Fault Management via Dynamic Reconfiguration for Integrated Modular Avionics

by

Peter David Hubbard

A Doctoral Thesis submitted in partial fulfilment of the requirements

for the award of Doctor of Philosophy of

Loughborough University

May 2015

i. Abstract

The purpose of this research is to investigate fault management methodologies within Integrated Modular Avionics (IMA) systems, and develop techniques by which the use of dynamic reconfiguration can be implemented to restore higher levels of systems redundancy in the event of a systems fault.

A proposed concept of dynamic configuration has been implemented on a test facility that allows controlled injection of common faults to a representative IMA system. This facility allows not only the observation of the response of the system management activities to manage the fault, but also analysis of real time data across the network to ensure distributed control activities is maintained.

IMS technologies have evolved as a feasible direction for the next generation of avionic systems. Although federated systems are logical to design, certify and implement, they have some inherent limitations that are not cost beneficial to the customer over long life-cycles of complex systems, and hence the fundamental modular design, i.e. common processors running modular software functions, provides a flexibility in terms of configuration, implementation and upgradability that cannot be matched by well-established federated avionic system architectures. For example, rapid advances of computing technology means that dedicated hardware can become outmoded by component obsolescence which almost inevitably makes replacements unavailable during normal life-cycles of most avionic systems. To replace the obsolete part with a newer design involves a costly re-design and re-certification of any relevant or interacting functions with this unit. As such, aircraft are often known to go through expensive mid-life updates to upgrade all avionics systems. In contrast, a higher frequency of small capability upgrades would maximise the product performance, including cost of development and procurement, in constantly changing platform deployment environments.

IMA is by no means a new concept and work has been carried out globally in order to mature the capability. There are even examples where this technology has been

implemented as subsystems on service aircraft. However, IMA flexible configuration properties are yet to be exploited to their full extent; it is feasible that identification of faults or failures within the system would lead to the exploitation of these properties in order to dynamically reconfigure and maintain high levels of redundancy in the event of component failure. It is also conceivable to install redundant components such that an IMS can go through a process of graceful degradation, whereby the system accommodates a number of active failures, but can still maintain appropriate levels of reliability and service. This property extends the average maintenance-free operating period, ensuring that the platform has considerably less unscheduled down time and therefore increased availability.

The content of this research work involved a number of key activities in order to investigate the feasibility of the issues outlined above. The first was the creation of a representative IMA system and the development of a systems management capability that performs the required configuration controls. The second aspect was the development of hardware test rig in order to facilitate a tangible demonstration of the IMA capability.

A representative IMA was created using LabVIEW Embedded Tool Suit (ETS) real time operating system for minimal PC systems. Although this required further code written to perform IMS middleware functions and does not match up to the stringent air safety requirements, it provided a suitable test bed to demonstrate systems management capabilities.

The overall IMA was demonstrated with a 100kg scale ‘Maglev’ vehicle as a test subject. This platform provides a challenging real-time control problem, analogous to an aircraft flight control system, requiring the calculation of parallel control loops at a high sampling rate in order to maintain magnetic suspension. Although the dynamic properties of the test rig are not as complex as a modern aircraft, it has much less stringent operating requirements and therefore substantially less risk associated with failure to provide service.

The main research contributions for the PhD are:

1. A solution for the dynamic reconfiguration problem for assigning required systems functions (namely a distributed, real-time control function with redundant processing channels) to available computing resources whilst protecting the functional concurrency and time critical needs of the control actions.
2. A systems management strategy that utilises the dynamic reconfiguration properties of an IMA to restore high levels of redundancy in the presence of failures.
3. A Demonstration of the operation of points 1 & 2 on a representative system, showing that dynamic configuration can occur whilst the service provision (i.e. real-time control action) is maintained.

The conclusion summarises the level of success of the implemented system in terms of an appropriate dynamic reconfiguration to the response of a fault signal. In addition, it highlights the issues with using an IMA to as a solution to operational goals of the target hardware, in terms of design and build complexity, overhead and resources.

For those who say ‘about time’, this is for you:

But most of all, for my Wife and Daughter.

i. Acknowledgements

I would like to thank both EPSRC and BAE Systems for funding this research activity. I also extend my thanks to my industrial supervisor, Matt Mapleston, and to my Academic supervisors Roger Goodall and Roger Dixon for supporting me through to the bitter end. Their support has gone above and beyond the expectations of supervisors and I hope I do your efforts service. Thanks also to my examiners for taking the time to read and critique this work in order to scientifically validate the findings.

Thanks to the support from my friends and family, both to those that 'got it' and to those that still don't really understand! I thank you all the same and I'm glad you are there. I'd like to thank my Wife, Ella, who has been through this with me from the start and offered her support throughout (with varying amounts of patience!), and I'm also going to thank my daughter, Elsie, even though she's just turned up.

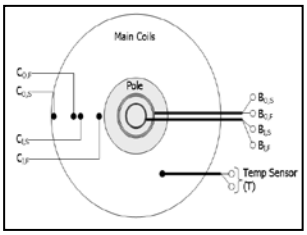
And finally, thanks to Rekha Patel, for providing a solution that no one else knew.

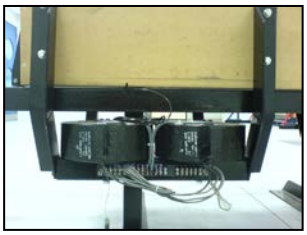
Pete.

iii. Contents

iv. List of Figures

v. List of Tables





CHAPTER 1:

Introduction

1. Introduction

This thesis addresses the concept of using dynamic reconfiguration within an avionic system to maintain high levels of redundancy in the presence of faults.

The concepts presented are based on the modern avionics architecture known as Integrated Modular Avionics (IMA). This solution to on-board computing requirements is designed to be flexible by specifying generic hardware resources for which 3rd party avionic applications can be developed. Compared to traditional avionics design, IMA reduces the complexity and cost of avionics systems by commonality of modules, flexible configuration options, and incremental update possibilities.

A further opportunity of IMA is exploiting the flexible configuration to extend operating periods of aircraft, even in the presence of failures. By optimising configuration using healthy parts of the system; sufficient levels of reliability could be maintained to retain operational performance. This thesis considers managing faults at the point of occurrence using dynamic reconfiguration (i.e. automatically re-configuring at run-time) to tolerate component failures.

1.1. Background

Future generations of avionics are transitioning from a traditional 'federated' design to flexible architectures that exploit benefits of modularity. These benefits are realised through potential savings in spares (via savings due to commonality of design), extended operating periods between maintenance (via abilities to 'gracefully degrade' and tolerate faults) and ease of upgradeability (via a software layering mechanism that removes dependency between hardware and software components).

A concept proposed and defined for the civilian sector by the standard ARINC 653 (ARINC, 2006), and in the military sector in DEF-STAN-00-78 (MOD, 2005), is that of Integrated Modular Avionics (IMA). The fundamental principle of IMA is that bespoke hardware and

software design for a specific function is replaced by generic processing units capable of delivering the same functionality by executing software 'applications'.

This format has the potential to be implemented using conventional systems design in that a configuration of hardware resources and software applications can be defined, with communications protocols and execution scheduling such that real-time task execution can be performed deterministically. This offers benefits over a federated design in that the hardware processing resources across the network can be similar, significantly reducing the spares and servicing requirements to maintain system availability.

To achieve further benefits of IMA, the flexible architecture can be exploited to reconfigure the system when failures occur. This benefit can be realised off-line (between operations) or on-line (during operation). In an off-line sense, a new configuration could be defined at the point of front-line servicing to extend operation between maintenance intervals. If reconfiguration could occur during run time, the system could adapt in order to maintain high levels of safety critical operation in the presence of failures.

1.2. Problem Statement

The introduction of IMA presents a number of opportunities, as previously highlighted, but also brings with it a number of challenges. Aircraft systems have to be certified to an incredibly high standard of reliability. This is a complex task that requires complete systems understanding to prove that common failure modes are not safety-critical or mission-critical. As such, the system design is 'frozen' early in the system lifecycle as changes to part of the infrastructure can lead to costly re-certification of the whole system whole. This makes the concept of reconfiguration (even in an off-line sense) a difficult one to introduce to the aviation industry. Current philosophies follow the idea of a 'multi-static' configuration, whereby a number of system configurations are designed and certified prior to operation. Such an approach enables some of the benefits of IMA to be seen, and allows some flexibility within the system configuration. IMA systems in service currently utilise this approach.

To achievement the full benefits of the flexible nature of an IMA, system configuration needs to be automatic. In such a case, the system would be capable of continually identifying the most effective allocation of applications to resources. This enables the restoration of high levels of operational capability shortly after the occurrence of a failure and a graceful degradation of the system such that reliability levels can be maintained by reconfiguration in the presence of component failure. Such an automatic configuration method would have to be robust (such that it does not implement bad configurations) and intelligent (such that functional and reliability requirements are not jeopardised by the configuration defined).

There are two fundamental principles that are evident when considering the problem of assigning avionic functions to system resource; that of assigning functions to specific resources (e.g. information input/output points), and that of ensuring point failures are not created by assigning redundant processing streams to the same physical resource. Alongside this is the temporal configuration problem, in that real-time control or data processing tasks need to occur concurrently such that the latest available data is used in calculations. Mismanagement of this can have implications such as reducing expected closed-loop performance specifications.

1.3. Contributions

Further to the problem analysis in the previous section, the main research contributions for this PhD are to:

1. A solution for the dynamic reconfiguration problem for assigning required systems functions (namely a distributed, real-time control function with redundant processing channels) to available computing resources whilst protecting the functional concurrency and time critical needs of the control actions.
2. A systems management strategy that utilises the dynamic reconfiguration properties of an IMA to restore high levels of redundancy in the presence of failures.

1.4. Objectives

In order to validate the contributions outlined in section 1.3, the following objectives have to be met:

1. Investigate relevant literature to determine current state-of-the-art in areas such as; IMA implementation, Fault management strategies within IMA, and methods for assigning avionic functionality to an IMA hardware installation.
2. Develop a method of automatic configuration/reconfiguration of required avionic applications to distributed process resources.
3. Design and develop a hardware test bed (i.e. a representative IMA system with appropriate 'middleware' and real-time communication strategy) to enable implementation of the automatic configuration/reconfiguration method.
4. Validate that the IMA implementation is capable of facilitating distributed real-time control operations with a range of functional topologies.
5. Verify the ability of the configuration method to appropriately assign avionic applications to available resources using a range of functional topologies and available modules.
6. Verify the ability of the IMA system to reconfigure a distributed real-time control functional set (following a fault injection) to an allocation that tolerates component failure.

1.5. Publications

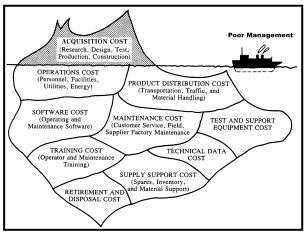
The following are publications as a result of the work detailed within this thesis.

