 68%. The error value is high due to the high amount of non-determinism within the system caused by parameters such as engine speed and engine load giving non-linearity and therefore deviation from the GLM. This value could be reduced through steady-state operation of the system in a known window of use, but for this initial DoE it is deemed acceptable as the epsilon scores have proved which variables are key in their effect.

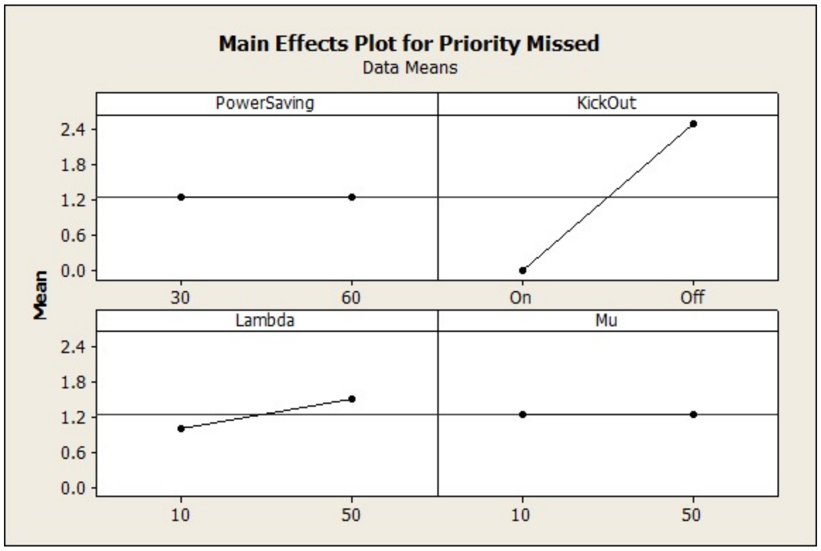


Figure 5.10: Further Revised Main Effects Plot
This shows the change in Priority Jobs missed when inputs are altered for the Heuristic Algorithm. Deactivating Kick Out can be seen to have a large effect on missed priority jobs as these are now made to wait until a core has completed execution, rather than allowing the job immediate access.

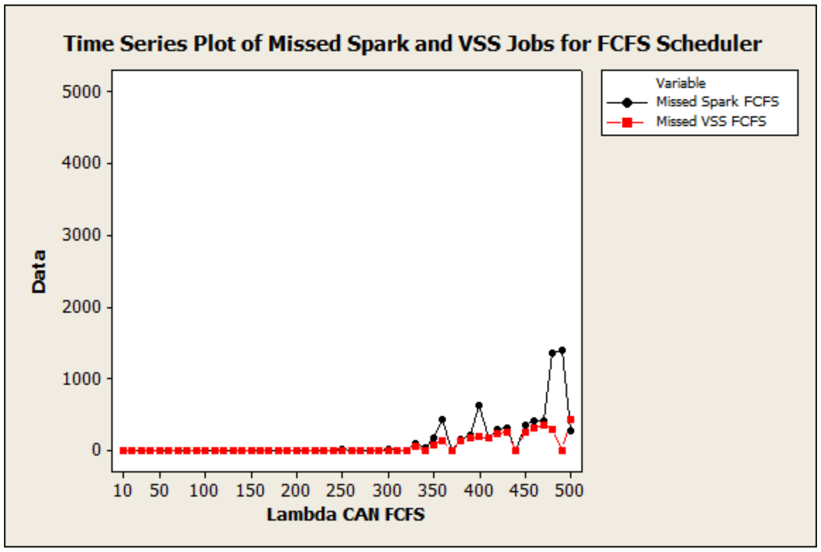


Figure 5.11: Graph of Missed Spark and VSS jobs for a FCFS Scheduler as the CAN Load increases

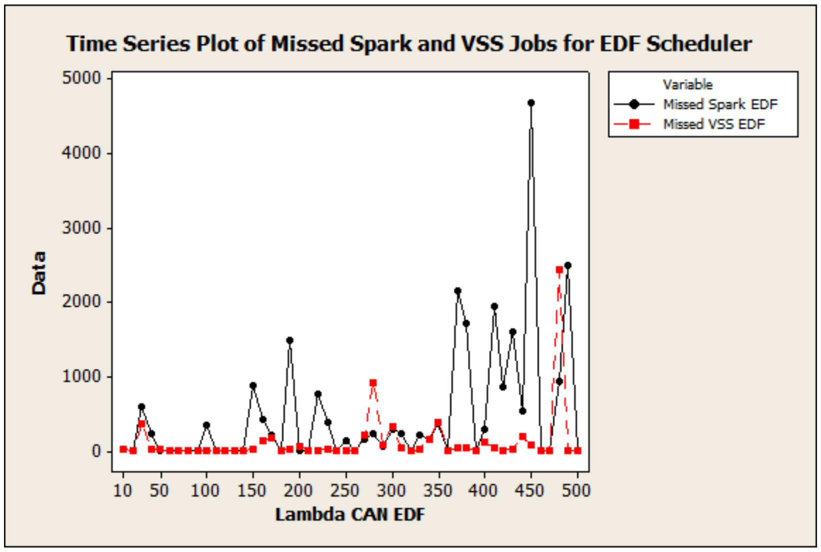


Figure 5.12: Graph of Missed Spark and VSS jobs for a EDF Scheduler as the CAN Load increases

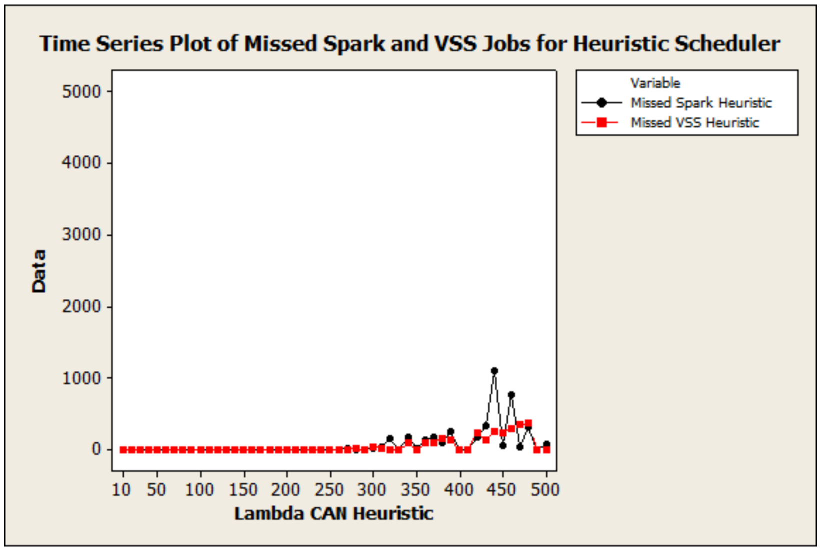


Figure 5.13: Graph of Missed Spark and VSS jobs for the Heuristic Scheduler as the CAN Load increases

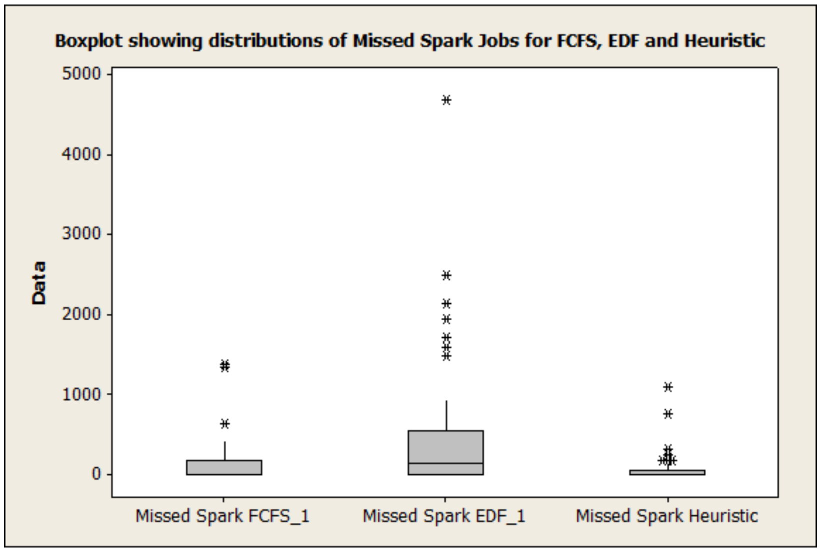


Figure 5.14: Box-plot showing distribution of missed Spark jobs for the three algorithms
Heuristic shows a lower maximum value and lower overall distribution of missed jobs

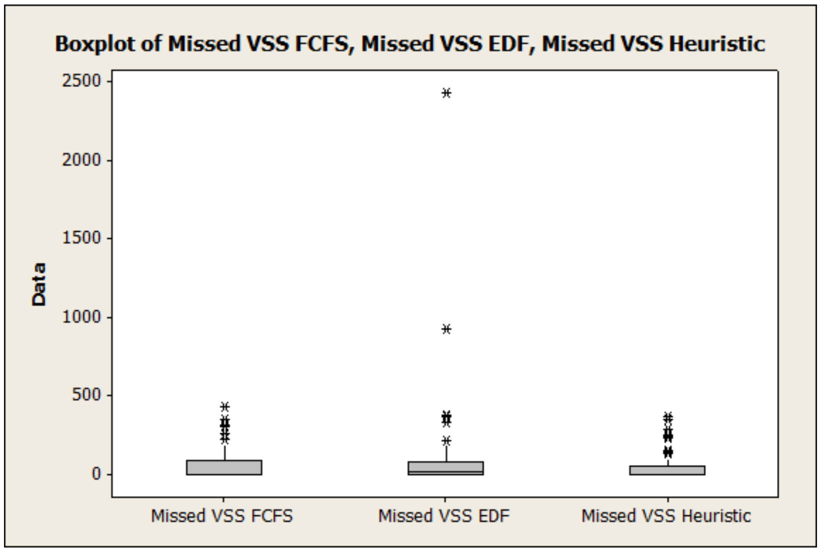


Figure 5.15: Box-plot showing distribution of missed VSS jobs for the three algorithms
Due to the low period of this job, little overall difference can be observed

Table 5.8: Epsilon Scores for General Linear Model Following DoE for Missed Spark Events

Source	**P-Value**	ϵ
Queue Length	0.348	3.948
CAN Superseding	0.949	0.018
CAN Arrival Rate	0.029	24.362
Queue Length*CAN Superseding	0.926	0.038
Queue Length*CAN Arrival Rate	0.348	3.948
CAN Superseding*CAN Arrival Rate	0.949	0.018
Queue Length*CAN Superseding *CAN Arrival Rate	0.926	0.038
Error		67.631
Total		100.0

Table 5.9: Epsilon Scores for General Linear Model Following DoE for missed VSS Events

Source	**P-Value**	ϵ
Queue Length	0.971	0.005
CAN Superseding	0.352	3.441
CAN Arrival Rate	0.009	32.723
Queue Length*CAN Superseding	0.859	0.121
Queue Length*CAN Arrival Rate	0.980	0.002
CAN Superseding*CAN Arrival Rate	0.347	3.524
Queue Length*CAN Superseding *CAN Arrival Rate	0.851	0.138
Error		60.045
Total		100.0

Missed VSS Events The Design of Experiments shows only one statistically significant outcome for missed VSS Jobs. Figure 5.19 clearly shows the increase in CAN arrival rate gives a direct proportionality to the increase in missed VSS jobs, which is to be expected as these are dominating the processor. However, it also shows the number of missed CAN Jobs required to allow CAN to dominate VSS has a minor effect. By increasing the number of missed CAN jobs from five to 20, the number of missed VSS jobs drops by a factor of two. Figure 5.20 also shows an interaction between Queue Length and CAN supersession. This is caused by the more frequent execution of Algorithm 4 at a low value of Queue Length, meaning the opportunity for CAN jobs to supersede VSS Jobs occurs more frequently. Changes in this interaction may be higher at more extreme values for these parameters, but whether these levels would occur in a real-life system is debatable.

Table 5.9 shows the General Linear Model output for the DoE. This shows the CAN Arrival Rate is dominant in missed spark jobs, contributing 32% to this event. CAN supersession on its own and combined with this gives an 7%

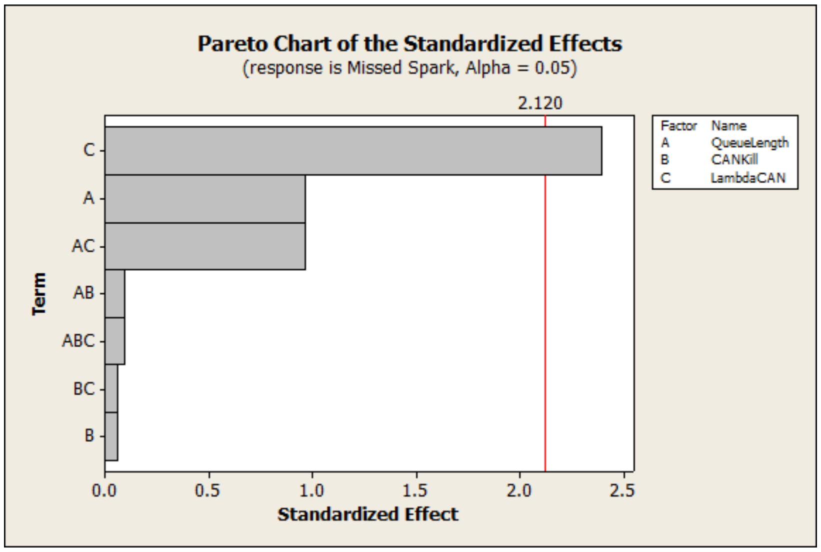


Figure 5.16: Pareto Plot for the Design of Experiments
This shows that only Factor C (increasing CAN Arrival Rate) has a significant effect on the rate of missed Spark Jobs