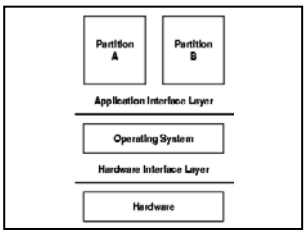
Hubbard, P., Goodall, R., Dixon, R., & Mapleston, M. (2008). Integrated Modular Systems for Maglev Vehicle Control. In *The 20th International Conference on Magnetically Levitated Systems and Linear Drives (MAGLEV 2008)*. San Diego, USA.

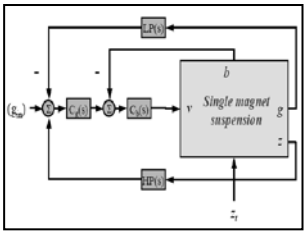
Hubbard, P., Mapleston, M., Goodall, R., & Dixon, R. (2008). Integrated Modular Processing for High Performance, High Integrity Control (IMPPIC). In *EDCC-7 Seventh European Dependable Computing Conference*. Kaunas, Lithuania.

1.6. Thesis Overview

This thesis details the key findings and results from the objectives listed in section 1.4. Chapter 2 contains a review of appropriate literature to identify the current capability of fault management within IMA, and other appropriate technologies and methods of similar areas that may have useful input to the study. Chapter 3 considers the findings of Chapter 2 and defines requirements in order to enable validation of the developed system and ensures tests are appropriate for proving the research objectives. Chapter 4 presents the real-time task control task (namely air-gap control of a Maglev vehicle) that will be the subject of the IMA. The purpose of this is to provide a complex control problem of an open-loop unstable system that should remain controlled in the presence of IMA component failure. Chapter 5 details the development of the IMA system, including the method used for configuring and reconfiguring avionics applications to available processing resources. Chapter 6 presents validation of the configuration method by analysing a series of configuration tasks and testing the timing properties of the system to ensure distributed real-time processing is occurring. Chapter 7 includes the results of a series of laboratory tests to observe the system response to IMA component failures. The goal of these tests is to show that after an occurrence of a failure, the system can reconfigure to regain previous reliability levels, whilst maintaining service provision. Chapter 8 documents the conclusions and main findings of the thesis.







CHAPTER 2:

Literature Review

2. Literature Review

IMA is a large topic branching into different disciplines and areas of interests resulting in a potentially large and diverse reading area. Figure 2-1 below shows the major areas of research undertaken and how they overlap to form common ground. Although this figure represents the main areas of study, it is not a definitive list of topics.

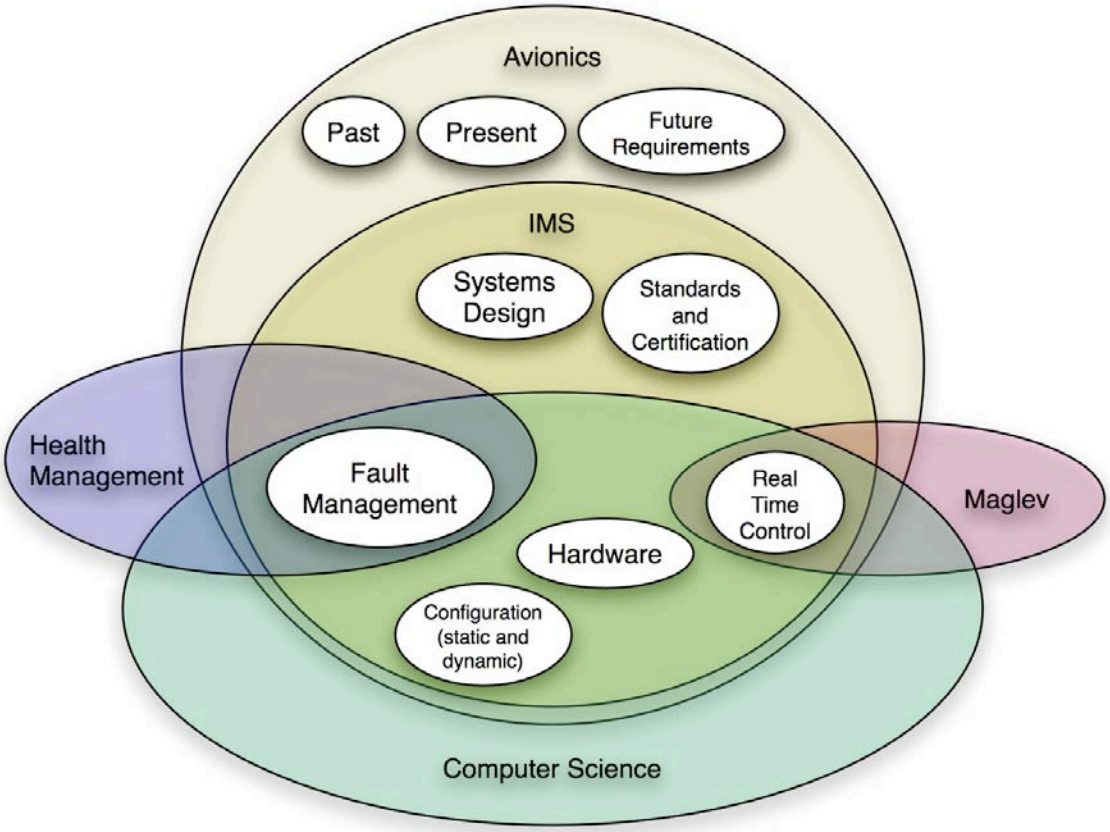


Figure 2-1 Research Topics

The following sections are the findings taken from academic publications, text books, journal articles and other sources of open information that are pertinent to the above topics.

2.1. The Developing Need for Improved Avionics

Over the past few decades, aircraft avionics systems have developed at a staggering rate into highly capable, costly and complex systems. This development has been fuelled by an end user requirement to exploit the potential capability offered by digital electronic devices and by a desire to constantly reduce the cost of maintaining a complex aircraft system throughout its lifecycle.

This section outlines these drivers and summarise the related future needs of aircraft systems.

2.1.1. Digital Revolution in Aircraft Systems

The rapid development in avionic system capability has been fuelled by the explosion of digital technology throughout the latter half of 20th century and to the current day. This fundamental relationship has been highlighted in a number of texts (Collinson, 2011; Moir, Seabridge, & Jukes, 2003, 2006; Moir & Seabridge, 2008).

World War II provided the first major drive for electronics to be embedded on board aircraft. These first systems were analogue based and relied on linear relationships for communication and processing of data throughout the system. Due to the extreme environment of an aircraft, these systems were subject to large amounts of drift and non-linearities. The alternative to these systems evolved during the 1950s and 1960s as the transistor replaced the thermionic valve which eventually led to the widespread use of digital electronics in the 1970s. The advent of micro-electronics and the capabilities that subsequent digital avionics systems could offer finally proved to be a better solution to providing the capabilities required for aircraft. As integrated circuits continue to follow development trends as highlighted with ‘rules of thumb’ such as Moore’s Law (Moore, 2006), highlighted in Figure 2-2, we can expect to see aircraft systems continue to provide increasing performance and capability.

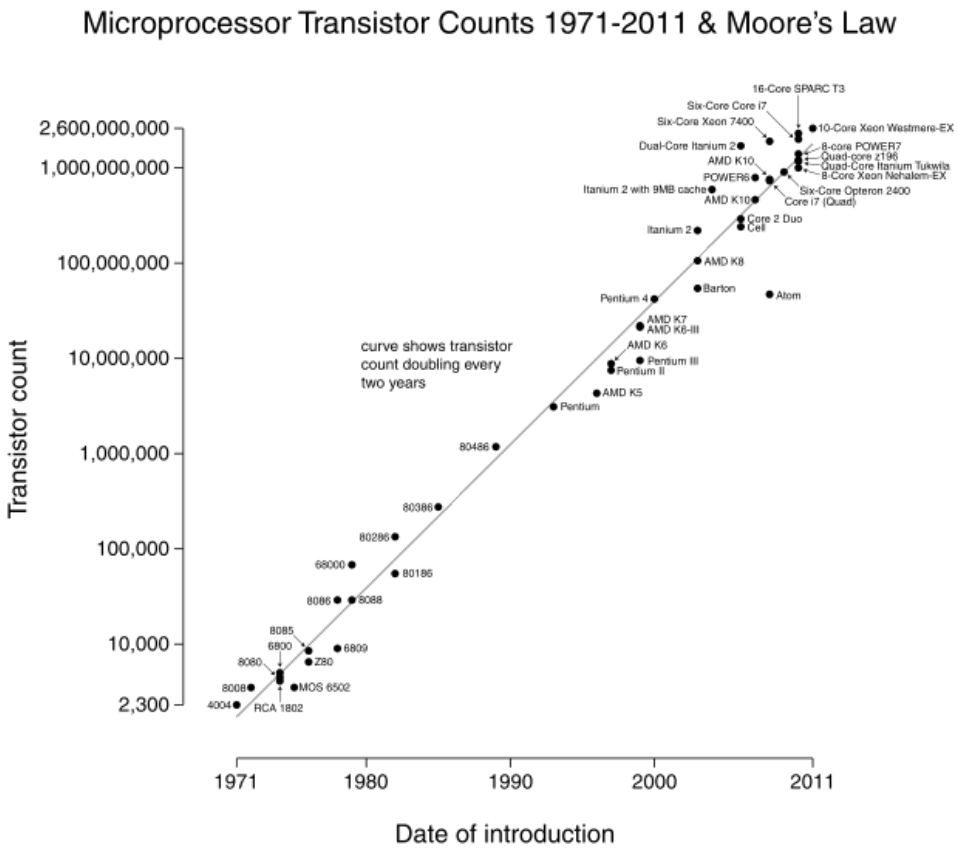


Figure 2-2 Historical Analysis of Moore's Law

In the current day, high performance digital technology not only offers the means to perform increasingly complex tasks within each individual subsystem, but to integrate avionic subsystems effectively creating the capacity to perform even more capable and complex functions.

2.1.2. The Development of Avionics Integration

The following is a summary of the work done by Zhang, Moir and Collinson (Collinson, 2011; Moir et al., 2006; Zhang, Pervez, & Sharma, 2003) who have all produced extensive works with regards to the development of avionics technologies.

Avionics have the potential to be integrated in different ways to yield different benefits. In the 1970s, a non-profit making organisation called Aeronautical Radio Inc. (ARINC) suggested that avionics units should have commonality in terms of form, fit and functionality – commonly known as F^3. The critical aspect of this standardisation is that a

Line Replaceable Units (LRU) that meets the F^3 specification can be directly replaced by another even if the replacement has different interior electronic implementation, or even manufactured by a different company. This specification lead to the terminology of Line LRUs or LRIs (Line Replaceable Items) which refers to modules that conform to the F^3 criteria. The benefits of this standardisation are widespread. It allowed for an open market place to exist for avionic components as well as simplifying the way spare units are managed.

In addition to the integration of the physical aspects of LRUs, it was widely recognised that passing data between aircraft systems would be a powerful capability. The sharing and fusion of data from different sources allows information to be inferred that may not be directly available from a single source. A classic example of this is the use of multiple sensors for object tracking (Varshney, 1997). In order to manage the sharing of data, it became clear that a standard was required for the data interaction of avionics.

During the early phase of this development, items were connected by a single source, single sink (or point to point, hard wired) connection wherever a communications channel was required. Although effective and logical to a degree, this format found limitations as the required amount of data passing between avionics functions increased to meet demands for more complex capabilities.

In order to address this issue and to provide consistency across the aerospace community, a number of standardised formats of connectivity between LRUs were defined. This section will look at three examples of these methods, namely ARINC 429, MIL-STD 1553 and the ARINC 629 specifications.

The ARINC 429 databus is a single source, multiple sink (SSMS) communication concept. The fundamental concept of a SSMS is that one node can send data to a number of recipient equipments via a digital serial link. This solution was adopted in aircraft such as the Boeing 757, 767 and the Airbus A300 and A310

The main limitation of the ARINC 429 databus is that it is a half-duplex design via a point to point communication strategy. In this topology each communication link requires dedicated hardware at both the source and sink of the transmission plus a dedicated cable to transfer the message. A difficulty with this system is that this set of components can only facilitate the transfer of a message in a single direction. For a reply signal to be sent, a second communication link inclusive of a duplicate set of hardware is required.

As systems using this topology grew in complexity and more information was required to be shared, the length and weight of cabling to facilitate these transfers grew substantially. For example, the Boeing 767-200ER contains approximately 90 miles (145km) of electrical wiring, clearly a huge maintenance issue. Furthermore, if the requirements of a system such as this were to change and extra LRUs are required, the redesign, refit and recertification of the system is a complex task.

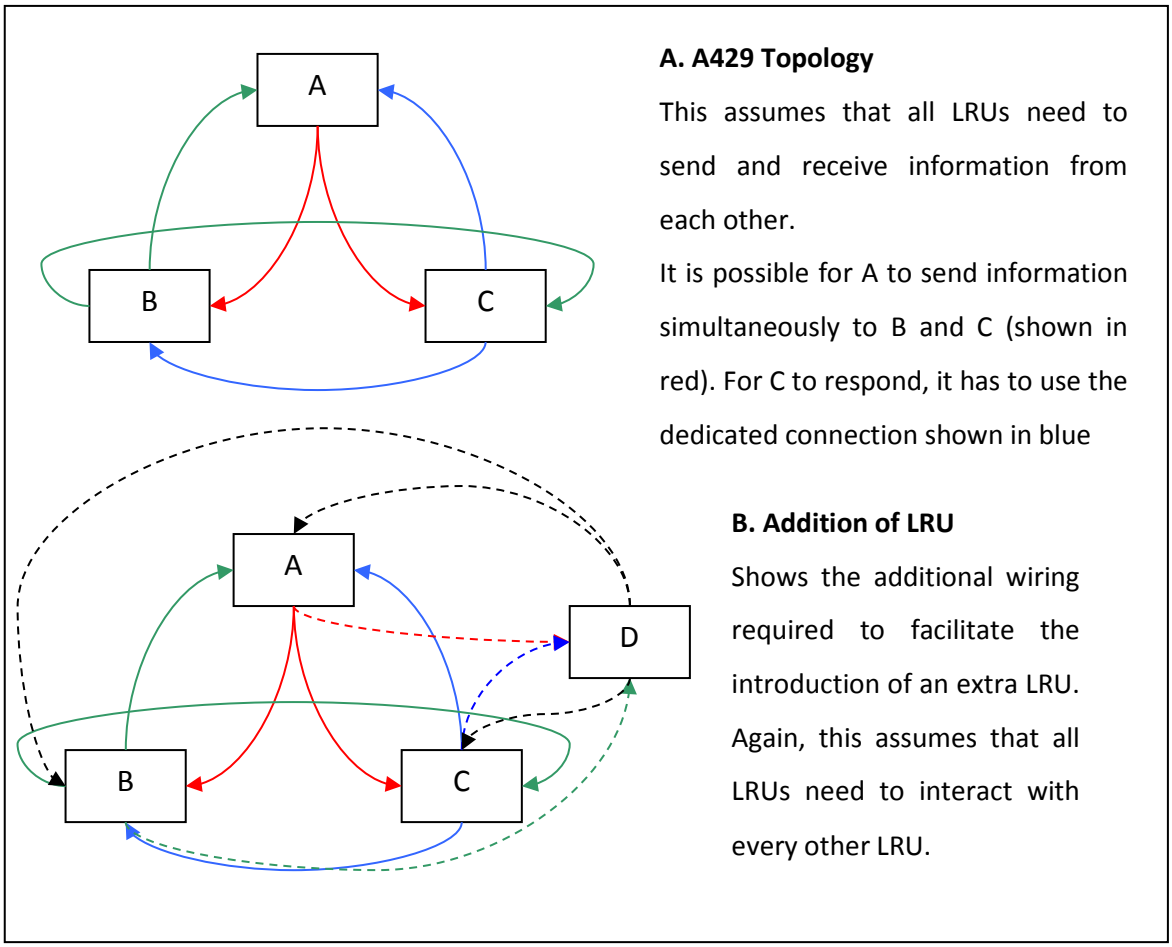


Figure 2-3 ARINC 429 Example

An alternative developed to this communications methodology in the military avionics domain is the MIL-STD (Military Standard) 1553 databus. Although the specification for the MIL-STD 1553 occurred slightly before the ARINC 429, its design has stood the test of time and is still widely popular in military avionic systems.

The MIL-STD 1553 is a true data bus. It is a half-duplex system, which means that although communication can occur in both directions along the same communication channel, they can only be sent one direction at a time. The advantage the 1553 bus has over the ARINC 429 is that this is a Multiple Source Multiple Sink (MSMS) method meaning that each node can send a message to a number of nodes connected to the bus. The general bus structure of the MIL-STD 1553 is shown in Figure 2-4 below.

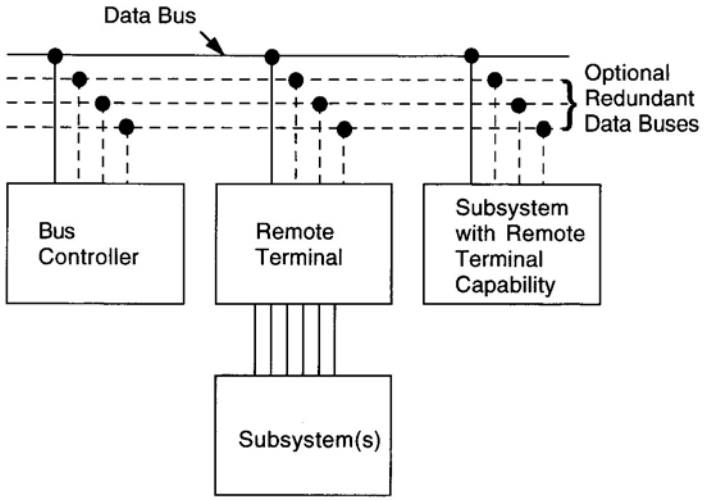


Figure 2-4 Multiplex databus system architecture (Moir & Seabridge, 2008)

The 1553 communications is controlled from a single node called the bus controller. The communication strategy in terms of data size, sender, recipient and timing for each data package for the particular avionics configuration implemented is predetermined and uploaded onto the controller. This preparation allows each configuration to be tested and verified before implementation via the use of software tools to ensure the communication meets concurrency and timing criticality requirements.

A further feature of the MIL-STD 1553 is the concept of multiplexing. In Figure 2-4, it can be seen that there is an option to include parallel channels of identical capability for redundancy purposes.

The ARINC 629 data bus is similar in concept to the MIL-STD 1553 bus as it is a true data bus, i.e. half duplex and MSMS. It is however of newer specification than the 1553 and offers more benefits, such as the capacity for 131 terminals against the 1553's 31 and a data transfer rate of 10 Mbit/sec compared to 1Mbit/sec. This particular system has been largely brought into popularity by the Boing 777 aircraft.

The main advantage the MIL-STD 1553 and the ARINC 629 architecture have over the ARINC 429 is the relative ease of adding new components. As mentioned before, for the ARINC 429, this requires a great deal more connections to satisfy the communication criteria. However for the true data buses another component can simply be added onto the length of the bus. This is certainly an advantage when it comes to modifying or reworking the system but does not remove all the complications with regards to re-testing and re-certifying.

2.1.3. Future Needs for Avionics Systems

There is a continuing drive from the aerospace industry to improve the quality and cost effectiveness of avionics across the entire systems lifecycle, inclusive of design, maintenance and upgrade costs. Avionics in general account for 30% of the cost of a new aircraft (Collinson, 2011) and carry a further burden with the rest of the lifecycle costs in terms of maintenance during operation, the resolution of faults and the cost of recertification should an upgrade be required. Figure 2-5 highlights the overall lifecycle cost of an aircraft in the form of an iceberg. Although not to scale, it shows how the acquisition cost of the aircraft is merely the 'tip of the iceberg' and the real cost lies in the support required to keep the platform available for use for as much time as possible. It can be inferred from here how the development, maintenance, support and upgrade of avionics systems relate to many of the different areas highlight in the diagram.

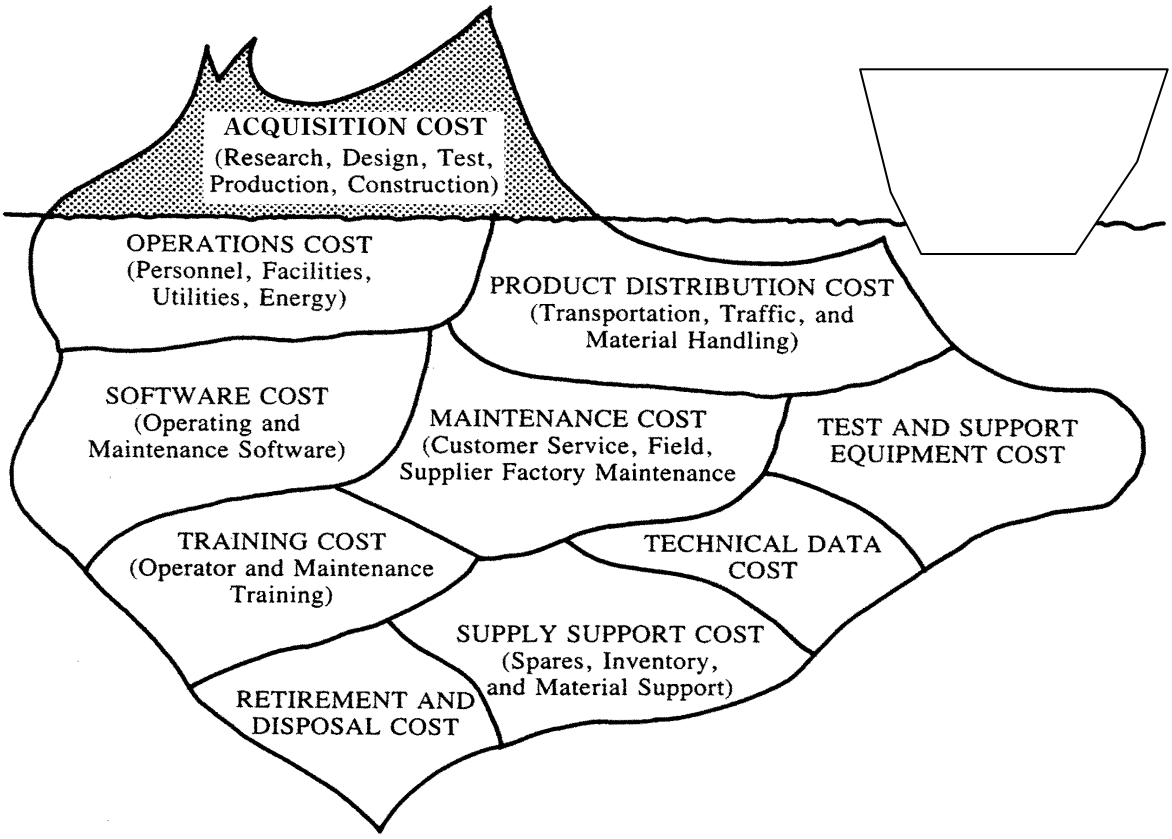


Figure 2-5 Aircraft System Lifecycle Cost (Mapleston, 2006b)

Aerospace companies are therefore seeking to identify the causes for the high cost of avionics, and find ways of reducing this.