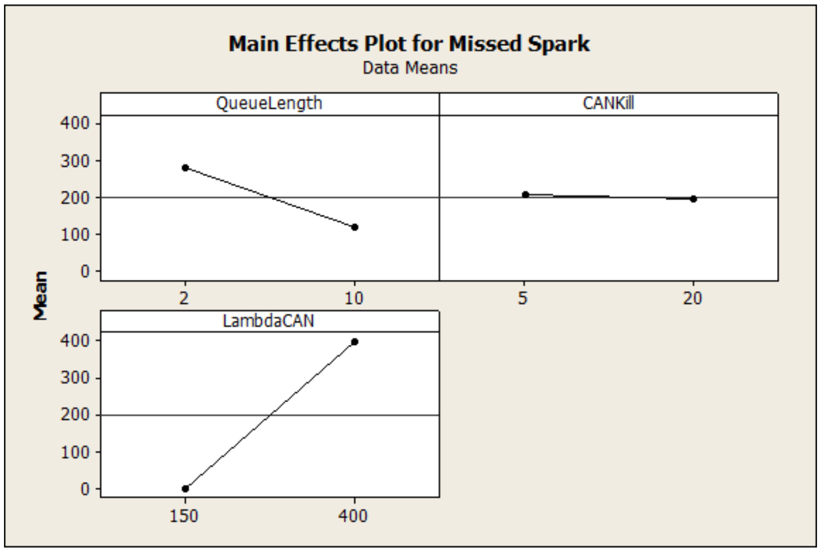


Figure 5.17: Main Effects Plot for the Design of Experiments
This shows that increasing the value for running the Heuristic Scheduler causes a reduction in the number of missed Spark Jobs. This also reduces the number of missed jobs caused by the increase in CAN Jobs

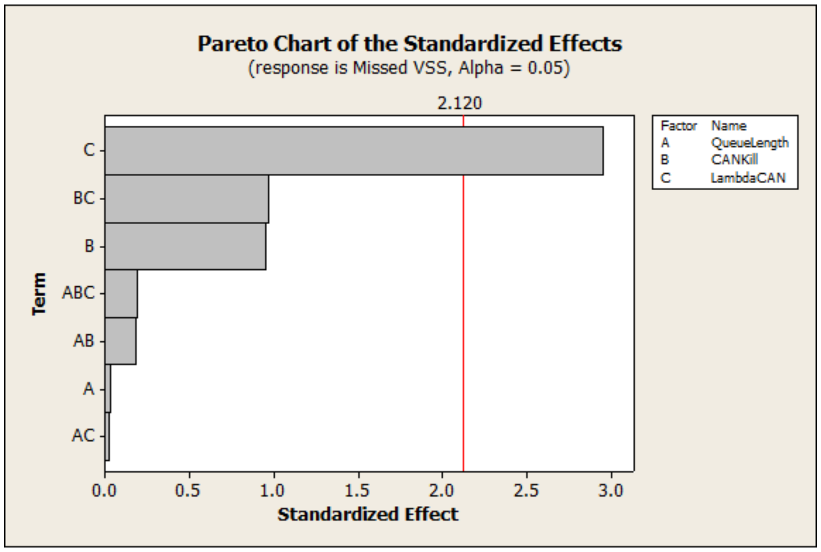


Figure 5.18: Pareto Plot for the Design of Experiments
This shows that only Factor C (increasing CAN Arrival Rate) has a significant effect on the rate of missed VSS Jobs

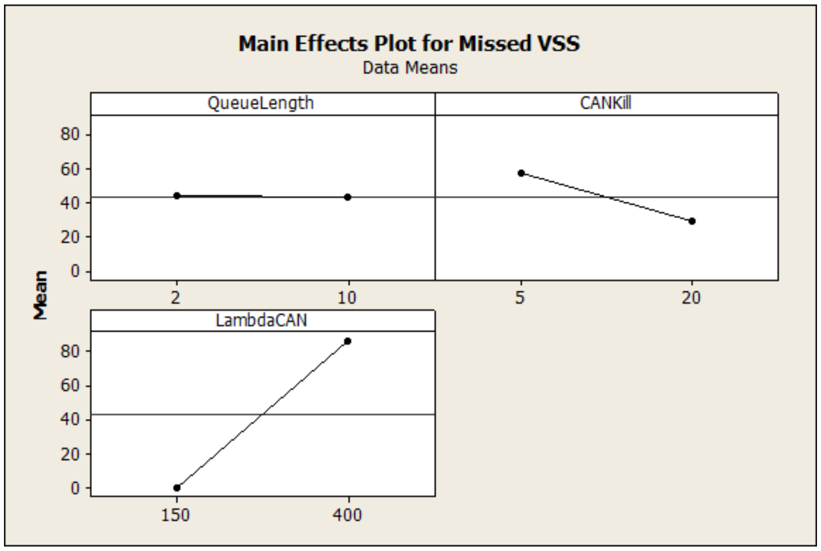


Figure 5.19: Main Effects Plot for the Design of Experiments
This showing that increasing the value for running CAN Jobs in place of VSS Jobs (Lines 21-27 in Algorithm 4) decreases the number of missed VSS jobs. Unlike in Figure 5.17, Queue length has no real effect, due to the long period between VSS Jobs

contribution, with error accounting for 60%. As with the previous DoE, the high value of error is caused by the non-determinism within a complex system. For this initial experiment, which has identified key inputs for testing, this value of error is deemed acceptable.

5.3.5 Optimization of Final Settings

Once the DoE was complete, a further run was made with CAN Loading set to a value known to cause missed Spark and VSS Jobs. This allowed Queue Length and CAN Deactivation to be altered and the effects on missed jobs recorded. To ensure accuracy and confidence in the results, the number of required samples was calculated using a 1-sample t-test power and sample size calculation. Assuming a Type I Error (α - False Positive) of 5% and a Type II Error (β - Missed Failure) of 20%, giving a Power value of 0.8. For the Standard Deviations in Table 5.6, this gave a required sample of 128. Once these were complete, regression plots in Figure 5.21 and Figure 5.22 show a strong regression for Spark Jobs ($P = 0.011, R^2 = 21.1\%$) but no contribution for VSS ($P = 0.185, R^2 = 6.2$). A repeat run for Spark Jobs with Queue Length between 40 and 50, shown in Figure 5.23, gave no contribution ($P = 0.804, R^2 = 0.4$). This allows us to conclude that a Queue Length of 40 and a CAN Supersession rate of 20 would be suitable values for an optimal run. Since missed jobs within the Heuristic Scheduler are found to occur above a CAN Load of 300, a final run with CAN between 300 and 600 was performed, which can be seen in Figure 5.24 and Figure 5.25. Spark shows a weak correlation ($P = 0.047, R^2 = 10.5$) with a strong correlation from VSS ($P = 0.000, R^2 = 91.8$) meaning that missed VSS Jobs could be predicted from the previous CAN Load. With more test samples, the number of missed Spark jobs could become statistically insignificant, meaning that this section of the Heuristic Algorithm is now optimized.

Table 5.10 shows whether the changes made have had a statistically significant contribution to the number of jobs missed for a CAN Load of 400. As can be seen, since $P > 0.05$ this has not been proved. However, the reduction in the number of missed jobs is visible and could become more significant at higher CAN Loads.

Reliability plots, shown in Figure 5.26 for FCFS and Heuristic Algorithms shows a slight increase in overall system availability for the Heuristic Algorithm. B_{50} life (the number of cycles at which 50% of the population will have at least one failure) has increased from 5014 to 5899; an improvement of 15% . This increase in availability is dependent on the CAN Load, meaning this should be carefully controlled to maximise system uptime. A long run, shown in Figure 5.27, demonstrates the effectiveness of optimisation. For 50 runs, 43 reached 1000000 cycles without failure and were censored from the analysis. The seven failures all occurred in early life, giving the system a B_{10} life of 7222 cycles and a B_{90} life of 1389335.

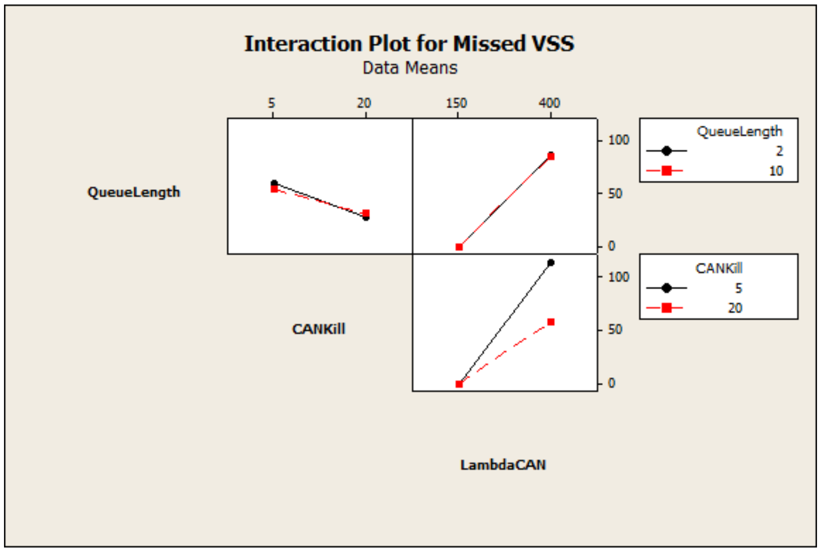


Figure 5.20: Interaction Plot for missed VSS Jobs
This shows a possible interaction for Queue Length to activate Heuristic Scheduling and number of missed CAN Jobs required to raise priority above VSS. For a higher value of Queue Length, fewer jobs are missed at a low value of CAN supersession

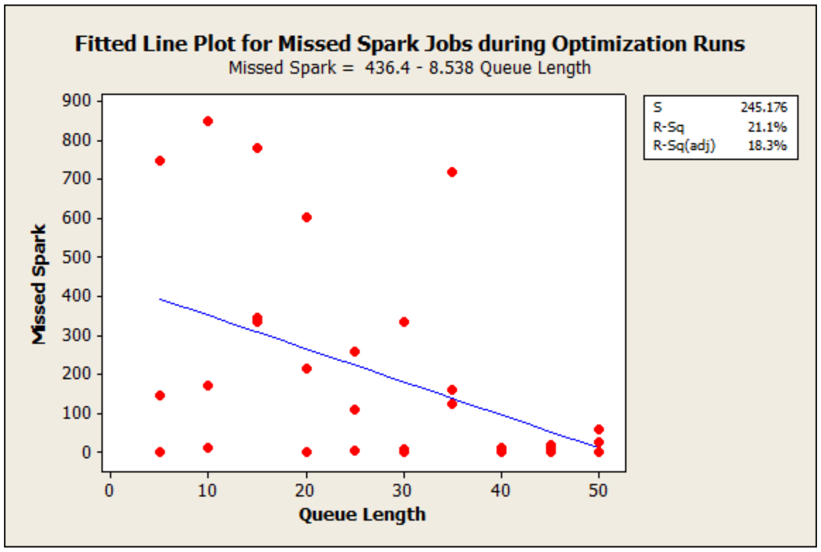


Figure 5.21: Regression Plot showing missed Spark Jobs with increasing Queue Length and fixed CAN Rate.
The data clearly shows a negative regression, improving as Queue Length to activate the Heuristic Algorithm increases

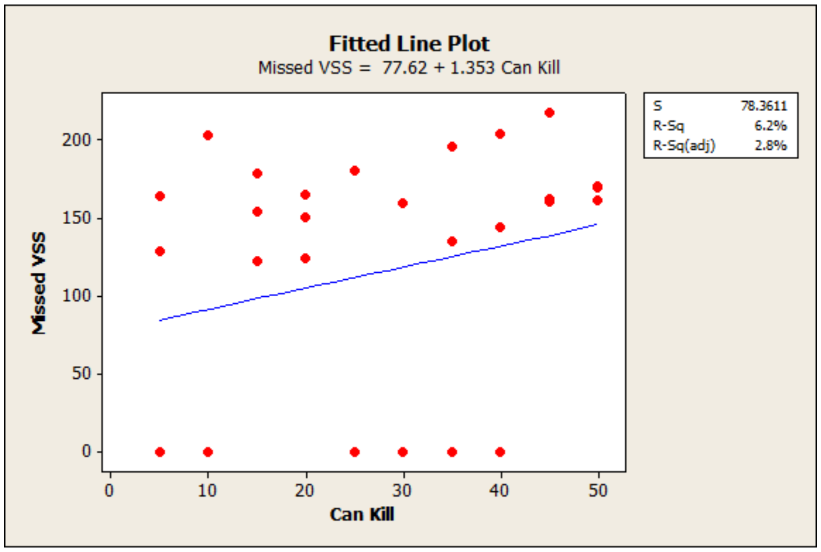


Figure 5.22: Regression Plot showing missed VSS jobs with increasing level of CAN Supersession and fixed CAN Rate.
No contribution is evident

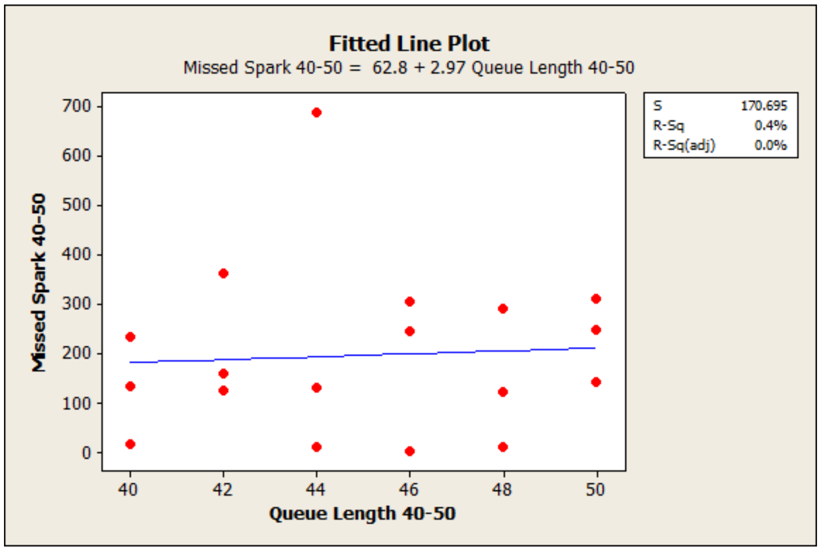


Figure 5.23: Regression Plot showing missed Spark Jobs with Queue Length between 40 and 50.
No contribution is evident

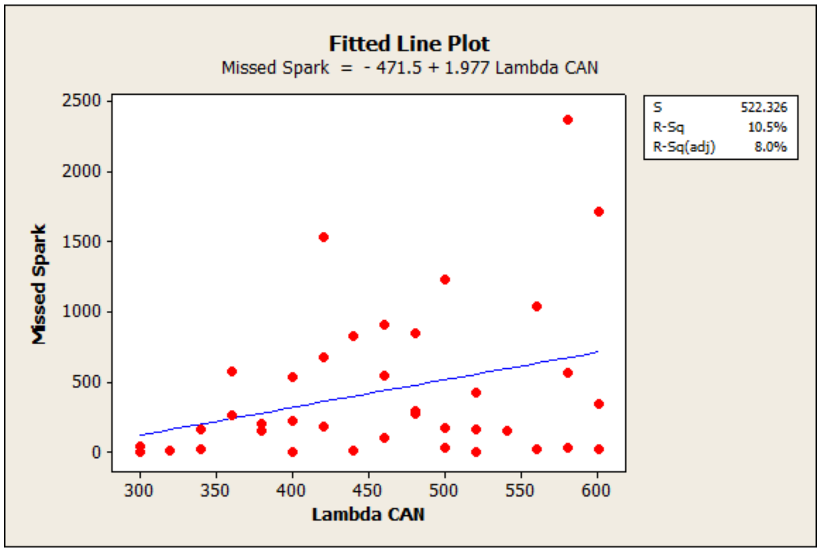


Figure 5.24: Regression Plot showing missed Spark Jobs with increasing CAN Load.