One of the causes of high avionics cost is that as customers request incremental change to existing designs or products to exploit new technology, bespoke platform specific solutions are developed to meet this need (Johnson & Omiecinski, 1998). As a result, there is little commonality between different aircraft types, resulting in spiralling costs associated with avionics equipment. To drive down this cost a requirement for commonality between systems and subsystems exists to allow the re-use of components.

A further contributing factor is identified as the early stages in the project at which aircraft design is fixed (Little, 1991). As part of certification, avionics systems designs are frozen at an early stage of the procurement cycle. This defines purchasing schemes, system designs, etc, that allow the system build to commence. As mentioned previously, by the time the

system is in service the requirements have usually changed. In addition to this, the technology available to build the system will have progressed and be more capable. The inherent problem of attempting to introduce new requirements and technologies, however beneficial they may be, is that the systems design, purchasing schemes and certification stages may no longer be valid and have to be repeated in order to clear the aircraft to fly. In addition to new technology or capability requirements posing problems to designs, the problem of component obsolescence is also a substantial issue. The lifecycle of an aircraft from initial design to retirement can be the order of 50 years. Even during the period from the initial design to manufacture, electronic parts often become outdated and stop being produced. This causes problems with the purchase of spare parts for the aircraft further down the lifecycle.

It is suggested (Johnson & Omiecinski, 1998; Little, 1991) that a modular approach to avionics design offers inherent solutions to these problems. The idea of a common processing unit, capable of executing a number of aircraft functions, would solve a number of issues. Firstly, if a generic processing module is implemented as a replacement to bespoke LRUs, these can be a common component not only throughout a single aircraft system but across all aircraft in the fleet, or potentially across all military and civil platforms. Clearly the knowledge base required to sustain these component would be drastically reduced, simplifying the complexity of maintenance. Secondly, if the software is suitably decoupled from the hardware, the software could be re-implemented quite simply on upgraded hardware, or vice versa. This removes some problems with obsolescence and upgradeability throughout the lifecycle. In addition, life cycle cost will be minimized by functional integration as it reduces the amount of duplication of hardware and software elements (Morgan, 1991). In line with this theory, Line Replaceable Modules (LRMs) are becoming increasingly popular with avionics design as a replacement to LRUs. LRMs are designed to provide a solution to the modularity paradigms that LRUs are unable to. This modular concept of design is studied in detail in section 2.2.2.3.

The potential cost benefits of implementing a modular system is highlighted by Little (Little, 1991):

"It has been estimate that if LRMs were substituted for today's black boxes or LRUs in a wing of 72 F-16 A/B fighter aircraft, a 50% reduction in the flight line avionics maintenance personnel could be made together with the elimination of the intermediate workshops. This represents a total manpower saving of 109 personnel, a reduction from 180 to 71. The same analysis showed a reduction in spares types from 437 to 43 and an increase in avionic system MTBF from 7.3 hours to 35 hours."

The key elements identified in the quote above are:

1. The quoted time for MTBF (Mean Time Between Failure). Having aircraft available to fly rather than sitting dormant in a hanger is a key capability for military situations and a huge cost benefit to civil aircraft. This increase in reliability of the system makes the solution better value to the customer.
2. The reduction of intermediate workshops results in more availability of the aircraft in that the system can be service and repaired on the front line. This means that the unit does not have to return back to a main headquarters to be repaired and made available.
3. The reduction of spares not only simplifies repair jobs, but reduces the logistic complexity of taking a military system to the front line, or maintaining a civil aircraft away from main base. The support infrastructure can be reduced resulting in another potential cost saving to the customer.

It is also suggested (Little, 1991) that military customers more specifically have three main requirements to utilise the opportunity for modular architectures:

a) They must provide the desired mission capability. In particular they must allow the myriad datastreams obtained from the platform sensors to be more effectively integrated for the pilot to improve their situational awareness. Moreover, the avionics must be adaptive not only to accommodate new technology when appropriate but also more importantly to cope with a rapidly changing threat.
b) They must be affordable. In particular life-cycle costs must be a design driver from initial concept through to, and including, operational service.

c) They must exhibit improved reliability, maintainability and availability. This is directly related to life-cycle costs which currently are dominated by maintenance costs, and the operational need for a flexible but sustained maintenance response remote from main operating bases.

A solution to the one or two of the above requirements can be found using conventional methods, but only a modulated solution can provide a solution to all three.

2.1.4. Summary of Future Avionic Requirements

The top level needs for future avionics systems have been highlighted from two main sources (Little, 1991; Sutterfield, Hoschette, & Anton, 2008). The first requirement set suggested (Little, 1991) are:

- Adaptive technology in order to meet changing mission requirements or changing technology
- Affordable through the lifecycle
- Exhibit improved reliability, maintainability and availability

In order to facilitate the growing capability desires of the customer, and to facilitate the overall needs highlighted, the expected technology requirements are to be (Sutterfield et al., 2008):

- Modular hardware operating at processing speeds more than two orders of magnitude great than today
- Dense multiplexing buses
- Multi-core modular processors
- Conform to open standards (F^3)

The methods by which these capabilities can be implemented, associated with the design challenges involved, are discussed in the following sections.

2.2. Integrated Modular System Concepts

Integrated Modular Systems (IMS) have been identified as a potential concept that can meet the requirements of future avionics systems as highlighted in the previous section. The overall design principle is to utilise modularity throughout the system in order to move away from the tightly coupled/federated systems design of previous years.

This section aims to introduce the concepts of IMS and describe the work done to date in the development of systems

2.2.1. Introduction to IMS

IMS can is also referred to as Integrated Modular Avionics (IMA), or Integrated Modular Architectures (also IMA). These systems offer conceptual solutions for future architectures for many aerospace programmes, with some limited IMS architectures already implemented (such as the Airbus A350 and the Boeing 777). IMS concepts are replacing current avionics systems as they have the potential to overcome many issues highlighted in the previous sections.

The concept of IMS was formalised in ARINC report 651 entitled “Design Guidance for Integrated Modular Architecture”. The concept is one formed around using powerful computing modules that provide resource for the independent processing of application software (Prisaznuk, 1992). This is facilitated by an appropriate operating system that allows different applications to operate side by side on a single module.

Conmy (Conmy, 2006) provides a good introduction to the IMA concept. In this reference, the term Integrated Modular Avionics is introduced as:

> *“…a blanket term used to describe a distributed real-time computer network aboard an aircraft. This network should consist of a number of computing modules capable of supporting numerous applications of differing safety criticality levels.”*

These sources highlight the idea of a distributed yet integrated modular architecture, a statement which at first appears to contradict itself. However, the processing modules can

be connected throughout the aircraft by high speed databuses (such as ARINC 629), then the functionality of the different processes can be functionally integrated, albeit physically distributed (Jolliffe, 2005; Prisaznuk, 1992; Watkins, 2006).

One of the fundamental benefits of IMA is the modularity with which the system can be designed, built and maintained. The idea is that a system can be built to a functional specification using or re-using generic building blocks. This applies to the software applications as well as the hardware component. It is suggested (Field et al., 1997) that IMA consists of 5 'building blocks' that could be used to create any sized architecture, namely:

- Software – the software architecture
- Hardware/Modules – the hardware architecture
- Packaging – the environmental conditions of the hardware, the cooling and the power conditioning
- Data transmission – the communication network
- Low and Medium bandwidth interfacing to sensors/effectors

In summary, IMS provides a capability for interchangeable software and hardware components with interfaces well specified by open standards. IMS provides a good option to quickly reconfigure the allocation of functions either statically to meet new or changed deployment requirements or dynamically to restore higher levels of redundancy should faults occur.

The current state of the art IMA enables multiple unrelated applications, with different criticalities, to share the same computational platform without interference. The design challenge remains to map platform system and subsystem level constraints in timing, safety and security. (Gaska, Watkin, & Chen, 2015)

2.2.1.1. Modular Hardware/Software Integration

An important aspect of IMS is to remove the closely coupled nature of hardware and software which has been a part of avionics for a long time (Collinson, 2011; Field et al.,

1997; Morgan, 1991). The applications still require appropriate hardware to execute the software commands and fulfil function, but it is necessary to make the relationship loosely coupled.

The generally agreed solution to this is to use a software architecture known as the ‘three layer stack’, which is a simplified version of that found in (Prisaznuk, 1992).

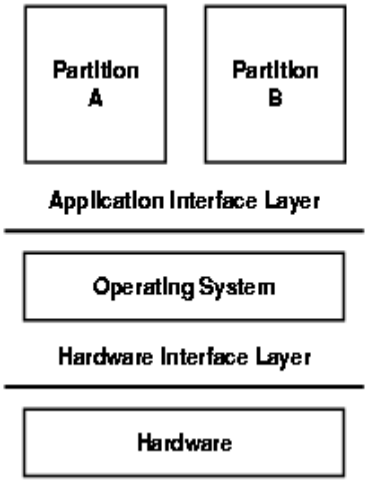


Figure 2-6 'Three Layer Stack' Model

The three layer stack is a fundamental principle for IMS as it provides the mechanism for the implementation of modular software upon modular hardware by facilitating well defined interface layers between the two. If this interface layer between hardware and software can be adequately implemented, the development of software can be performed without any direct knowledge of the hardware component. The hardware then becomes an unseen service provider of resource, such as facilitating communication or execution of functions. This is termed ‘hardware transparency’ (Conmy, 2006).

As applications are intended to be hardware transparent, the underlying hardware can theoretically be upgraded or replaced without affecting the design and source code of the application software. Conversely, an application can be upgraded or incrementally changed without directly affecting other applications, or requiring a hardware change. Furthermore, different companies will be capable of the production of parts of the system promoting

ease of development and competition for provision. Currently any changes to hardware or software require requalification of the system.