A weak correlation is present

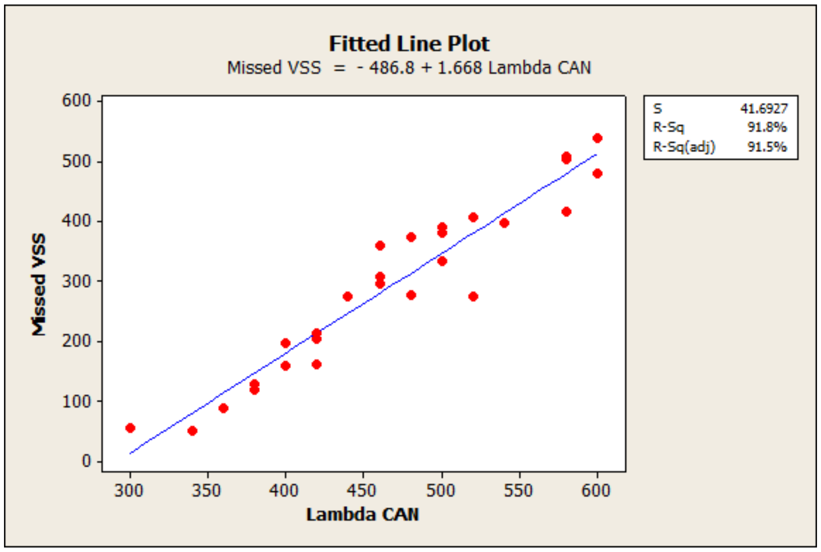


Figure 5.25: Regression Plot showing missed VSS Jobs with increasing CAN Load.

A strong contribution is evident

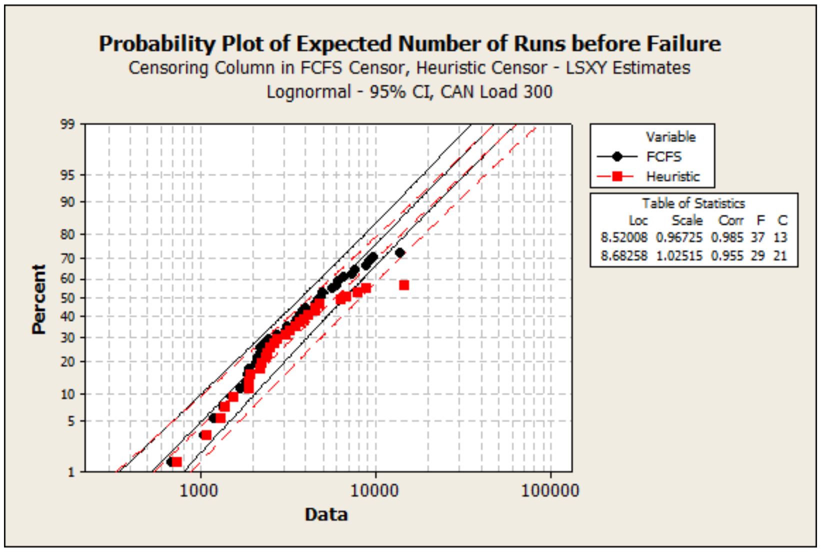


Figure 5.26: Reliability Plot showing the number of runs before expected failure for FCFS and the Heuristic Algorithm and a CAN Load of 300.

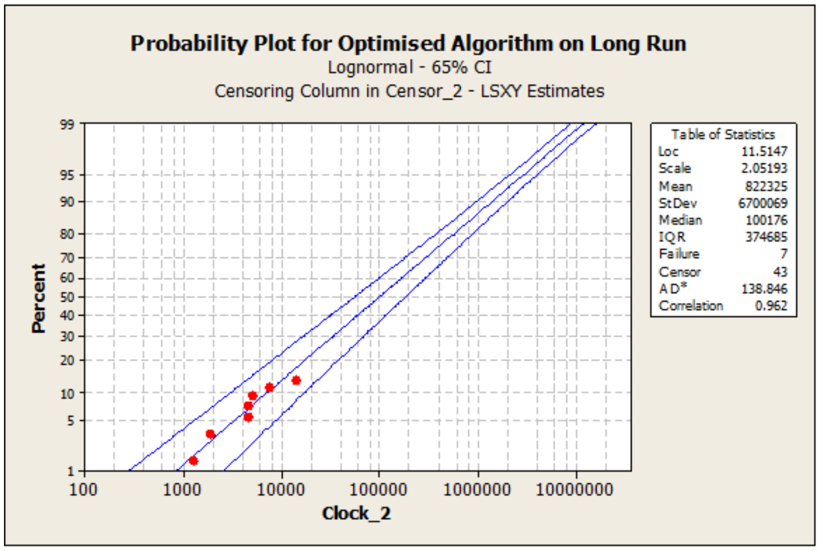


Figure 5.27: Reliability Plot for a 1000000 cycle run, halting on any missed Spark Job.

Table 5.10: Analysis of Variance (ANOVA) Results for Missed Spark jobs with Respect to Scheduling Algorithm Used following Optimization

Source	**Mean**	**Standard Deviation**
FCFS	418.7	416.1
Heuristic	315.1	285.8
P-Value		0.148

5.4 Conclusion

The initial work in this chapter shows that under low loads ($\lambda < \mu$), a first come first served scheduler is capable of managing all jobs easily. With only one core active, FCFS gives a high value of maximum wait time compared to average wait time. Therefore this would not be suitable for a system with priority jobs. With two cores active however, this wait time reduces dramatically and priority queuing may not be required. However, when $\lambda > \mu$, priority queuing and multiple active cores becomes a feasible way to manage jobs.

The heuristic algorithm presented provides the flexibility of both strategies, combined with dynamic core management and therefore increased energy efficiency. By altering the queue length required to activate a new core with respect to energy available the algorithm, this extra non-deterministic aspect can be managed and the quality of service for an end user maintained.

When $\lambda \gg \mu$, a drop in performance for all scheduling methods takes place. As the arrival rate is so much larger than the service rate, the queue grows exponentially; meaning processor utilisation is always at 100 %. Due to this, no dynamic core management can take place and all schedulers perform at a comparable rate. Within these simulations, this leads to FCFS and Priority Queuing outperforming the Heuristic Algorithm, as the model designed considered the extra complexity required and increased leakage power accordingly — causing lifetime for the Heuristic Algorithm to be reduced.

While this system has only been tested up to three cores, it is thought that the algorithm would feasibly cope with a larger number of available devices in its current design. Since the scheduler operates as an overseer for all devices, it simply places jobs onto the first core it finds available; or activates a core if required. A limitation for this is the simulation work required to determine suitable values for $MaxQueueLength$ and the Energy at which to alter this. This work also currently presumes a homogeneous layout, but it is expected that heterogeneous microprocessors (where cores of different design are placed on the same silicon) will become more commonplace in the near future.

The Automotive Heuristic Scheduling Algorithm successfully reduces the number of missed spark jobs for a negligible increase in execution time. Through the use of a Design of Experiments, an optimal value for all input variables within the heuristic system has been determined. Repeated runs on MATLAB with these optimized values found a further reduction in missed spark jobs for an equivalent execution time and a low number across the MNEDC Cycle in Figure 5.5. Repetition with higher CAN Load found an improvement in missed jobs for the Heuristic Scheduler over FCFS. VSS also shows strong correlation to CAN Load rate, meaning a further improvement could be made by monitoring previous CAN Loads and determining a level to pause CAN for other jobs based on this, possibly based on Statistical Process Control; similar to work in Chapter 3.

Spark jobs are only missed at high CAN Loads. It is concluded that when CAN loading is very high, the processor is given over so exclusively to this that it causes missed deadlines due to small amounts of runtime deviation within other jobs. More complex algorithms could compensate for this, but would require more advanced processors; which is a contradiction to the goals of this chapter. Therefore, to use a lower level processor such as the one simulated here, system designers must be mindful of lower priority jobs and their possible monopolisation of a processor, so must adjust parameters accordingly to prevent this occurring.

This work shows, through simulation, that a low complexity embedded system is capable of running multiple complex benchmarks within an automotive environment. By using a hybrid approach of FCFS and heuristics, a significant reduction in the number of critical missed jobs can be seen for a minimal increase in runtime.

The use of simulation has allowed rapid alteration and testing of these algorithms, at significantly reduced cost compared to hardware testing in an engine test cell. Issues have been debugged on line, which would not be possible with hardware testing, with no risk of hardware damage.

This chapter proves the concept of a heuristic algorithm is viable and can dynamically manage a multi-core stored energy environment with minimal complexity. Key factors in its management have been identified and initially tested through simulation, which has allowed rapid testing of the many permutations and validation of the concept. The progression of this work is to conduct a Design of Experiments (DoE) to optimize the algorithm through determining each input variables overall contribution to both wait time and system availability. DoEs have been used in previous work to great effect in identifying key items within an energy harvesting environment [29].

While conceptually the design can be construed as sound, tests in a practical multi-core architecture will determine whether the algorithm works in a real-world aplication. Tests using a real-world processor and desmonstrating the hardware implementation of this algorithm can be seen in Chapter 6.

Chapter 6

Hardware Testing

It is pointless to do with more, what can be done with less
William of Ockham

6.1 Introduction

As the previous chapters show, the use of heuristics and Game Theory in scheduling yields an improvement in results for real-time and energy harvesting systems. However, these results are only based in simulation, meaning they can only be seen as a proof of concept. To take this further, the algorithms must be ported from simulation into a micro-controller and tested.

6.2 Possible Micro-controllers

To test this work in hardware, it was decided to use an up-to-date processor with high capabilities. Though a multi-core system was initially suggested, it was decided to minimise development and learning time by using a single-core system based on an educational or prototyping platform. Four such systems were identified that could deliver the benchmarks in Chapter 5. These were:

- Raspberry Pi
- Arduino Due
- Imagination Technologies Minimorph
- Texas Instruments Beagleboard

6.2.1 Raspberry Pi

Created specifically for teaching in schools and universities, the Raspberry Pi is a stripped down computer that has found popularity within the engineering community [90]. The board runs a version of the Debian Operating System, with Python as the primary programming language. Options for interfacing are available, making this board highly successful; with over 1 million sold in the first 18 months [7] .

Table 6.1: Summary of Prototype Boards

	Arduino Due	**Minimorph**	**Raspberry Pi**	**Beagleboard**
OS	Arduino C	Linux	Debian	Linux
Cores	1	1	1	1
Processor	ARM Cortex M3 SAM3X8E	XENIF TZ1090	ARM1176JZFS	ARM Cortex 8
Memory	512MB	255MB	512MB	256MB
SD Interface	Upgradeable	No	Yes	Yes
Switchable Clock	Yes	No	Yes	Yes
Board Power Consumption	2W	2.5W	3.5W	2W
Real Time Capable	Yes	Yes	No	Yes
Forum Support	Yes	No	Yes	No

6.2.2 Arduino Due

Developed in Italy, the Arduino family of Micro-controllers are an established platform for prototyping [8]. Widely used by amateur and professional engineers, they allow easy interfacing to external components and programming of the on-board micro-controller though a bespoke compiler, running a variant of the C programming language capable of managing a variety of inputs and outputs.

6.2.3 Minimorph

Designed to help professionals develop embedded solutions [48], the Minimorph is developed by Imagination Technologies for companies to ease integration of their devices into larger systems. The board manages all interfacing, allowing the user to concentrate on managing signal conditioning and decisions. Running Linux gives access to application and device support, along with maximising the capabilities of the on-board DSP.

6.2.4 Beagleboard

Manufactured by Texas Instruments, the Beagleboard is an open source device designed for the education market [108]. Similar to the Raspberry Pi, it runs a Linux Shell and is capable of producing high-quality graphics and sound encoding. An update to this has been the ”BeagleBone” released in 2012, which is similar to the Arduino in concept and delivery.

A summary table of each board’s features can be seen in Table 6.1

6.3 Determining Micro-controller

To determine the most suitable board from these four, a Cause and Effects Matrix was used. This is a common tool in the Measure Phase of Six Sigma (see 2.4.4), allowing quantitative and objective analysis to determine the most suitable or noteworthy items. To use this, an item is given importance; scored from 1 to 10. Each board is then scored on how well it fits this suitability with a score of 1 (remotely fits), 3 (moderately fits) or 9 (strong fit).

Table 6.2: Cause and Effects Matrix for Microprocessor

Factor Score	**10**	**8**	**7**	**6**	**6**	
Processor	**Use of Real-Time**	**Power Consumption**	**Switchable Clock**	**Forum Support**	**Development Tools**	**Total**
Arduino Due	3	9	9	9	9	**273**
Minimorph	9	3	1	1	3	145
Raspberry Pi	3	1	9	9	9	209
Beagleboard	3	3	9	1	1	129

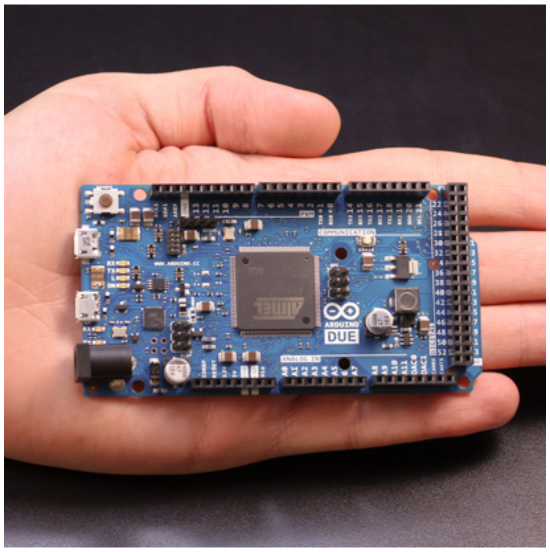


Figure 6.1: The Arduino Due

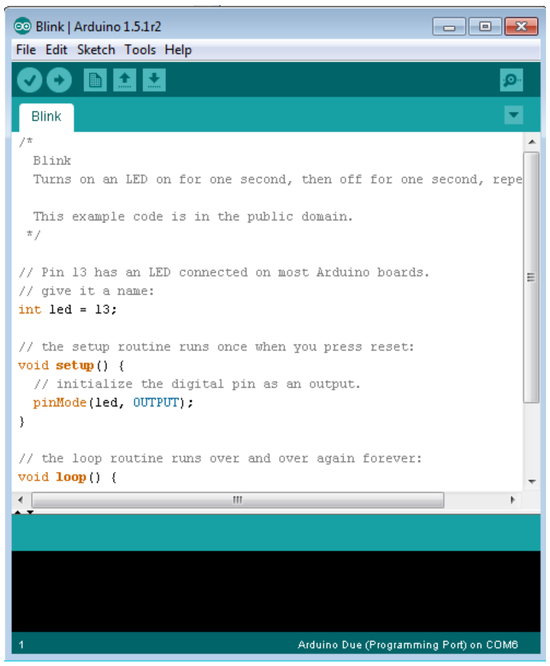
For this C&E, the following factors were taken into account and scored, with the results in Table 6.2:

- Use of Real Time: Score 10
- Power Consumption: Score 8
- Switchable Clock: Score 7
- Level of Forum Support: Score 6
- Development Tools Available: Score 6

As Table 6.2 shows, due to the ability to switch clocks and run a Real-Time Operating System (RTOS), combined with strong Forum Support and Development Tools, the Arduino Due was chosen as the development board on which to conduct tests.

6.4 The Arduino Environment

The Due runs at 84MHz with a 32-bit core. With 96kb of SRAM on-board and a Direct Memory Access (DMA) Controller, it is a small but powerful embedded


Blink | Arduino 1.5.1r2
File Edit Sketch Tools Help
Blink
1
Arduino Due (Programming Port) on COM6

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

Figure 6.2: The Arduino Development Environment

system capable of flexible reconfiguration and execution of complex algorithms through the ARM Cortex M3 architecture selected at the primary controller. Through the use of the Harvard Architecture [6], which gives instructions and data separate buses, the controller can deliver significant speed advantages over more complex controllers and can deliver operations on 4 byte wide data in a single CPU clock cycle.

6.4.1 Initial Testing

The Arduino is programmed through a USB Connection to a host PC and use of a development environment shown in Figure 6.2. The syntax is based on C-semantics allowing quick development with fundamental programming knowledge. Unlike "classic" C, where only a `void main()` is declared, the Arduino language has two `void()` classifications that exist to set and run parameters within the controller. The `void setup()` routine runs once and is used to initialise Input-Output parameters and perform any preliminary calculations on variables. Following this, the `void loop()` function runs continually to perform calculations and output results. A simple "Hello World!" example that makes the on-board LED blink is shown in Algorithm 9.

As it uses C, mathematical and logical actions can be performed on variables to set up and run simple loops. An example of this, adding pulse width modulation (PWM) to Algorithm 9 can be seen in Algorithm 10. This uses variables to control the time Pin 13 is set high and low, with a counter set to allow phasing of the LED brightness in a continual up and down cycle.

```
Pin 13 as Digital Output
void loop
**loop**
    Set Pin 13 High
    Wait 1 Second
    Set Pin 13 Low
    Wait 1 Second
**end loop**
```

Algorithm 9: Arduino Hello World Program

```
void setup
Set Pin 13 as Digital Output
$y = 25$
void loop
**loop**
    Set Pin 13 High
    Wait $x$ ms
    Set Pin 13 Low
    Wait $y - x$ ms
    **if** $UpDown == 1$ **then**
        $x += 1$
    **else**
        $x -= 1$
    **end if**
    **if** $x == 0$ **or** $x == 25$ **then**
        $UpDown =$ **not** $UpDown$
    **end if**
**end loop**
```

Algorithm 10: Arduino PWM Program

An issue with both these algorithms is using `delay()` within the Arduino environment to wait causes the processor to pause, meaning no further work can be carried out. Therefore, further algorithms perform waiting through the use of interval timers using the free-running counters `millis()` and `micros()` discussed below. This allows the loop section of the program to continually run and execute all functions with no degradation to QoS.

6.4.2 Use of Libraries

The Arduino Environment gives strong support to the use of libraries, which are identical to h files used in formal C. Designed by users and developers, these can reduce design time by giving defined functions for items such as digital or analogue IO or more complicated routines such as the creation of Hash Functions for digital signatures. For this research, a library called Scheduler was found on the Arduino Playground that allowed calling of functions at a time interval set within a built in command [15]. The Scheduler Library runs through two functions: `void update()` that checks whether a function within the queue must be called and `void schedule(function,time)` that adds a specific `void` function to the scheduler and defines how many milliseconds should elapse until this is called. The time aspect of this is managed by built in counters `millis()` that acts as a free-running clock within the Arduino once power is applied and runs for 50 days before an overflow and `micros()` which runs 1000 times faster than `millis()` for finer time management.

To test the Scheduler library effectiveness, the PWM example in Algorithm 10 was redesigned as Algorithm 11, calling PWM control as functions, with the time for execution set by variables.

6.4.3 Hardware Interfacing

One major advantage of the Arduino over systems such as the Raspberry Pi is the easy interfacing of external discrete components through the IO sockets mounted around the edge of the PCB that can be clearly seen in Figure 6.1. This is deemed such a critical difference that in [113] the authors give advice on interfacing an Arduino to a Raspberry Pi through a USB connection to run Python scripts on external devices. On the Arduino Due, 54 pins are available as digital IO (12 configurable as PWM), 12 as Analogue inputs and 2 as Analogue outputs. All IO pins are capable of delivering 130mA of current, with power managed on the PCB by a separate regulator circuit. This means while the Cortex-M3 operates at 3.3V, up to 12V can be supplied into the board to control ancillary components. Therefore, while the board can give out a 5V supply limited to 800mA, only 3.3V can be used on IO pins, otherwise serious damage to the Micro-controller could occur. This means any external circuits used with the board must be designed accordingly to limit input voltage and prevent electrical overstress (EOS) taking place.

6.5 Hardware Design

Once tests of the benchmark algorithms on the Arduino were complete, the external circuits to control conditions within the microprocessor were designed,

```
void setup
Set Pin 13 as Digital Output
$x = 1$, $y = 32$, $UpDown = 1$
Schedule SetHigh in 1ms
void loop
loop
    Check Scheduler for any required executions
end loop
void SetHigh
Set Pin 13 High
Schedule SetLow in $y - x$ms
void SetLow
Set Pin 13 Low
Schedule pwmUp
Wait $x$ ms
Set Pin 13 Low
Wait $y - x$ ms
if $UpDown == 1$ then
    $x+ = 1$
else
    $x- = 1$
end if
if $x == 0$ or $x == 25$ then
    $UpDown =$ not $UpDown$
end if
```
Algorithm 11: PWM Program designed using Scheduler Library

tested and fitted to the board. The systems under test are discussed in the sections below.

6.5.1 CAN System

For this work, the CAN libraries within Arduino will be used to transmit a CAN Signal onto a CAN Bus. Within the EEMBC Benchmark simulated in Algorithm 6, the system only transmits when a CAN Message is received containing a remote data request. Since the Arduino Due already has a CAN Transceiver fitted, this RDR will be sent by a simulated CAN Node on-board the Arduino. Output is on the ADCH header through pins 7 and 8, transmitting onto CANLo and CANHi respectively.

6.5.2 Vehicle Speed Sensor System

For the VSS system designed on the Arduino, only Vehicle Speed and Engine Speed will be used. These will be delivered through the use of astable 555 timers as seen in Figure 6.3 and Figure 6.4. The use of variable resistors gives a wide operating range, allowing assorted values of vehicle and engine speed to be fed into the system for analysis. As the on/off time of a 555 is determined by the equation $f = \frac{1.45}{(R_1+2R_2)C}$, this gives a frequency range of 3157Hz to 18947Hz for Engine Speed and 582Hz to 3062Hz for Vehicle Speed.

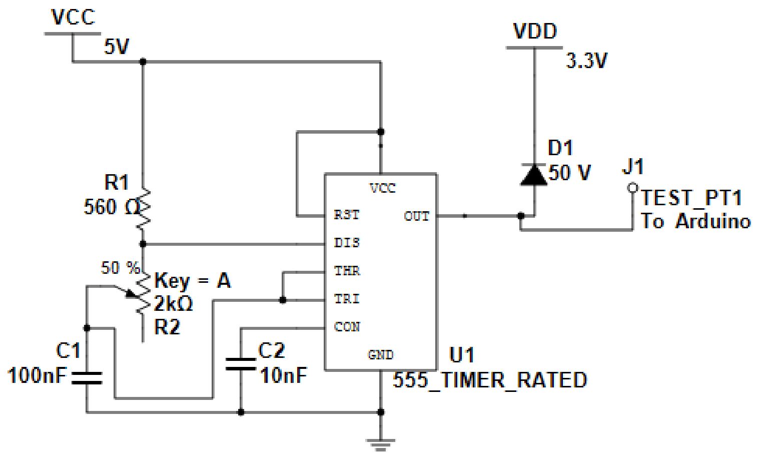


Figure 6.3: Engine Speed Sensor Design

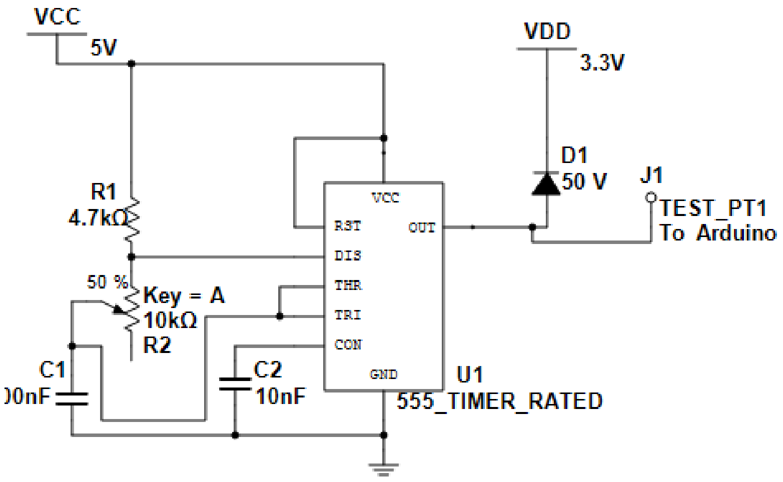


Figure 6.4: Vehicle Speed Sensor Design

6.5.3 Fuel Injector Control

Within this design, only two cylinders are considered, with the Arduino determining injection and dwell time, along with the injection volume based on engine speed, engine load and the value of λ detected at the exhaust. The output circuit is shown in Figure 6.5. This uses a low-side driver to energise the coil with a fly-back diode fitted in parallel to the inductive load, allowing dissipation of the coil current when the Field Effect Transistor (FET) is opened. This design is still commonplace within the automotive market and deemed more feasible to implement than systems relying on capacitance or magnetos that are seen in high performance and compact engines [42]. Two cylinders were used as this design of engine has recently seen a return in popularity after the success of the twin-air design used by Fiat on their new version of the 500 and are capable of delivering high fuel economy for low emissions, which could lead to a resurgence in this design in the near future.

6.5.4 Other Circuits

In addition to the circuits above, inputs simulating the accelerator and starter motor; along with a digital input to force overruns were added to the completed design. The full circuit can be seen in Figure 6.6.

The accelerator was chosen to be a simple single potentiometer due to the ease of implementation. Though automotive designs would normally use dual-potentiometer and include proximity switches to give redundancy and the ability to disable the throttle in should an issue arise, it was felt that since this project is not investigating the qualification of accelerators under ISO26262 that only using a single potentiometer throttle would be acceptable for development purposes.

To place the engine into a cranking (starting) state, a simple active-low switch (SW1) was placed onto a digital IO pin to simulate the activation of the starting motor. This will place the controller into the cranking state of the EEMBC Tooth to Spark benchmark and demonstrate the algorithm design is capable of delivering the full benchmark. A second switch (SW2) is present to place the algorithm into a wait cycle and force the processor to miss real-time jobs. This is present to ensure the system can recover from an overrun should one occur. Both these switches are configured as active-low as this removes the risk of an over-voltage taking place on a pin of the Arduino that could cause damage. Active-low is commonplace within automotive design for this reason [77].

To protect the main micro-controller from further damage, all inputs to the Arduino are connected to the 3.3V supply through clipping diodes. These are used so that any voltage over 3.3V present on a digital line will be dissipated through the Diode (D1 as an example) and prevent over-voltages on the digital IO Pins and are a suggested design addition in [44].