The arrangement of software functionality to hardware resources possible with IMS means that resource redundancy can be managed in new ways (Prisaznuk, 1992). For example, secondary redundancy in resource can be incorporated at component level or the system can be reconfigured by moving required software functions to available spare processing units. This may be done statically (during system down time) (Little, 1991) or dynamically (during operation).

In addition to providing for the hardware/software interface, the three layer stack provides a method of defining how to run multiple applications, or avionics functions, on shared resources such as processing and memory. If left unchecked this would present a problem as there is potential for a number of different safety critical avionics functions, all with their own real time constraints to function on shared resources (Lee, Kim, Younis, Zhou, & McElroy, 2000). In this situation, the safety case for the aircraft would be compromised and a potentially disastrous scenario could arise should one function hog a shared resource and delay the execution of another. To avoid this, a robust partitioning mechanism including processing resource scheduling needs to be implemented that prevents applications interfering with one another.

In terms of the production of an IMS, the final system should display the following properties (Conmy, 2006):

- Technology Transparency - The underlying hardware should not have any impact on an application either during development or execution
- Scheduled Maintenance - The system should have inbuilt capability to operate in the presence of failures so that Maintenance Free Operating Periods (MFOPS) can be achieved and only scheduled maintenance need occur.
- Incremental Update - The system should be designed such that applications can be inserted/altered with minimum impact on other applications and on the supporting safety case

Another way of expressing some of the above properties is to say that an IMS should be designed as 'future proof', i.e. designed with technology transparency and with the capacity for incremental update. A future proof system can be described as (Edwards, 1997):

"A future proof system remains viable in terms of capability and affordability throughout its life cycle, despite the evolution of the technology that it embodies. Evolution of technology presents two major challenges to a system: obsolescence and capability growth."

A manner in which this can be achieved is to design a truly 'open' system architecture. An open system is one where no proprietary interface specifications exits that prevent outside agencies designing and building components capable of being integrated. It is likely that all components will have a specification following a so called F^3I guideline (Form, Fit, Functionality and Interface). This F^3I is a development of the F^3 idea introduced by ARINC for LRUs. These specifications will exist in the public domain.

In order to test the openness of a system, BAE SYSTEMS have 7 tests (Edwards, 2001):

a) Information published & publicly available – open access
b) Sufficient information provided to allow independent implementation
c) No royalties – open exploitation
d) Not dependant on proprietary components or processes
e) Standards and essential components not restricted by export controls
f) Possible to create special-to-type items which conform to the interfaces defined by the open standards and are interoperable with other items which conform to the standards (in modular systems this means that the system builder is not constrained to use only the standard modules)
g) Open to technology growth & system growth – technology transparent. In other words, open over a long period of time.

2.2.1.2. Why Use IMS?

IMS has the potential to address the needs identified for future avionics, highlighted in section 2.1. Table 2-1 below shows how the features of IMS directly address the key items highlighted. Full details on the capabilities mentioned are covered in section 2.2.2.

Requirement	Feature
Adaptive Technology to meet changing mission requirements/technology (Little, 1991)	Modular software/hardware design means it is possible to incrementally change parts of the system more easily than methods today. Certification could theoretically be achieved more quickly with modular driven design methodologies.
Affordable through the lifecycle (Little, 1991)	IMS represents lifecycle cost savings in many ways. The reduction of spares reduces logistic footprints and maintenance complexity is reduced by having similar components throughout the system. The reduction in onboard components needed (Johnson & Omiecinski, 1998) reduces weight, power and space requirements.
Exhibit improved reliability, maintainability and availability (Little, 1991)	Maintainability and availability are addressed by the benefits highlighted above. If the platform is easier to maintain and repair, the platform is more available. The commonality of modules also means that it is possible to do more maintenance at the front line, dramatically increasing availability. Reliability can be improved beyond the capabilities of conventional means by the utilisation of dynamic reconfiguration.
Modular hardware operating with processing speeds more than two orders of magnitude great than today (Sutterfield et al., 2008)	IMS applications can be implemented on naturally increasingly powerful computers. Incremental upgrade capabilities allow IMS to keep track of improving technology as it becomes available.

Table 2-1 Comparison of IMS Capabilities to Highlighted Requirements

The transfer from a federated to integrated modular solution carries with it not only benefits but risks too. Watkins (Watkins & Walter, 2007) offers an insight into these implications. Three main benefits of this paradigm shift highlighted here are:

- IMA Provides opportunity to optimise processing resources

- IMA has reduced weight and power requirements
- IMA lead to an optimised development due to the modular nature of the system

Watkins also mentions that the movement to IMA carries with it risks that should be considered. The main warning is based around encouraging companies to take a holistic approach to the problem. A Global perspective on the business model and resource management is inferred by the introduction of this technology along with requirements for maintaining links with legacy systems that may remain part of the system.

A review of the IMA system design trade-offs is presented by Grigg (Grigg, Audsley, Fletcher, & Wake, 1999) and are summarised as:

- Flexibility vs Predictability – The flexible nature of the system potentially contradicts fundamental requirements of repeatable predictable performance
- Integration vs Isoltaion – Functionally and physically coupling systems exposes risks of single failures imposing on multiple systems functions
- Run-time Efficiency vs Technology Transparency – The ability to 'plug and play' hardware modules has the potential to effect the way data is shared and handled in subtle ways, which could affect the efficiency of operation.

2.2.2. IMS Research Areas

The following sections highlight some of the major challenges with implementing IMS. They highlight the cause of the problem and any solutions or potential solutions found.

Each topic identified is explored in detail and although it is attempted to address each issue independently, there is a natural crossover of topics that occur.

2.2.2.1. Standards

A thorough investigation of IMS standards has been performed in (Stephenson, Nicholson, & McDermid, 2006). Here, three standards have been identified and their capabilities summarised, which are summarised below:

- ARINC 653 (ARINC, 2006b)
    - Defines a uniform 'Application Executive' (APEX) interface between the hardware and software layers of an avionics computer in order to provide the hardware services to the applications. This is similar to the Operating System layer as shown in the Three Layer Stack diagram of Figure 2-6. The main point of reference of this standard is the specification of time and space partitioning. This is where the applications are distributed into isolated 'partitions' upon the hardware, with independent memory allocations and scheduled processor time-slots.
- Allied Standard Avionics Architecture Council (ASAAC)
    - This standard is generated from a consortium of aerospace organisations from UK, France and Germany. It defines interface standards for software, communications, common functional modes, packaging and architecture. In addition, areas of possible obsolescence are identified for each area, such as rack size and electrical connections.
    - The ASAAC standard considers more aspects than ARINC such as file handling, threads and debugging.
- DO-297 (RTCA, 2005)
    - Provides guidance on the certification issues of IMS. The aim of this document is to highlight to designers and developers the issues of implementing an IMS on board an aircraft. Its concerned largely with the qualification of toolsets that generate configuration data and the process by which the configuration is loaded into the system.

Further to the APEX standard defined by ARINC 653, ARINC 651 (ARINC, 2006a) details potential systems architectures for implementation. The standard ARINC 651 suggests a number of hardware configurations (Johnson & Omiecinski, 1998), each with their benefits and limitations. It is highlighted that the proposed architecture shown in Figure 2-8 has the most plausible solution to the problem.

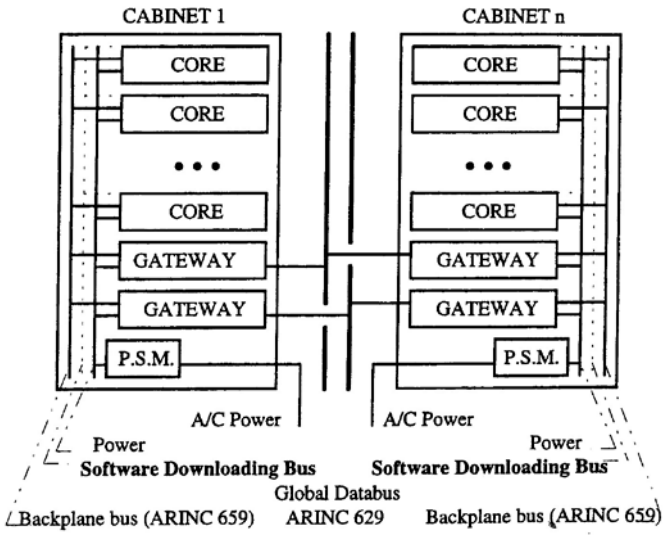


Figure 2-7 Possible IMA architecture based on ARINC 651 architecture C (Johnson, Omiecinski 1998)

Full reconfiguration across the entirety of the avionic system will be impractical as it will be too difficult to certify (Johnson & Omiecinski, 1998). However, significant benefits can be achieved by reconfiguration within a single cabinet. The theory behind this system is that each core can store a number of applications, and some decision making process decides which of these is to be executed. It lends itself well to progress to a dynamic reconfiguration scheme as no transfer of program code is required during the reconfiguration process thereby removing a potential source of a critical failure mode.

ARINC 653 is a key enabler in the development of IMA (Prisaznuk, 2008). The purpose of this standard is to define a Real Time Operating System (RTOS) and an Application/Executive Interface (APEX) to provide a general purpose interface between modular software components and the underlying RTOS, as shown in Figure 2-8. ARINC 653 does not define hardware components (Prisaznuk, 2008) but does infer requirements on their design by specifying needs for memory management and processing control such that true partitioning can be achieved by the RTOS between software applications. This standardisation of the interfaces will promote competition in development of applications as functions will be able to be generated in a modular fashion.

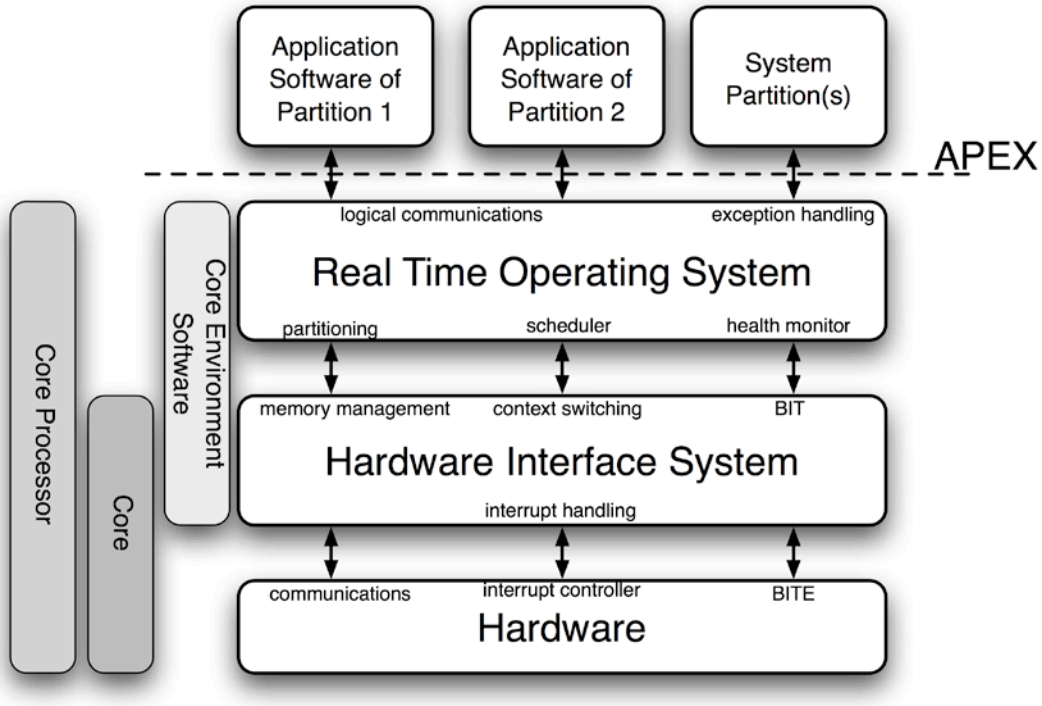


Figure 2-8 ARINC 653 System Architecture (Prisaznuk, 2008)

ARINC 653 is presented in 4 parts defining different aspects of the RTOS. The content of these sections are summarised in Table 2-2.

Part 1:	Part 2	Part 3	Part 4
"Required Services"	"Extended Services"	"Conformity Test Specification"	"Subset Services"
• Partition Management • Process Management • Time Management • Memory Management • Interpartition Communication • Intrapartition Communication • Health Monitor	• File System • Sampling Port Data Structures • Multiple Module Schedules • Logbooks • Sampling Port Extensions • Service Access Points	*standard test suite to validate parts 1&2*	*defines data exchange*

Table 2-2 ARINC 653 content summary (Prisaznuk, 2008)

2.2.2.2. System Design and Certification

Avionics systems design and integration is a complex undertaking (Schavey & Duba, 2008). Requirements not only come from the need to satisfy the desired functionality, but are required to minimise space, weight and power. IMA simplifies some of these issues by naturally reducing housing requirements (Little, 1991), but introduces different problems, such as having a large number of functions developed by independent suppliers all sharing the same physical resources. In addition, to realise the full benefit of an IMS, the system should have a flexible configuration which may be changeable during the operation of the platform. This complicates the certification issue as it may be impossible to conduct a safety case assessment for each possible configuration before operation.

A solution to software development is that of Model Driven Development (MDD) (Evans, 2003). The process discussed is a design philosophy which allows software models, with specific requirements and constraints, to be prepared and tested before any actual lines of code are produced. This kind of design is very portable and re-useable as should the target system be changed, only a reiteration of automated software generation is required to update the existing software. This design philosophy has a much greater influence on the system management throughout the entire system lifecycle.

Work towards a more complete solution to a model driven approach is presented in (Schavey & Duba, 2008). Here it is shown that the resolution of the complexity problem of integrating of the separate functionalities designed by independent institutions can only be streamlined through the use of model driven methodologies. The development of a rigorous set of modelling tools for IMS will allow the development and analysis of alternative implementation modes or incremental updates to be performed quickly. Optimising the design at ‘Aircraft level’ by considering not only the functional problems, but that of space weight and power can make drastic savings (Salzwedel, Fischer, & Baumann, 2008). The cost of architecture is quoted as a potential reduction of 72.6%, with an 80kg reduction in weight of the whole avionics system and an improvement from 0.99 to 0.999999999998 when compared to reference architecture, similar to the conclusions in (Little, 1991).

An extension to the principles used in design processes such as the Department of Defence Architecture Framework (DODAF) can be used as a solution to the design problem of IMS (Mapleston, 2006b). The idea is to deal with the system in two entities, hardware and software. One obviously has an impact on the other, i.e. there needs to be enough processing power to execute the desired functionality, but they need to be as loosely coupled as possible. Essentially, the software design is approached in terms of functionality, i.e. a description of each of the avionics functions is realized and defined, much in the same way as MDD. The interactions between each of these avionics functions can also be defined. The hardware needs to be designed in terms of processing power required, memory requirements and available network bandwidth, along with a suitable network architecture put in place. This then allows a logical mapping of software to hardware architectures to take place. As part of this mapping a number of rules are required to be implemented e.g. redundant functions cannot be mapped onto a generic processing module that contains the function it is duplicating. This process allows systems configurations to be developed in the design phase of the system. In a similar way, this could be performed on-line to help choose an alternate configuration should it be necessary, or performed off-line to produce a number of acceptable reconfigurations available (Porcarelli, Castaldi, Di Giandomenico, Inverardi, & Bondavalli, 2003).

There are two main issues with the certification of IMS (Hollow, McDermid, & Nicholson, 2000):

1. The certification of the re-use of components either from one IMS project to another, or as part of a change in configuration or an upgrade in component
2. The certification of a dynamic, reconfigurable system. As it is possible for a system, given a set of conditions, to generate a new configuration that has not been tested, it is impossible for this configuration to have been certified.

DO-297 (RTCA, 2005) is working toward having a process by which certification can be achieved for new applications and/or modules in an IMS, without the need for re-

acceptance of the whole system (Elmqvist, Nadjm-Tehrani, Forsberg, & Nordenbro, 2008). There are 6 tasks within DO-297 that are followed to achieve this incremental acceptance:

1. Module Acceptance
2. Application software or hardware acceptance
3. IMS system acceptance
4. Aircraft integration of IMS system – including Validation and Verification (V&V)
5. Change of modules or applications
6. Reuse of modules or applications.

However, the problem still exists on how to change application or configuration without re-certifying the entire system.

Not only are there design requirements for hardware and software development, there are organisational arrangements that have to be in place to design and commission an IMS (Elmqvist et al., 2008; Mazuk, 2008). This is in-line with the standard DO-297 which requires certain design development roles to be owned in order for certification to be achieved. All references identified suggest that there is a need for good systems engineering in order to successfully implement an IMS to its full beneficial capability.

A further three points to consider in the system design problem are highlighted by Watkins are are as follows (Watkins & Walter, 2007):

- The Optimisation of Systems Resources – it is important systems integrators and developers have a common goal on how spare resources are utilised.
- Change containment when hosted systems change – how is the impact of changing processing modules minimised when they could alter performance characteristics of software applications
- Change containment when the IMA platform changes – If the platform is modified with new capabilities, how can these additions be accounted for without having to reconsider the full system.

2.2.2.3. Hardware

The modular design of all parts of an IMS is the key to its success in displaying the three attributes outlined in (Conmy, 2006). Already the cost-benefit of using LRMs over LRUs have been highlighted in section 2.1, with the case study performed in (Little, 1991) and (Salzwedel et al., 2008). It is clear from this how designing modules built to a F^3I specification can provide long term benefits to the end user.

It is widely proposed that to mount the hardware into the platform, a small number of 'racks' will be used to house a number of LRMs. This concept is demonstrated in Figure 2-9.

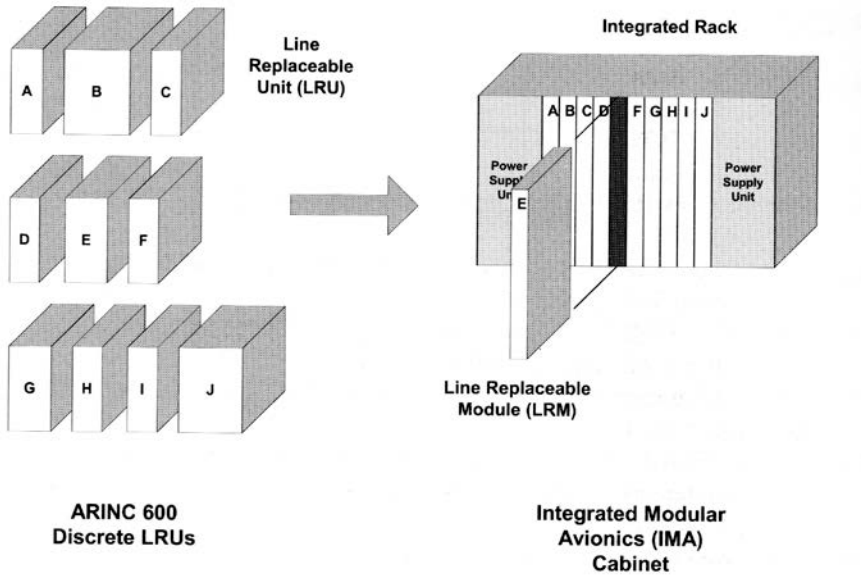


Figure 2-9 Integrated Modular Cabinet Comparison (Moir et al., 2006)

The modules within the rack are connected by a databus known as the 'backplane bus'. This can be connected to the main system bus by the inclusion of a gateway device.

This packaging method is already in use in some aircraft today. One of the main benefits found is the potential weight saving of this design. It can be shown (Moir et al., 2006) that replacing all the individual power supply units within each discrete LRU by 2 power supply modules responsible for a whole rack can have significant weight saving benefits. This also increases the redundancy of the power supply and reduces power dissipation.

A further description of IMS hardware is shown with an example system implemented in the Advanced F-35 Multi-Mission Jet Fighter (Sutterfield et al., 2008). Figure 2-10 shows an Avionics rack designed to take a number of standardised modules. Communication takes place between inserted modules via a standardised back-plane bus. In addition, the rack uses a liquid cooled heat sink method to keep the temperature of the modules within a safe operating range.

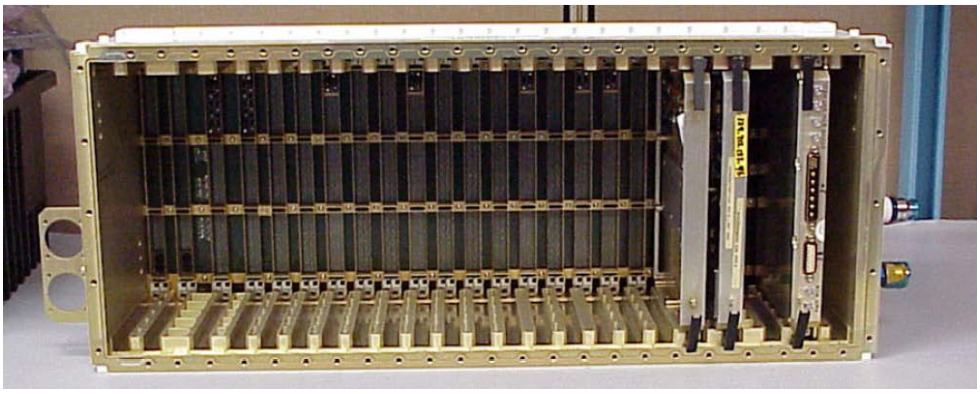


Figure 2-10 High Density Packaging/Avionics Rack Example (Sutterfield et al., 2008)

A concept for a standard processing module is shown in Figure 2-11 (Sutterfield et al., 2008). The module suggested contains a number of standard FPGA (Field Programmable Gate Arrays) that can be loaded with the latest cores when upgrades become available. This increases the operating life of the processor units because the hardware need not be replaced every time there is a change to the processor, resulting in lifecycle cost savings.