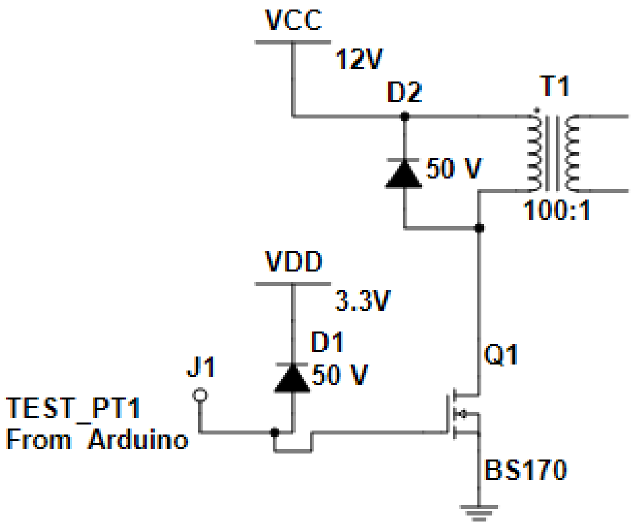


Figure 6.5: Circuit for Delivery of High Voltage to Spark Plug

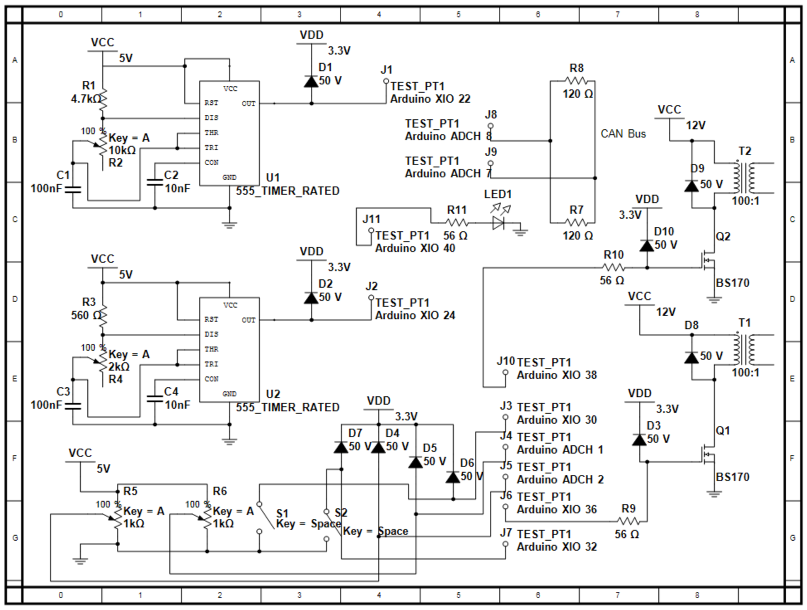


Figure 6.6: Full Hardware Design for Interfacing to Arduino

6.6 Methodology

6.6.1 Automotive Controller

Each circuit above was connected to the Arduino and tested seperately to ensure the device was capable of performing the required functions.

VSS and Engine Speed Calculation

To test the system with the two astable circuits, an algorithm shown in Algorithm 12 was created. This ran at the Nyquist limit for the engine speed sensor, which was calculated as $\frac{1}{2*20000} = 25\mu s$, based on the highest frequency present being 18947Hz and the Nyquist limit required being $\min(2 * freq)$. The circuits were set to a known frequency and checked using an oscilloscope. Following this, the Arduino was connected to these waveforms and run for five minutes. After this time, the number of pulses registered was outputted through the console window in the Arduino Environment. A target of less than three percent error was deemed acceptable for the Arduino to be considered capable of successfully monitoring VSS and Engine Speed.

Analogue Inputs

As stated previously, the Arduino has 12 Analogue inputs available. This allows analogue inputs such as the accelerator and engine load to be directly attached to these pins without additional analogue to digital conversion; as this is managed internally. To test this, Algorithm 12 was modified to set Pins A0 and A1 as analogue inputs and output the values of these to the terminal at a $25\mu s$ interval. The counter section of this algorithm was ignored, as this was not required to read in analogue values.

Switch Inputs and Outputs

To test the input switches and outputs from the FET and LED, an algorithm shown in Algorithm 13 was created. This observed the inputs from SW1 and SW2 in Figure 6.6 and activated Pins 22, 24 and 40 when these were read as zero, meaning the switch was closed. As the Arduino is capable of sourcing 130mA at 3.3V, the gate resistors and LED output resistor (R8, R9 and R10) were chosen to be 56Ω to ensure an overcurrent did not occur.

CAN Circuit

For the CAN Circuit, a library was downloaded from the Arduino Playground. This was then used to output signals at a $1ms$ interval to the CANLo and CANHi lines, which were observed using an oscilloscope. As this library contained built-in functions, these were later called from Algorithm 6 to test periodic CAN functionality, as this would be required in the full function tests. Timing was managed through the on-board `millis()` counter, also showing the Arduino was capable of running using a form of time-triggered interrupt.

```
void setup
Set Pin 22 and Pin 24 as Digital Inputs
void loop
loop
    if $Pin22State \neq PreviousPin22State$ then
        $PreviousPin22State = Pin22State$
        $Pin22Counter++$
    end if
    if $Pin24State \neq PreviousPin24State$ then
        $PreviousPin24State = Pin24State$
        $Pin24Counter++$
    end if
    Wait $25\mu s$
    Output $Pin22Counter\&Pin24Counter$
end loop
```

Algorithm 12: Counter Algorithm to test Arduino with VSS and Engine Speed Sensors

```
void setup
Set Pin 22, Pin 24 and Pin 26 as Digital Outputs
Set Pin 28 and Pin 30 as Digital Inputs
void loop
loop
    if $Pin28 == 0$ then
        $Pin22 = 1$
        $Pin24 = 1$
    end if
    if $Pin30 == 0$ then
        $Pin26 = 1$
    end if
    Wait $25\mu s$
end loop
```

Algorithm 13: Output Algorithm for Arduino

Table 6.3: Wiring Loom for Arduino

Arduino Pin	**Device**	**Colour**
22	555 U1 Pin 3 (VSS Output)	Green
24	555 U2 Pin 3 (Engine Output	Yellow
30	SW1 (Cranking/CAN Message Switch)	Purple
32	SW2 (Overrun Switch	Grey
36	Q1 (Cylinder 1 FET)	Blue
38	Q2 (Cylinder 2 FET)	Orange
40	LED1 (Overrun LED)	Green
A0	R5 Wiper (Accelerator Input)	White
A1	R6 Wiper (Engine Load Input)	Brown
CANRx	CANLo	Yellow
CANTx	CANHi	White
3.3V	Board Power	Red
0V	Board Ground	Black

Wiring Loom

Following these tests, the input and output wires were attached to a loom and connected to the Arduino through the I/O pins present on the PCB. The wiring setup can be seen in Table 6.3. The use of different colours allowed easy identification during the debugging process and checking of signals through oscilloscope or digital voltmeter probes if required.

Automotive Controller Algorithm Development

Following on from the completion of the initial tests, the above programs were integrated into a scheduling algorithm based on Algorithm 8 and incorporating Algorithms 5, 6 and 7. Due to the creation of these systems as primitives in the previous chapter, the algorithm was designed using techniques based on the Vienna Development Method (VDM) to ensure all jobs were called and scheduled correctly. The developed algorithm can be seen in Algorithm 14. As each component had been previously tested and the system had been successfully simulated in MATLAB, the design required minimal debugging. All job C_i and D_i from this chapter were used also, as these values were calculated using the above algorithms within the Arduino environment.

6.6.2 Energy Harvesting SPC Calculator

In Chapter 3, a suggestion of hardware implementation was made in Chapter 3. Due to the ease of development, rather than pursue a VHDL implementation of this design into an FPGA, it was decided to emulate Algorithm 1 within the Arduino environment. For this, a 32-entry Array was defined to store input values, with these coming initially from the built in `random()` function and later from a signal generator connected to a digital IO pin of the Arduino. The experiment aimed to identify the percentage error within the Arduino from the use of Integer numbers for Mean and Standard Deviation calculation, as well as

```
void setup
Set Following as Digital Inputs: Pins 22, 24, 30, 32
Set Following as Digital Outputs: Pins 36, 38
Set Following as Analogue Inputs: Pins A0, A1
Set Following as CAN Pins: Pins CANRx, CANTx
void loop
loop
    if $Pin24 \neq PreviousPin24$ then
        $PreviousPin24 = Pin24$
        $EngCounter++$
    end if
    if $EngCounter == 29$ then
        Schedule Algorithm 5 (Spark) on Pin 36
        Store $C_i$ and $D_i$ for job in Scheduler
    end if
    if $EngCounter == 58$ then
        $EngCounter = 0$
        Schedule Algorithm 5 (Spark) on Pin 38
        Store $C_i$ and $D_i$ for job in Scheduler
    end if
    if $Pin22 \neq PreviousPin22$ then
        $PreviousPin22 = Pin22$
        $VSSCounter++$
    end if
    if $millis() - VSSTimer == 10$ then
        $VSSTimer = millis()$
        Schedule Algorithm 7 (VSS)
        Store $C_i$ and $D_i$ for job in Scheduler
    end if
    if $Pin30 == 0$ then
        Schedule Algorithm 6 (CAN)
        Store $C_i$ and $D_i$ for job in Scheduler
    end if
    if $Pin32 == 0$ then
        Delay $1ms$
    end if
    Run Algorithm 8 (Scheduler)
    Wait $25\mu s$
end loop
```

Algorithm 14: Scheduling Control Algorithm

the α and β errors for identification of energy levels more than one, two or three Standard Deviations below the mean. Though the power consumption of the Arduino could not be directly controlled from the development environment, it was thought that this could act as a low power observer, only activating on arrival of a new value from the energy harvester through an interrupt, allowing the processor to enter a sleep mode in which it will draw $< 2.5\mu A$. The use of Arduino also reduces development time significantly as common mathematical functions such as squares and square roots are incorporated into the language, meaning no self-designed functions are required to create the SPC module.

6.6.3 Nash Equilibrium Calculator

To test the viability of the Arduino for the Game Theoretic Algorithm in Chapter 4, a Nash finder was created within the environment based on Algorithms 2 and 3. Two-dimensional arrays were used in place of the matrices within MATLAB, with location of the maximum values for column and row transposed into one-dimensional arrays that were compared for any matching values — meaning a Nash Equilibrium had been found. The algorithm also stored the maximum overall value for one job, allowing the prudent outcome to be used in the event of no Nash Equilibrium being found. Matrix values were manually loaded prior to compilation and the corresponding Nash coordinates outputted through the Arduino console window. If no Nash Equilibrium was present, the console window would show the coordinates of the prudent outcome for the first job loaded.

6.7 Results

6.7.1 Preliminary Tests

The circuits in Figure 6.6 were constructed on prototype board using discrete components from the electronics laboratory at Newcastle University. With all components fitted and running, total current drawn at 3.3V was 0.04A, well inside the capabilities of the Arduino for current sourcing. These components were connected to the pins specified in Figure 6.6.

Vehicle and Engine Speed Inputs

Within the test circuits, U1 and U2 are both present to provide different square wave inputs to the Arduino. Through using the 555 in an astable mode, the frequency of these waves (f) can be calculated using equation 6.1, giving a theoretical range of 583-3062 Hz for U1 and 3158 - 18947 Hz for U2. Results from practical tests of these circuits can be seen in Table 6.4. This shows values close to the expected, allowing these circuits to be used with the Arduino.

$$f = \frac{1.45}{(R1 + 2R2)C} \tag{6.1}$$

Ignition Control Circuit

Circuits Q1 and Q2 were constructed to allow switching of a small coil from a relay as a reduced scale spark system. When 3.3V was supplied from the power

Table 6.4: Output Frequencies for U1 and U2

Circuit	**Lowest Frequency/Hz**	**Highest Frequency/Hz**
U1	574	2647
U2	3230	219700

Table 6.5: Percentage Errors for use of Integer Calculation and α and β errors sampled using Arduino Random Number Generation Function

Item	**Error%**
Mean Error	3.28
Standard Deviation Error	0.11
α Error	0
β Error	0

supply to the gate through R5 and R6, the FET reached saturation within $30ns$. On switch-off, diodes D8 and D9 successfully dissipated the energy stored in the coil, allowing the coil to be ready for use within 1ms. From these results, the circuit was allowed to be used with the Arduino.

Other Circuits

Potentiometers R5 and R6 successfully gave readings from 0V to 3.3V, with switches S1 and S2 floating when open and reading 0V when depressed. LED1 was controlled by the Arduino using Algorithm 9, with Pin 13 connected to the external LED through the Header Connections. From all these tests, the full circuit was deemed to simulate operation of a vehicle adequately for the purposes of these tests, allowing these to be interfaced to the Arduino and this to be used as an engine ECU.

6.7.2 Energy Harvesting SPC Calculator

As Figure 6.8 shows, the Arduino handles SPC Calculation within $52\mu s$ and with a low error for Mean and Standard Deviation shown in Table 6.5. Further tests with the board sampling a Gaussian signal generator can be seen in Table 6.6, with a comparison of execution times and errors using a Mann-Whitney Test shown in Table 6.7. No significant differences ($P < 0.05$) are present for either execution time or error, showing that the Arduino is running consistently and with feasible results from both the on-board generator and an external source. When the benchmark calculation was moved to the beginning of the `void loop()` section, the execution time increased to a mean value of $85\mu s$ due to the extra time required to read and quantise an analogue input pin by the Arduino. Table 6.6 shows an increase in Standard Deviation error as peak to peak voltage decreases. This is due to the population becoming more densely packed and thus the Standard Deviation becoming smaller. For a steady state harvester with very small variance (such as a photovoltaic cell), this could lead

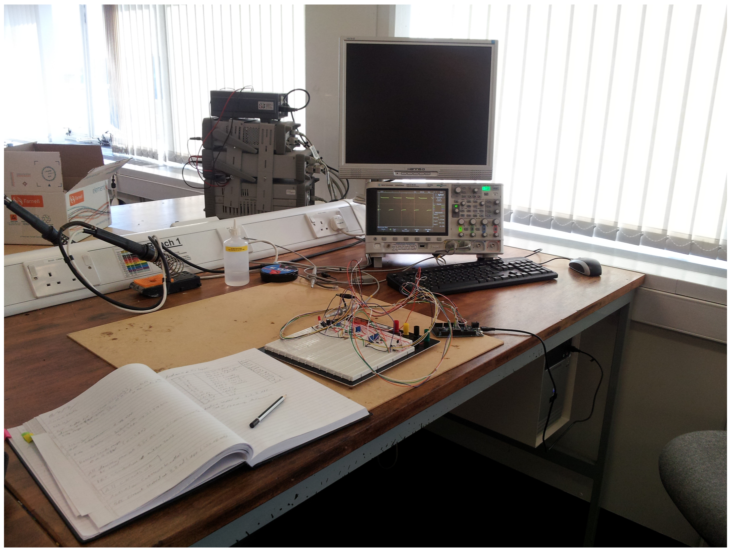
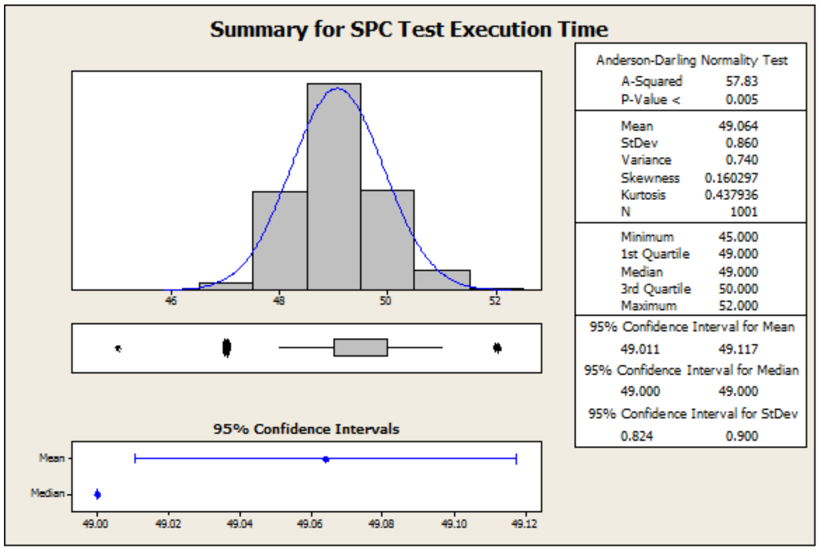