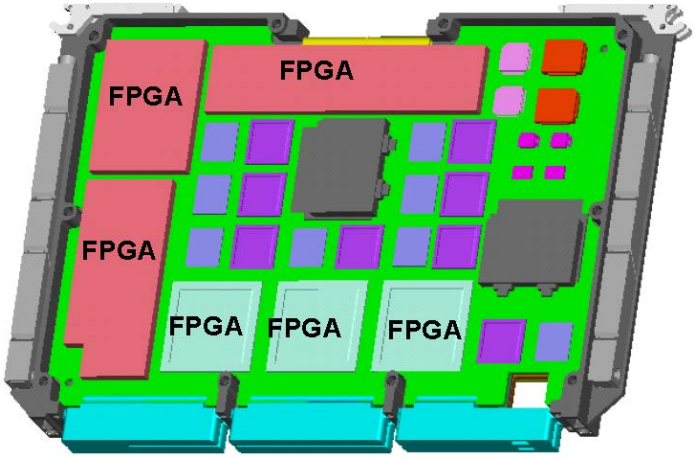


Figure 2-11 FPGA Standard Module for Processing (Sutterfield et al., 2008)

Recent complications in partitioning systems have arisen from the development of multi-core processors. (Kim, Yoon, Bradford, & Sha, 2014) suggests a process for managing this complication to ensure that the advantages obtained from multicore processors can be absorbed with minimal impact on previously designed and validated software functions.

2.2.2.4. Partitioning

An architectural design of generic core module produced by Wind River, based on requirements from ARINC 653, is outlined in (Parkinson & Kinnan, 2003). The focus here is on a design that allows multiple applications to run on a single module, without each application unexpectedly affecting one another.

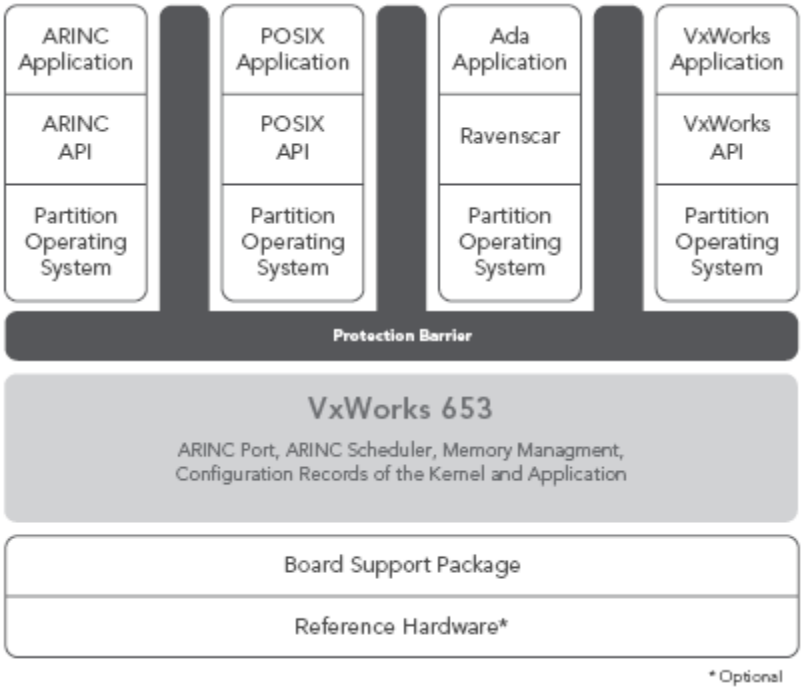


Figure 2-12 Partitioning Scheme for a Core module (Parkinson 2006)

Figure 2-12 demonstrates a partitioning scheme, originally drafted in (Prisaznuk, 1992), that not only protects applications from faults in their neighbours, but allows legacy software applications to be executed by the system. The operating system used to enforce the partitioning, and ensure resource scheduling to the applications is VxWorks 653. Note the similarity of this structure to the three layer stack in Figure 2-6.

Moir (Moir & Seabridge, 2008) identified four in-service systems that demonstrate a modular partitioning system as described above. These are:

- Green Hills software with level A, INTEGRITY-178B RTOS on the Sikorsky S-92 helicopter
- Lynux Works Lynx-OS level operating system in association with Rockwell Collinson the adaptive flight display system on the Bombardier Challenger 300 business jet
- Wind River Systems with AE653 on the Boeing 767 tanker transport and C-130 avionics modification program (AMP);
- CsLEOS RTOS developed by BAE SYSTEMS and certified to DO-178B level A for a-by-wire flight control system upgrade to the Sikorsky S-92 helicopter

The need for not only spatial but temporal partitioning as a fundamental need of a distributed real-time system (Yann-Hang, Daeyoung, Younis, Zhou, & McElroy, 2000). Spatial partitioning is performed in hardware where appropriate boundaries prevent cross interference between applications or partitions. Where resources are shared, temporal partitions are used to prevent interference and ensure each function meets their timing constraints. A model presented for Strongly Partitioned Real-Time Systems (SP-RTS) demonstrates a solution for temporal and spatial partitioning. The stronger the partitions are and the more isolated the applications are and the easier it is for modular certification to be applied as a change in configuration has less collateral impact on neighbouring functions. Furthermore, requirements for temporal and spatial partitioning drive the possible configurations available for a system (Yann-Hang et al., 2000). Configuration possibilities can be restricted as each task and communication require timely access to the shared communication and processing resources in ways that do not interfere with the operational of neighbouring functions.

The temporal problem of the real-time system is one that is holistic, in that it can only be assessed by a complete system analysis as opposed to a study of a small part of the system (Grigg, 2002). Changes in design to part of the system therefore affect the system as a whole. Grigg suggests a ‘reservation-based timing analysis’ that provides the ability to

model the timing behaviour of parts of a distributed system, and limits the scope of re-analysis required for localised changes.

2.2.2.5. Configuration

There has been a wide amount of research investigating the methods for allocating real-time functions to a flexible set of resources whilst maintaining the integrity of the function. A technique developed in (Yann-Hang et al., 2000) presents a requirements set in order to configure real-time functions upon a single board in terms of managing the processor and network timings. This technique uses the deadline requirements of tasks and messages of individual functions in order to find a configuration that satisfies them all.

An investigation into further allocation techniques was performed in (Hladik, Cambazard, Daplanche, & Jussien, 2008). In similar fashion to (Yann-Hang et al., 2000) a method using constraint programming is implemented to solve the problem. The techniques studied included graph theory, branch and bound, genetic algorithms, clustering, steepest descent, tabu search, simulated annealing, neural networks and dedicated heuristics. It was found that for specific problems no technique was more appropriate than the other. It was also found that most of these techniques were too dependent on their initial constraint and made any changes to the modelled system difficult to implement.

The problem was solved in this instance by splitting the allocation problem into two phases. Firstly the applications are allocated based on resource constraints, and then the timing for each processing stream is corrected in order to satisfy the real time functions based on the overall global timing constraints. This method was found to be a robust method of satisfying a large amount of functional sets and achieving good utilisation rates of resources and good use of the bandwidth of the communication bus.

Current maturing methods of deriving a configuration are based on a MDD methods (Evans, 2003; Schavey & Duba, 2008) and are concerned with producing a series of system arrangements known as Blueprints (Grigg et al., 1999; Jolliffe, 2005). By considering the hardware, software and configuration requirements, model-based, offline mapping tools

can be used to optimise and certify blueprints and store them as a lookup table for later reference.

All of these investigations have been concerned with a static reconfiguration scenario, i.e. when the system in question is off-line and the allocation of functions are designed and assessed before run time.

2.2.2.6. Reconfiguration

Reconfiguration is a complex problem, particularly when associated with the safety requirements of modern day aircraft. Historically, there is an overriding requirement for the demonstrable predictability of the real-time system by off-line analysis (Grigg et al., 1999). The system designed needs to be certain that the process of reconfiguration will not induce a failure, nor implement a non-functional or inadequate configuration. There are two fundamental methods that can be utilised to implement reconfiguration; multi-static configuration and dynamic reconfiguration. Multi-static and dynamic reconfiguration in the following way (Field et al., 1997):

Multi-static reconfiguration:

- Completely established at design time
- The system stores a set of pre-defined configurations. Each configuration explicitly identifies the allocation of software (functionality) to hardware (resources) and has been checked for predictability, meeting defined time constraints, latencies etc.
- Contains a set of pre-defined rules or triggers that initiate changes to the system from one configuration to another

Dynamic reconfiguration:

- Is achieved by using algorithms that are continually running while the system is live to determine the next best allocation of software (functionality) to hardware (resources).
- This algorithm considers the requirements of what software functionality is required to be executed and the condition of the available resources

The ability for the system to change its configuration is one of the primary benefits of IMA (Field et al., 1997). This ability supports the maintenance and continued operation of the system in the potential presence of faults, all of cost benefit to the user. However Johnson (Johnson & Omiecinski, 1998), highlights that there are stringent requirements to the reconfiguration process, particularly online reconfiguration, in order for it to be certifiable. These requirements are summarized as:

- Reconfiguration shall reduce the overall redundancy requirement
- Reconfiguration shall allow maintenance to be deferred
- Existing System safety requirements shall be met or exceeded
- The reconfiguration scheme shall not be susceptible to random hardware failure
- Reconfiguration delays must be short
- Transient behaviour during reconfiguration must be safely bounded
- System state must be preserved during reconfiguration.

Because of these stringent requirements, it is likely that for a first step, IMS will be implemented without the ability to dynamically reconfigure (Johnson & Omiecinski, 1998). This form will utilize the benefits already highlighted of modularity, but will not yet exploit the full potential of IMS. This statement is now supported by the various systems already highlighted in this document in Section 2.2.4 that have been implemented on aircraft.

Coutinho (Coutinho, 2008), suggests the use of a multi-static approach using an AIDA (Analogue Integrated Design Automation) architecture. The system developed is structured in a hierarchy with a 'Systems Manager' making overriding decisions supported by a lower level 'Module Manager' which is responsible for managing the service provision to the applications. A similar approach is investigated by Porcarelli (Porcarelli et al., 2003) who has performed some research into managing reconfiguration. The concept suggested is that a number of strategies or functional arrangements are defined off-line to provide configurations during run-time. A decision making agent continually assesses which strategy is the most appropriate and configures the system accordingly. This study focuses on telecommunication networks, rather than the stringent requirements of a hard real-time application. The most extensive method suggested is by Hollow (Hollow et al., 2000)

who proposes that vast numbers of safe reversion configurations can be established offline and referred back to as a look-up table when required.