Figure 6.7: Arduino Connected to Test Circuits



Anderson-Darling Normality Test	
A-Squared	57.83
P-Value <	0.005
Mean	49.064
StDev	0.860
Variance	0.740
Skewness	0.160297
Kurtosis	0.437936
N	1001
Minimum	45.000
1st Quartile	49.000
Median	49.000
3rd Quartile	50.000
Maximum	52.000
95% Confidence Interval for Mean	
49.011	49.117
95% Confidence Interval for Median	
49.000	49.000
95% Confidence Interval for StDev	
0.824	0.900

Figure 6.8: Execution Time for 1000 runs of SPC Calculator using Arduino Random Number Generation Function

Table 6.6: Percentage Errors for use of Integer Calculation and α and β errors sampled using Gaussian Noise Generator Running at 3V, 2V and 1V Peak-to-Peak and sampling at 100ms

Item	**3V**	**2V**	**1V**
Mean Error	3.37	3.28	3.21
Standard Deviation Error	0.69	5.11	6.74
α Error	5.99	4.50	11.1
β Error	1.40	0.710	0.1

Table 6.7: Mann-Whitney Test Results for Execution Time, Mean Error and Standard Deviation Error Comparing Bench Run to Sample Run

Item	**P-Value**
Mean Error	0.40
Standard Deviation Error	0.52
Execution Time	0.81

to an overly aggressive energy management system, as α error increases from 6 percent to 11 percent through this reduction. This can be compensated for though increasing the buffer size to calculate populations from; though this would increase required resource, which is contrary to the goals of this circuit. A more suitable method, often used in safety control software, would be for two consecutive points to be logged as out of control to flag a fault. When this was done at 1V peak to peak, α error reduced to 4.1 %, reducing false occurrences of faults by more than half.

6.7.3 Automotive Controller

Preliminary Tests

After running the Arduino connected to the astable outputs for five minutes, the number of pulses recorded was compared to the calculated number expected. Results, which can be seen in Table 6.8, showed a small experimental error, believed to be due to loop delays in the created algorithm. These errors were below those deemed acceptable for use in Engine Speed or Vehicle Speed calculation, qualifying the Arduino for use with these areas of the Fuel ECU.

When the analogue inputs were read through the console window, values between 0 and 1024 were observed. This is in line with the Arduino specification, which states that Analogue Inputs A0 to A11 have ten bits of resolution as standard. To further test this, `analogReadResolution()` was changed within the program to test from eight to 12 bits resolution, giving ranges of 256 bits and 4096 bits respectively. For this work and to allow compatibility with other Arduino controller types, ten bits was deemed a sufficient level of clarity and continued to be used in later tests. This test also proved the viability of using clipping diodes within the circuit, as when 4V was supplied through an external

Table 6.8: Error in Engine and Vehicle Speed readings for Arduino at Differing Frequencies

Engine Frequency	**VSS Frequency**	**Expected Counter**	**Actual Counter**	**Error** Percent
600	4000	18000/1200000	17981/1198685	0.11/0.11
1500	10000	45000/3000000	44497/2999745	1.12/0.01
2500	20000	75000/6000000	74763/5989631	0.32/0.17

power supply, only the maximum value was recorded and 3.3V observed at Pin A0.

Depressing SW1 and SW2 successfully illuminated LED1 and presented a voltage at the gates of Q1 and Q2 respectively, demonstrating that the Arduino was capable of registering an active-low input. To further test this, an algorithm was created that illuminated the LED when SW1 was depressed, latched this value and would only extinguish LED1 if SW2 was depressed. When tested, this worked as specified, proving all areas of the Arduino and algorithms were suitable for use and allowing steady state Fuel ECU tests to be conducted.

Steady State Tests

Algorithm 14 was loaded into the Arduino Due, U1 and U2 were set to their lowest frequencies of 583Hz and 3158Hz respectively, R5 and R6 were set to 1.5V and the program was activated. LED1 did not illuminate, showing no overruns were present. When checked with an oscilloscope, the Gate Voltage of

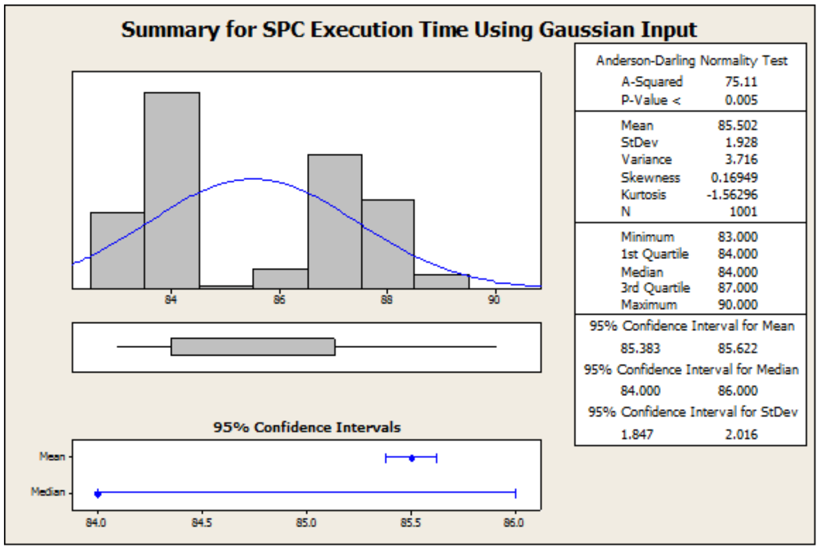


Figure 6.9: Execution Time for 1000 runs of SPC Calculator using Gaussian Noise Generator

Q1 and Q2 could be seen to rise in the same time as in section 6.7.1. When SW1 was closed, a CAN Pulse could be seen on Pins CANHi and CANLo and LED1 did not illuminate. From this, it could be concluded that the Arduino was operating correctly and successfully controlling the Spark, CAN and VSS Signals.

Increasing CAN Rate

To increase the CAN Rate, Pin 30 was disconnected from SW1 and placed onto a signal generator. This was set to give a square wave output and increase the number of CAN Jobs scheduled. The hypothesis for this was an increased CAN rate would cause a Spark job overrun and illumination of LED1. To test this, the signal generator was set between values of 100Hz and 1000Hz and LED1 observed for any illumination over a five minute period. To ensure this would be recorded, a 7432 Quad 2-input OR Gate was added to the circuit as seen in Figure 6.10. This would act as a latch between the Arduino output and LED1 to hold any brief voltage seen on Pin 40. Results for this can be seen in Table 6.9.

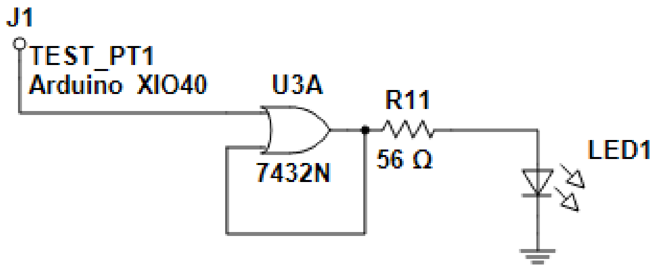


Figure 6.10: Circuit Modification to latch LED1 on in Event of Overrun

Table 6.9: LED1 Output for Increasing Value of Frequency at Pin 30

Input Frequency / Hz	**State of LED1**
100	Off
200	Off
300	Off
400	Off
500	Off
600	Off
700	Off
800	Off
900	Off
1000	Off

For all these values of frequency on Pin 30, the LED did not illuminate. Due to the layout of Algorithm 14, Spark jobs are scheduled before CAN jobs that occur on the same loop. Furthermore, due to the program running at $25\mu s$, it is thought that a great deal of jobs at the higher frequencies are filtered, with others removed by the heuristic nature of the algorithm, preventing an overrun from occurring due to CAN Overload. This was confirmed as depressing SW2 caused LED1 to illuminate, proving an overrun could be detected by the system.

Dynamic Tests

To test the effectiveness of Algorithm 14 in a real-life environment, a test using the MNEDC Cycle seen in Figure 5.5 was conducted. As only vehicle speed is given within this cycle, assumptions of other items such as engine load and accelerator percentage were made. These can be seen in Table 6.10.

Table 6.10: Assumptions for MNEDC Cycle

Situation	**Accelerator Percentage**	**Engine Load Percentage**	**Engine Speed/rpm**
Ramping Up	100	100	Relative to Vehicle Speed
15km/h Steady State	10	10	1000
30km/h Steady State	15	15	1200
50km/h Steady State	25	25	1500
70km/h Steady State	50	40	2000
100km/h Steady State	70	60	2500
120km/h Steady State	80	70	3000
Ramping Down	0	0	Relative to Vehicle Speed

As can be seen in Table 6.10, the level of complexity in controlling the MNEDC cycle is too great for the use of manual circuits such as those in Figure 6.6. Therefore, a second Arduino Due was borrowed from the School of Mechanical and Systems Engineering to provide these outputs for the MNEDC Cycle. The digital and analogue output pins on the board were configured to give the signals required for the MNEDC Cycle from Table 6.10, running at a $25\mu s$ loop. The ECU under test was connected to the second Due and the MNEDC Cycle run for CAN input frequencies of 100, 500 and 1000Hz, with LED1 observed for any illumination. Results for this test can be seen in Table 6.11.

Table 6.11: LED1 Output for MNEDC Cycle

Input Frequency / Hz	**State of LED1**
100	Off
500	Off
1000	Off

Once again, LED1 did not illuminate across any of the 20 minute test cycles. From this, it can be concluded that the Arduino Due can successfully control three real-time jobs at a variety of settings and loads in a practical environment.

6.7.4 Nash Equilibrium Calculator

Using examples within Chapter 2 of this Thesis, game matrices were placed into the arrays within the Arduino to determine whether Nash Equilibriums could be found. For the examples given (Figure 2.5, Figure 2.7 and Figure 2.9), the Algorithm successfully identified the correct Nash Equilibrium in a mean time of $14\mu s$. This showed that the calculation of Nash Equilibrium using the method in this Thesis was possible and could give an accurate result in adequate time.

6.8 Conclusion

This chapter provides support to Chapters 3 and 5 in showing that the Arduino Due, a device costing under 40 dollars, can be used to control a complicated system using previously developed libraries and the Heuristic Scheduler previously simulated in MATLAB.

With CAN Loading set to an appropriate level, the system runs continually with no overruns detected — unless SW1 is depressed to cause these. Owing to the design of Algorithm 8, the overrun is quickly dealt with by purging infeasible jobs from the queue, allowing a return to normal operation as soon as SW1 is detected as open circuit.

The heuristic algorithm successfully ran across the MNEDC cycle, a real-life assessment cycle used in engine development, with no overruns present; showing that the processor in the Arduino Due could be used in a practical environment and deliver correct operation. The tests conducted here would cover the Expanded Function, Common Operating Mode and Worst Case Tests for an automotive part under ISO26262, as seen in Table 2.4. Over Limit testing was not conducted due to the risk of damage to the Arduino Due, but could be performed by increasing the input voltages beyond 3.3V and setting the CAN delivery rate to a level beyond worst-case.

Owing to the optimisation carried out previously, combined with the testing of all software components both in MATLAB and as individual primitives during the Arduino development, the hardware implementation of all algorithms took place with virtually no debugging required. Therefore, this work shows, through practical implementation, that a low complexity embedded system is capable of running multiple complex benchmarks within an automotive environment. By using a hybrid approach of FCFS and heuristics, a significant reduction in the number of critical missed jobs can be seen for a minimal increase in runtime.

This chapter also proves the viability of the SPC Technique introduced in Chapter 3 and shows that, for a Gaussian System, it can identify low-energy situations with very high accuracy. This accuracy was further improved by making the system require two consecutive points below one standard deviation to occur before action was taken, giving a false occurrence error below five percent when this was added. To further this work, the designed system could be added to a real-life energy harvester and placed as the overseer for a circuit under test — when two points more than one standard deviation below the mean are identified, a simple digital output from the Arduino could be given to the operating circuit, instructing it to perform power saving; either through a technique such as DVFS or by removing low priority jobs from the wait queue, as suggested in Chapter 3. If successful, this design could be fabricated within an area of the circuit under test and allow power saving to be performed more effectively on the device.

Though only preliminary implementation of the Game Theory system was carried out, this chapter also shows that a Nash Equilibrium can be successfully calculated within the Arduino Environment, meaning that the testing of the scheduler designed in Algorithm 2 could take place. Proof that the Game Theoretic Algorithm operates successfully in a practical environment is therefore thought of as a key area of future research.

Chapter 7

Conclusion and Future Work

In the following year, therefore, London was set on fire in case anyone should have been left over from the Plague, and St Paul's Cathedral was built instead. This was also a Good Thing and was the cause of Sir Christopher Wren, the memorable architect.
W.C.Sellar & R.J.Yeatman, 1066 and all that

7.1 Conclusion

7.1.1 Importance of Work and Original Contribution

As previously stated in this thesis, the use of embedded systems in a variety of settings and environments continues to rise to improve product quality, efficient operation, traffic flow, air quality and many other areas of life. Often these uses are so small that people are unaware of their existence, but for this to continue, devices must become smaller, more power efficient and reliable to totally pervade our existences.

Through the chapters in this thesis, it has been shown that short, rule-based algorithms can successfully manage energy consumption and job schedules effectively to improve availability of microprocessor lifetime and prioritise job execution.

While statistical techniques, heuristics and game theory have been proven in their use through the literature, this work expands on much of the previously published findings and includes an investigation into Real-Time systems, which are of vital importance in many embedded system environments. Through submission at conferences and workshops, the concept of using rule-based algorithms has been seen as novel and a future direction for research into schedulers as devices increase in complexity. These techniques not only have been shown to improve availability through simulation, but also allow lower specification — and hence, lower cost microprocessors to be used in environments such as the Automotive ECU designed in Chapter 5. This work also showed usefulness in a multi-core environment, which is a key future area of research for all forms of scheduler, especially Real-Time systems.

7.1.2 Summary of Findings

Chapter 3 investigates the use of Gaussian Energy Harvesters in an Autonomous Hybrid system and finds that the rate of power delivery must be at least twice the power consumption rate to assure a system can give constant operation. While this seems counter-intuitive, a result such as this occurs due to the crossover between Gaussian distributions of similar sizes. Investigations in this chapter show that the use of Statistical Process Control to monitor the remaining energy stored can give *aposteriori* management that vastly improves operational life of the system for minimal sacrifice in execution time.

Chapter 4 shows the use of a Game Theoretic Model in scheduling a variety of job types on a single-core environment. Through careful game design and the use of Nash Equilibrium in a non-cooperative environment, availability of a system was improved over other queuing types through the use of an iterative algorithm. The use of Nash, combined with the game design, gave a near-optimal outcome for simple algorithm design and was deemed to be the optimal outcome given the time constraints placed on the system to create a feasible schedule.

Chapter 5 takes the ideas from the previous chapter and tests these using established embedded benchmarks. From tests using popular scheduler types, a heuristic scheduler was created that runs at higher job loading with less missed jobs, for a negligible increase in execution time. Through the use of design development tools such as Six Sigma, this algorithm is optimised to give maximum performance and tested using automotive standard tests. Again, through simulation, rapid prototyping and development of the system took place, giving a hybrid scheduler that outperformed its rivals.

The work from Chapters 3 and 5 were then placed into an embedded device for testing in Chapter 6. Once the optimal device was selected and input circuits manufactured, tests took place that validated the simulation results on a modern and commonly-used ARM Architecture. A simple Fuel ECU was constructed and placed onto automotive standard test cycles to determine performance and optimise the algorithms for use.

7.1.3 Limitations of Thesis

Within Chapter 4, the Game Theoretic Scheduler was only designed using Non-Cooperative Games using Nash Equilibrium to determine a suitable schedule. However, it has been proved that non-cooperative methods such as finding the Stackelberg Value or using Cooperative Games can give a better outcome than only using Nash. Therefore, testing the developed algorithm with a variety of alternate Game Theoretic Techniques would allow validation of the system and possible improvements over Nash.

The Game Theoretic Algorithm has only been tested in a single-core environment; with SPC and Heuristic Algorithms tested with only up to three cores. As discussed in Chapter 5, the testing of more than three cores was not conducted within this thesis due to the excessive computational time required, but the feasibility of this system for a many-core system would be a necessary development to assess its viability for the next generation of embedded systems, which are predicted to reach several hundred cores within the next decade.

Though benchmarks have been used in Chapters 4 and 5, only three were chosen to allow simplified testing and direct comparison within this thesis. While both Game Theory and the Heuristic Algorithm have been found to give some improvement over current scheduling paradigms, this investigation is very much in the early stages and requires more research to prove the effectiveness of both algorithm types.

Chapter 3 shows how SPC gives a viable improvement in availability for a system running with an energy harvester, but only considers Gaussian and Uniform Harvester types. While these are popular, the most common type used is Photovoltaic Cells, which operate with a delta function ideally and a heavily random delta function owing to the influence of artificial lighting and cloud cover predominantly. The energy management within this only considers situations where DVFS or other power management techniques such as clock gating are not available. This limits the proof of SPC to a conceptual realisation and more work is required in this area.

7.2 Future Work

In order to develop the work within this thesis further, the theoretical and simulation examples developed require further testing within a practical environment. For development of the SPC Energy Management program, this must be added to an existing multi-core embedded system within the scheduler and tested with commercial and developmental harvesters. Once the method has been found to be suitable, more advanced management techniques such as DFVS could be added and run at specific statistical results from the harvester output. Long term monitoring for Autonomous Hybrid Solutions such as solar arrays also could be considered to increase the lifetime of a system through managing power once the harvesting source goes into its regular cycle of famine.

A possible diagram for the hardware implementation can be seen in Figure. 7.1, which performs the SPC results and provides a coded output based on the level of energy management required. One concern within this is the use of Standard Deviation, which, through its use of square roots, would require a complicated logic circuit. The use of dividers and multipliers would also add complexity and reduce the speed of this circuit.

An architecture for this system was designed and can be seen in Figure. 7.2. This removes the need for multipliers by having a logical right-shifter. The complexity of the square root is also removed by placing the approximate values in a lookup table, rather than having them explicitly calculated each round. This also removes the need for floating point, as a comparison experiment between "true" values and the approximation created by this model had less than 1.6% error when only nearest integer values for mean and standard deviation were used.

In addition to this, as the system works as a heuristic model and building the matrix takes time, simulations of the game theoretic algorithm running with Stackelberg values and cooperative games would provide validation or improvement over the Nash Equilibrium system developed and allow testing of this further developed algorithm with a larger variety of job types — including those with precedence to evaluate performance of all Game Theory types in a more complex non-deterministic environment. These could also be tested, along with the Heuristic Algorithm, using a grid computing system to evaluate

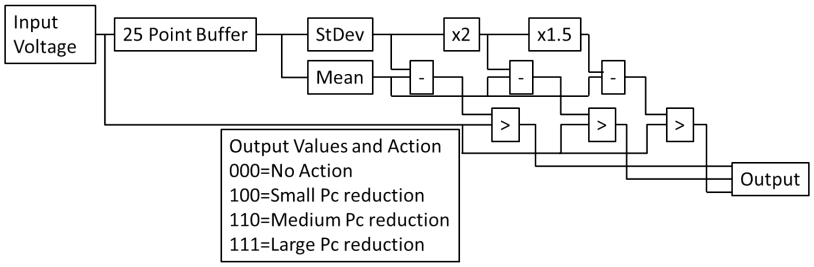


Figure 7.1: High Level Block Diagram of SPC Module

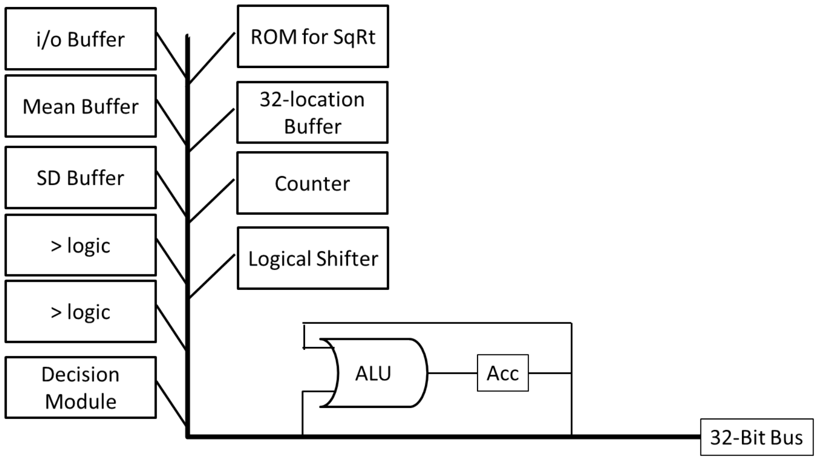


Figure 7.2: SPC Hardware Design

a full-lifetime run for many cores and determine whether this can outperform an expert system for availability conservation. A many core environment is necessary as the number of cores on a single piece of silicon is predicted to reach more than 300 by 2020. Therefore, tests of massive many-core layouts would be necessary to prove the versatility and usefulness of the Heuristic Algorithm against other more-advanced scheduling paradigms. While MATLAB is a useful tool for performing this, the run times for a 300 core simulation on a desktop computer would be excessive. One solution to this would be the use of a grid computer such as HTCondor to run the Monte Carlo simulation across a distributed system; thus reducing the test time and allowing validation of this concept on a massive many-core architecture. It is likely that a hybrid approach, using pruning and filtering techniques would give the most efficient performance, as certain situations would be managed by a standard schedule design, saving the use of heuristics for intensive or high-difficulty cases. Through this method, it is thought that the heuristic design could be preserved and used on more complicated systems without the need for redesign at each iteration.

Though the automotive fuel ECU was successfully constructed using an Arduino Due, this was only tested using simulated data signals from discrete devices. As only three jobs were tested simultaneously, the use of this device for successful replacement of current ECU's cannot be determined. This device requires more testing in a true automotive environment, with real-world components or signals captured from these in a hardware in the loop (HIL) test. This would demonstrate the capability of the system to interpret, calculate and output results to power ignition coils and manage all other jobs an ECU must control that were not considered in this chapter such as bus communication; but would be required for automotive qualification. Tests on a multi-core environment would also be required, to determine the true scalability of the Heuristic Algorithm. This work could lead to significant reductions in vehicle development time and major cost savings for automotive OEM's. Furthermore, the entire system would require testing to ISO26262 standards for an ASIL-D part, including HALT and endurance testing . If these tests were successful and proved the viability of the SAM3X8E ARM micro-controller for use in the Fuel ECU, automotive qualification would be required, through work detailed in Section 2.5.

Statistical techniques and job modelling have existed for many years and been used to improve capability and save time in various industries and roles. Work has often required definite proof whether an item is good or bad, or a difference has been noted before action will be taken. However, as this thesis shows, by adopting a rule-based system and accepting a near-optimal outcome over a truly-optimal one, we can improve performance and availability through time saving and allow a system to run successfully for longer than if we overburden it through searching for perfection.

Bibliography

[1] 205 Forum. Sample Ignition Graph, 2012. Available http://www.track-monkey.co.uk, Accessed 9 Aug 2013.

[2] Luca Abeni. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems*, (27):123–167, 2004.

[3] Ishfaq Ahmad. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–6, April 2008.

[4] Mustafa Imran Ali. *Design Considerations of Harvested-Energy Management*. Phd thesis, University of Southamption, 2012.

[5] S F Ali, M I Friswell, and S Adhikari. Piezoelectric energy harvesting with parametric uncertainty. *Smart Materials and Structures*, 19(10):105010, 2010.

[6] F Anceau. *The architecture of microprocessors*. Addison-Wesley, Wokingham, England ; Reading, Mass., 1986.

[7] Crispin Andrews. Raspberry Pi - retro computer of the future. *E&T Magazine*, March 2013.

[8] Arduino. Arduino, 2013. Available: www.arduino.cc . Accessed 1 August 2013.

[9] Atmel. SAM3X4/8E Datasheet, 2012. Available: http://www.atmel.com/images/doc11057s.pdf . Accessed 28th July 2013.

[10] Andrea Bartolini, Matteo Cacciari, Alessio Cellai, Manuel Morelli, and Andrea Tilli. Fault Tollerant Thermal Management for High-Performance Multicores. In Diana Marculescu and Suzanne Lesecq, editors, *Workshop on Micro Power Management for Macro Systems on Chip as part of DATE 11*, Grenoble, France, 2011.

[11] David Bellhouse. The Problem of Waldegrave. *Electronic Journal for History of Probability and Statistics*, 3(2), 2007.

[12] K G Binmore. *Fun and games : a text on game theory*. D.C. Heath, Lexington, Mass., 1992.

[13] Shekhar Borkar. Thousand Core Chips A Technology Perspective. In *Design Automation Conference*, pages 746–749, 2007.

[14] Robert Bosch. *Automotive Electrics and Automotive Electronics*. Wiley-Blackwell, London, 5th editio edition, 2007.

[15] Alexander Brevig. Scheduler Library, 2009. Available: http://playground.arduino.cc/Code/Scheduler . Accessed 5 July 2013.

[16] Forrest W Breyfogle. *Implementing six sigma : smarter solutions using statistical methods*. Wiley, Hoboken, NJ, 2nd edition, 2003.

[17] A Burns. Scheduling hard real-time systems: a review. *Software Engineering Journal*, 6(3):116, 1991.

[18] Giorgio C Buttazzo. *Hard real-time computing systems : predictable scheduling algorithms and applications*. Springer, New York, 2nd edition, 2005.

[19] Ewerson Carvalho, N Calazans, and F Moraes. Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs. In *Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on*, pages 34–40, 2007.

[20] S Chalasani and J M Conrad. A survey of energy harvesting sources for embedded systems. In *Southeastcon, 2008. IEEE*, pages 442–447, 2008.

[21] Hui Cheng. A High Efficient Task Scheduling Algorithm Based on Heterogeneous Multi-Core Processor. *2010 2nd International Workshop on Database Technology and Applications*, (3):1–4, November 2010.

[22] Hyeonjoong Cho, Haisang Wu, Binoy Ravindran, and E Douglas Jensen. On Multiprocessor Utility Accrual Real-Time Scheduling With Statistical Timing Assurances. In *In IFIP Embedded and Ubiquitous Computing (EUC*, 2006.

[23] D. Dal and N. Mansouri. Power Optimization With Power Islands Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):1025–1037, July 2009.

[24] Constantinos Daskalakis, Paul Goldberg, and Christos Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing*, 29(1):195–259, 2009.

[25] A David, K G Larsen, Li Shuhao, and B Nielsen. A Game-Theoretic Approach to Real-Time System Testing. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 486–491, 2008.

[26] Robert I. Davis and Alan Burns. FPZL Schedulability Analysis. *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 245–256, April 2011.

[27] G Despesse, T Jager, C Condemine, and P D Berger. Mechanical vibrations energy harvesting and power management. In *Sensors, 2008 IEEE*, pages 29–32, 2008.

[28] K Dietsche and M Klingebiel. *Bosch Automotive Handbook.* John Wiley & Sons Inc., Postfach, 7 edition, 2007.

[29] J Docherty, A Bystrov, and A Yakovlev. Identification of Key Energy Harvesting Parameters Through Monte Carlo Simulations. In D Al-dabass, editor, *UK SIM 2012*, volume 0, pages 486–490, Cambridge, UK, 2012.

[30] J Docherty, A Bystrov, and Alex Yakovlev. Simulation Testing of a Real-Time Heuristic Scheduler with Automotive Benchmarks. In D. Al-dabass, editor, *UK SIM 2013*, Cambridge, UK, 2013.

[31] Christof Ebert and Capers Jones. Embedded Software: Facts, Figures, and FutureTitle. *Journal of Computers*, 42(4):42–52, 2009.

[32] EEMBC. The Embedded Microprocessor Benchmark Consortium, 2013. Available: http://www.eembc.org/ . Accessed 17 June 2013.

[33] M.a. El-Gendy, a. Bose, and K.G. Shin. Evolution of the internet QoS and support for soft real-time applications. *Proceedings of the IEEE*, 91(7):1086–1104, July 2003.

[34] Mohammad Farsi and Manuel Barbosa. *CANopen implementation : applications to industrial networks.* Research Studies Press, Baldock, Hertfordshire, 1 edition, 1999.

[35] Firebox. HY Mini Wind Turbine, 2013. Available www.firebox.com . Accessed 8 August 2013.

[36] Michael John Fryer. *An introduction to linear programming and matrix game theory.* E. Arnold, London, 1978.

[37] Rick Gillman and David Housman. *Models of conflict and cooperation.* American Mathematical Society, Providence, R.I., 2009.

[38] Donald Gross. *Fundamentals of queueing theory.* Wiley, Hoboken, N.J., 4th edition, 2008.

[39] GSMArena. Intel announces dual-core 'Clovertrail+' Atom processors. *GSM Arena*, page 1, February 2013.

[40] Adnan Harb. Energy harvesting: State-of-the-art. *Renewable Energy*, 36(10):2641–2654, October 2011.

[41] Shaun Hargreaves Heap and Yanis Varoufakis. *Game theory : a critical introduction.* Routledge, London ; New York, 1995.

[42] V Hillier and P Coombes. *Fundamentals of Motor Vehicle Technology.* Nelson Thornes Ltd, Cheltenham, 4 edition, 1991.

[43] Qi Van Eikema Hommes. ASSESSMENT OF THE ISO 26262 STANDARD , ROAD VEHICLES FUNCTIONAL SAFETY . Technical report, Society of Automotive Engineers, 2012.

[44] Paul Horowitz and Winfield Hill. *The art of electronics.* Cambridge University Press, Cambridge England ; New York, 2nd edition, 1989.

[45] Nigel Howard. *Paradoxes of rationality: theory of metagames and political behavior.* MIT Press, Cambridge, 1971.

[46] J Hsu, S Zahedi, A Kansal, M Srivastava, and V Raghunathan. Adaptive Duty Cycling for Energy Harvesting Systems. In *Low Power Electronics and Design, 2006. ISLPED'06. Proceedings of the 2006 International Symposium on*, pages 180–185, 2006.

[47] IEEE. Harvest for the world [energy harvesting techniques]. *Power Engineer*, 20(1):34–37, 2006.

[48] Imagination Technologies. Imagination announces Minimorph development system for connected devices, 2011. Available www.imgtec.com . Accessed 7 May 2013.

[49] International Standards Organisation. ISO26262: Road Vehicles, Functional Safety, 2011. Available www.iso.org . Accessed 1 July 2013.

[50] ITRS. International Roadmap for Semiconductors, 2011. Available www.itrs.net . Accessed 8 August 2013.

[51] Alex Janek and Christian Steger. Power Management Strategies for Battery-driven Higher Class UHF RFID Tags Supported by Energy Harvesting Devices. In *IEEE Workshop on Automatic Identification Advanced Technologies*, pages 122–127, 2007.

[52] P Koch. How to interface energy harvesting models with multiprocessor scheduling paradigms. In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pages 21–25, 2009.

[53] Ilya V Kolmanovsky and Claude Dextreit. Game Theory Controller for Hybrid Electric Vehicles. *IEEE Transactions on Control Systems Technology*, (99):1–12, 2013.

[54] Hermann Kopetz. Event-Triggered versus Time-Triggered Real-Time Systems. In *International Workshop on Operating Systems of the 90's and Beyond*, page 16, Berlin, 1991. Springer-Verlag.

[55] Hermann Kopetz. *Real-time systems : design principles for distributed embedded applications.* Kluwer Academic Publishers, Boston, 1997.

[56] Alan Larson. *Demystifying six sigma : a company-wide approach to continuous improvement.* AMACOM, New York, 1 edition, 2003.

[57] Jinkyu Lee and Kang G. Shin. Controlling Preemption for Better Schedulability in Multi-Core Systems. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 29–38. Ieee, December 2012.

[58] Peng Li and Binoy Ravindran. Fast , Best-Effort Real-Time Scheduling Algorithms. *IEEE Transactions on Computers*, 53(9):1159–1175, 2004.

[59] Shuhui Li, Shangping Ren, Yue Yu, Xing Wang, Li Wang, Gang Quan, and Senior Member. Profit and Penalty Aware Scheduling for Real-Time. *IEEE Transactions on Industrial Informatics*, 8(1):77–89, 2011.

[60] Xueqiao Li, Shuang Liang, and Yuan Chen. Research and Improvement of Rate-monotonic Scheduling Algorithm. In *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on*, pages 66–69, 2010.

[61] Peder Lindberg, James Leingang, Daniel Lysaker, Samee Ullah Khan, and Juan Li. Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. *The Journal of Supercomputing*, 59(1):323–360, April 2010.

[62] Jason R. Marden and Adam Wierman. Overcoming limitations of game-theoretic distributed control. *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6466–6471, December 2009.

[63] MarketPublishers. Embedded Systems Market Reviewed & Forecast by GIA in Report. *Yahoo Finance*, page 1, May 2013.

[64] Measurement Specialties. Piezofilm, 2013. Available www.mess-spec.com . Accessed 8 Aug 2013.

[65] Maria Michalopoulou and Petri Mahonen. Game theory for wireless networking: Is a Nash equilibrium always a desirable solution? *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, pages 1249–1255, September 2012.

[66] Minitab. Minitab, 2013. Available www.minitab.com . Accessed 10 June 2013.

[67] Douglas C Montgomery. *Design and analysis of experiments.* Wiley, Hoboken, NJ, 7th edition, 2009.

[68] Gordon E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[69] C Moser, L Thiele, D Brunelli, and L Benini. Adaptive Power Management in Energy Harvesting Systems. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–6, 2007.

[70] Clemens Moser, Davide Brunelli, Lothar Thiele, and Luca Benini. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems*, 37(3):233–260, July 2007.

[71] Herve Moulin. On Strategy Proofness and Single Peakedness. *Public Choice*, 35(4):437–455, 1980.

[72] John Forbes Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[73] National Highway Traffic Safety Administration. Light-Duty Vehicle Greenhouse Gas Emission Standards and Corporate Average Fuel Economy Standards, 2010. Available www.nhtsa.gov . Accessed 1 April 2013.

[74] United Nations. United Nations Environment Programme, 2013. Available www.unep.org . Accessed 7 June 2013.

[75] Niki Nixon. Timeline: The History of Wind Power, October 2008. Available www.guardian.co.uk . Accessed 7 June 2013.

[76] Y. Nomura, M. Iwamoto, T. Yamanouchi, and M. Watanabe. A heuristics guided scheduling framework for domains with complex conditions. *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, pages 752–755, 1994.

[77] Malcolm Nunney. *Light and Heavy Vehicle Technology.* Butterworth-Heinemann, London, 4 edition, 2006.

[78] John Oakland. *Statistical Process Control.* Butterworth-Heinemann, Oxford, 5 edition, 2003.

[79] Milton Ohring. *Reliability and failure of electronic materials and devices.* Academic Press, San Diego, 1998.

[80] J A Paradiso and T Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1):18–27, 2005.

[81] Jaeok Park and Mihaela van der Schaar. The Theory of Intervention Games for Resource Sharing in Wireless Communications. *IEEE Journal on Selected Areas in Communications*, 30(1):165–175, January 2012.

[82] Moonju Park and Heemin Park. An Efficient Test Method for Rate Monotonic Schedulability. *IEEE Transactions on Computers*, (99):1–8, 2012.

[83] M. Parlar and M. Sharafali. Dynamic Allocation of Airline Check-In Counters: A Queueing Optimization Approach. *Management Science*, 54(8):1410–1424, August 2008.

[84] Giancarlo Medeiros Pereira, Miguel Afonso Sellitto, Miriam Borchardt, and Albert Geiger. Procurement cost reduction for customized non-critical items in an automotive supply chain: An action research project. *Industrial Marketing Management*, 40(1):28–35, January 2011.

[85] George Peters and Barbara Peters. *Automotive Vehicle Safety.* CRC Press, London, 2002.

[86] D Pimentel and P Musilek. Power management with energy harvesting devices. In *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, pages 1–4, 2010.

[87] William Poundstone. *Prisoner's Dilemma.* Oxford University Press, Oxford, 1 edition, 1993.

[88] Diego Puschini. *Optimisation dynamique et distribuée des architectures MPSoC basée sur la théorie des jeux.* Phd thesis, Montpellier, 2009.

[89] Howard Raiffa. *The Art and Science of Negotiation.* Belknap Press of Harvard University Press, Cambridge, MA, 1 edition, 1982.

[90] Raspberry Pi Foundation. Raspberry Pi, 2012. Available www.raspberrypi.org . Accessed 1 Aug 2013.

[91] Jessica Riskin. *Science in the Age of Sensibility The Sentimental Empiricists of the French Enlightenment.* University of Chicago Press, Chicago, 1 edition, 2002.

[92] Insa Rouen. WorkShop Audace : ISO 26262 and reliability. In *Analyse des causes de défaillances des composants des systèmes mécatroniques embarqués*, pages 1–17, Saint-Etienne-du-Rouvray, France, 2012.

[93] S Roundy, D Steingart, L Frechette, P Wright, and Jan M Rabaey. Power sources for wireless sensor networks. In *Wireless Sensor Networks*, pages 1–17. Springer, Berlin, 2004.

[94] M D Rowe, Min Gao, S G K Williams, A Aoune, K Matsuura, V L Kuznetsov, and Fu Li Wen. Thermoelectric recovery of waste heat-case studies. In *Energy Conversion Engineering Conference, 1997. IECEC-97., Proceedings of the 32nd Intersociety*, pages 1075–1079 vol.2, 1997.

[95] Kaushik Roy and Sharat Prasad. *Low power CMOS VLSI circuit design.* Wiley, New York, 2000.

[96] Barbara Ryan and Brian Joiner. *Minitab Handbook.* Duxbury Press, Belmont, 3 edition, 1994.

[97] W Saad, Zhu Han, M Debbah, A Hjorunges, and T Basar. Coalitional Game Theory for Communication Networks. *Signal Processing Magazine, IEEE*, 26(5):77–97, 2009.

[98] M. Saksena. Real-time system design: a temporal perspective. *Conference Proceedings. IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.98TH8341)*, 1:405–408, 1998.

[99] Alvin Scodel. Probability Preferences and Expected Values. *The Journal of Psychology*, 56(2):429–434, 1963.

[100] Sanjit a. Seshia and Alexander Rakhlin. Game-theoretic timing analysis. *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 575–582, November 2008.

[101] K.G. Shin. Statistical real-time communication over ethernet. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):322–335, March 2003.

[102] Sammy Shina. *Six Sigma for electronics design and manufacturing.* McGraw-Hill, New York, 2002.

[103] Y Shoham and K Leyton-Brown. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations.* Cambridge University Press, Cambridge, 1 edition, 2009.

[104] Southampton University. Holistic Energy Harvesting Group, 2013. Available www.holistic.ecs.soton.ac.uk/ . Accessed 12 August 2013.

[105] J.A. Stankovic and K Ramamritham. The Spring kernel: a new paradigm for real-time systems. *IEEE Software*, 8(3):62–72, 1990.

[106] Yi Su and Mihaela Van Der Schaar. A New Look at Multi-user Power Control Games. In *IEEE International Conference on Communications*, pages 1072–1076, 2008.

[107] Andrew S Tanenbaum. *Modern Operating Systems*. Pearson/Addison Wesley, Upper Saddle River, N.J., 3rd edition, 2009.

[108] Texas Instruments. Beagleboard, 2013. Available www.beagleboard.org . Accessed 7 May 2013.

[109] C.J. Tomlin, J. Lygeros, and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, July 2000.

[110] Lionel Torres, Pascal Benoit, Gilles Sassatelli, Michel Robert, Diego Puschini, and Fabien Clermidy. *An Introduction to Multi-Core System on Chip Trends and challenges*. Springer, New York, 1 edition, 2011.

[111] Ramona Trestian, Olga Ormond, and Gabriel-Miro Muntean. Game Theory-Based Network Selection: Solutions and Challenges. *IEEE Communications Surveys & Tutorials*, 14(4):1212–1231, 2012.

[112] Chi-Ying Tsui and Wing-Hung Ki. A Batteryless Vibration-based Energy Harvesting System for Ultra Low Power Ubiquitous Applications. *2007 IEEE International Symposium on Circuits and Systems*, pages 1349–1352, May 2007.

[113] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. John Wiley & Sons Inc., London, 2012.

[114] Janos Vegh, Jozsef Vasarhelyi, Jan Turan, and Daniel Drotos. The von neumann computer model on the mirror of new technologies. In *Carpathian Control Conference (ICCC), 2013 14th International*, pages 411–416, 2013.

[115] M Villalva, J Gazoli, and E Filho. Comprehensive approach to modeling and simulation of photovoltaic arrays. *IEEE Transactions on Power Electronics*, 24(5):1198–1208, 2009.

[116] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, Princeton, N.J. ; Woodstock, 60th anniv edition, 2007.

[117] Donald J Wheeler. *Understanding Variation: the Key to Managing Chaos* . Longman Higher Education, 1993.

[118] John Wilcock. The Staffordshire University Computing Futures Museum Cambridge Page, 2010. Available www.fcet.staffs.ac.uk/jdw1/sucfm/cambridge.htm . Accessed 3 June 2013.

[119] Michael Wooldridge. Does Game Theory Work? *IEEE Intelligent Systems*, 27(6):76–80, November 2012.

[120] G K Yeap and Farid N Najm. *Low power VLSI design and technology.* World Scientific, Singapore ; River Edge, N.J., 1996.

[121] Baoxian Zhao, Hakan Aydin, and Dakai Zhu. Reliability-Aware Dynamic Voltage Scaling for Energy-Constrained Real-Time Embedded Systems. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, volume 546244, pages 633–639, 2008.

[122] Guanghui Zhou, Pingyu Jiang, and George Q. Huang. A game-theory approach for job scheduling in networked manufacturing. *The International Journal of Advanced Manufacturing Technology*, 41(9-10):972–985, June 2008.

Index