Hollow (Hollow et al., 2000) tells us that current certification practises require an assessment of each complete configuration. Therefore, as long as the number of configurations generated is low, both the above approaches have good chances of meeting certification standards. Utilising multi-static configurations allows the operator to take advantage of the modular nature of IMS, and have some capacity to restore system functionality in the presence of faults. Salomon (Salomon & Reichel, 2011) is performing research into a toolset that will perform the appropriate safety case evaluations for an IMA in an attempt to reduce the design time and certification of various configurations. Although this process could be seen as quick in terms of the design of a static configuration, it does not yet (if it ever will) seem appropriate for the use in automatically accrediting configurations at run time during a dynamic reconfiguration problem.

Strunk et al., (Strunk, Knight, & Aiello, 2004) is working towards a reconfiguration process that may one day be certifiable to stringent standards. The reconfiguration process in this work is performed by a single module, which is responsible for the configuration at any one time. This paper addresses many of the problems of dynamic reconfiguration, such as the timing of the process. The plan pivots around a utilizing a central controller which provides an opportunity for a single point failure to cripple large parts of the system. However, having a single decision maker will ease the certification process.

An interesting concept raised by Lopez (Lopez, Royo, Barrado, & Pastor, 2008), is taking a wider viewpoint to the reconfiguration problem and investigates not only the configuration problem, but also how to use mission requirements and available resources to generate the required functional assignment at run time. Lopez provides a good framework and process to the dynamic configuration of an aircraft.

Further to the totally autonomous aspect of reconfiguration, research is being performed into the inclusion of a Human operator in the reconfiguration decision loop (Dajiang, Jinxia,

& Jihong, 2011; Montano & McDermid, 2008). Dajiang et al. (Dajiang et al., 2011), suggests a Systems-Theoretic Process Analysis method to involve a pilot into the decision making aspects of choosing a new configuration. Involving a human in the process may solve some problems with decision making authority and potentially reducing the risk of reconfiguring to an inappropriate configuration, but introduce new problems such as the display and communication of relevant information to the decision maker who already has a high workload demand upon them.

A thorough investigation into a dynamically reconfigurable system was conducted by (Ellis, 1997) where the primary method of configuration was to upload new configurations over the network during the boot process, then use redundant processes stored in RAM as a fast access options in the event of errors. It was also highlighted that in the instance of failure, redundancy is required by multiplexed systems as well as spare processing capacity for re-allocation of tasks.

2.2.2.7. Network Requirements

The IMS concept is reliant on function integration of tasks but physical separation across a platform. In such hardware implementations, software applications have to exchange information via appropriate communication media. There is a pressing need to develop higher speed data transfer buses (Zhang et al., 2003), particularly for all the hard, real-time tasks that need to be executed in parallel, system wide. High speed data buses do exist in the commercial world in the form of high speed Ethernet, but there are stringent requirements from avionics specifications that mean Ethernet in its current form is not suitable for use on an aircraft. A potential solution to this problem would be to use a fibre channel communication that would offer data rates greater than 1Gbit/s. A commercial example of this technology already widely used is ‘Firewire’, a communication protocol used with standard personal computers (Collinson, 2011).

The reason Ethernet is unsuitable in its current form is because the communication is non-deterministic. Ethernet uses a collision detection system that detects when two nodes are attempting to use the common bus, and delays the sending of the message by a pre-

determined amount of time until the bus is clear. The result is that the package of information cannot be guaranteed to be delivered at a set time under all conditions. In simple terms, it is possible to use Ethernet in a deterministic way, by using a collision avoidance technique. Each required communication during the process is allocated an amount of time to use the bus to send information. By doing this, Ethernet becomes a realistic option for real time distributed networks.

A method named Time Triggered Ethernet (TTE) (Kopetz, Ademaj, Grillinger, & Steinhammer, 2005) seeks to formalise the determinism of Ethernet communications for purposes of real time communications. In this method, the processor clocks are routinely synchronised and messages are assigned a time period in which to be transmitted.

A deterministic Ethernet method that has matured sufficiently for widespread use onboard aircraft is AFDX (Avionics Full-Duplex Switched Ethernet) (Bisson & Troshynski, 2003). Here a 'scheduler', a management method akin to a bus controller, manages the timing of communication via defined 'virtual links', which are software defined connections between data source and destination.

2.2.2.8. Distributed Control

There are limitations of traditional real-time system design/analysis when applied to IMA (Grigg et al., 1999). Design is conducted by assessing the Worse Case Execution Time (WCET) followed by a system wide 'schedulability' analysis. This analysis must be repeated for the system component under design if any of the following change:

- Its own timing requirements, WCET or communications requirements
- The requirements (as above) of any other software components allocated to the same shared resource
- The hardware implementation of any required processing or communication resource
- The overall allocation (mapping) of software components to hardware resources

This paper goes on to present a useful approach to the temporal aspects of the scheduling problem using a resource reservation approach. This is a top-down approach where system

timing requirements are considered with a logical architecture definition of systems functions and a specified amount of resource that is required to be reserved for its execution. This can then be analysed offline with different conceptual hardware arrangements in mind. It is evident that this process is not appropriate for a system designed with flexibility in mind. In fact, many dynamic system approaches provide little or no features to facilitate highly dependable, real-time performance required by critical systems (Ford, Bull, Grigg, Guan, & Phillips, 2009). Some of the above issues highlighted have been addressed by (Grigg, 2002) where a method is suggested involving *a priori* specification of its timing requirements and testing to show that these have been met using timing analysis methods. Using a 'reservation-based timing analysis' the approach provides an ability to model timing behaviour of parts of the distributed system in isolation from the system whole, enabling localised changes to occur without total revalidation.

The problem of creating a hard, real-time distributed system is not a new one and techniques exist to ensure concurrent, repeatable systems are developed (Kopetz, 2011). However, this theory is based on the concept that a system is designed in a fixed configuration then remains unchanged during operation. Clearly these methods require adjustment and modification for a reconfigurable application and are more attuned to a multi-static approach.

A method suggested as a solution is a Physical Asynchronous/Logically Synchronous (PALS) design pattern (Miller, Cofer, Lui, Meseguer, & Al-Nayeem, 2009). This is a method that acknowledges the fact that each node within a network will maintain its own time but ensures that the message transmissions are logically arranged such that events remain concurrent and deadlock or race conditions are avoided. Essentially, a global PALS time period is defined and within this period time is allowed for computation and message transmissions to occur. Each of these events are allowed time where local clock variations will occur relative to the overall global clock. Miller goes on to discuss that this method can be applied to IMA with careful consideration between the PALS time definitions and the intrinsic IMA timings. This can save complex clock synchronisation techniques between

processing units (Grigg et al., 1999) as each avionics unit will still be subject to their own offset, drift and jitter from the true global time (Miller et al., 2009).

A promising solution offered to provide a real-time solution when generating a configuration to a given IMA installation is the adoption of dependable, real-time service oriented architectures (Tsai, Lee, Cao, Chen, & Xiao, 2006). This solution uses a service-oriented approach where 'services' and 'consumers' announce their provisions or needs respectively to a service broker (or systems manager)., in a process called 'service discover'. Using the timing and communication needs identified, a recursive search pattern can be implemented to identify the 'optimal' solution to the configuration. This method seems to have been tested in simulation for a variety of service needs but does not offer specific examples of configuration solutions. There is some scepticism of this method of how a recursive solution can be bounded to a real-time solution.

2.2.3. Fault Management in IMS

A key area of interest as defined by the scope of this work is that of fault management within IMS. A thorough investigation into these concepts was performed by Mapleston (Mapleston, 2006b). The following is a summary of the work performed in order to highlight the key areas.

By comparison to existing fault management techniques, there are four key areas required for performing fault management in IMS, and to eliminate failure entirely from the system as shown in Figure 2-14.

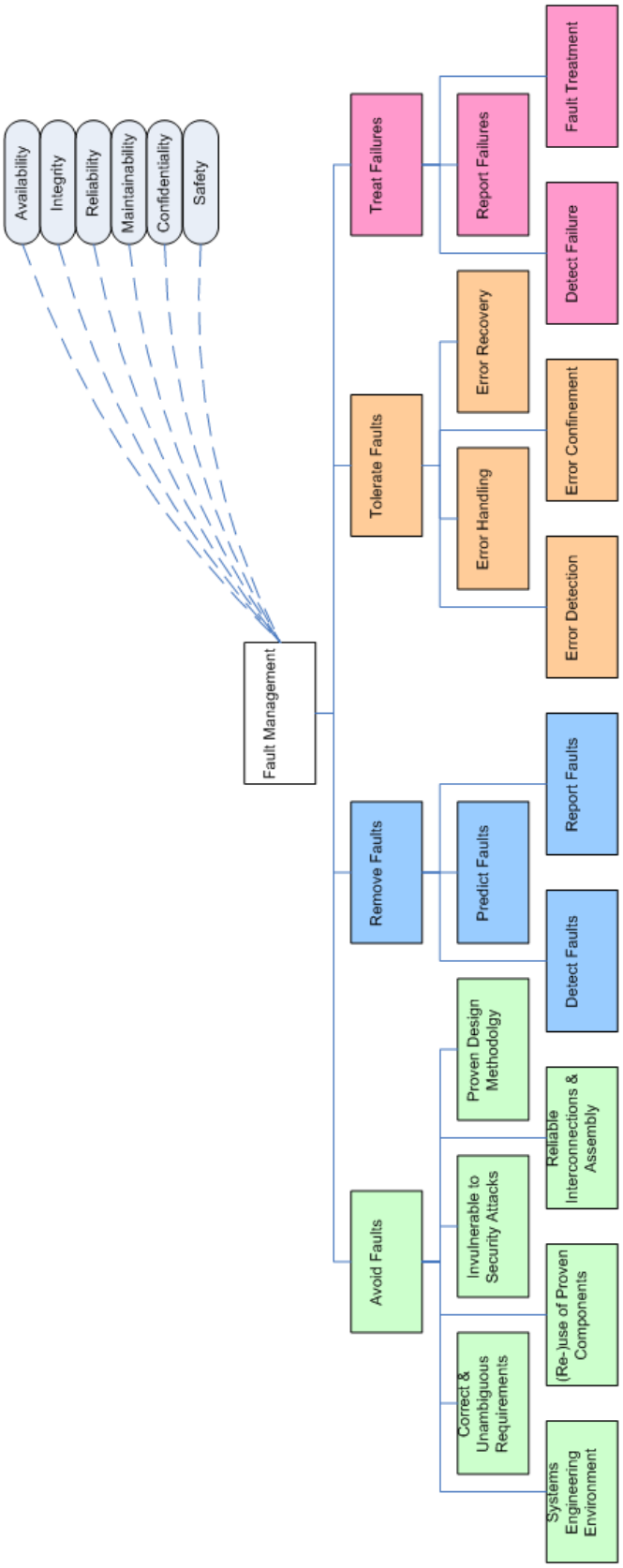


Figure 2-13 Structured Requirements for Fault Management (Mapleston, 2006a)

In summary, the four key areas identified are:

- Avoid Faults: Requiring a Systems Engineering approach to improve capture, maintenance and presentation of complex information during all stages of the lifecycle.
- Remove Faults: Requires a capability to predict the risk and identification of faults in the aircraft system so they may be removed.
- Tolerate Faults: Requires the handling of an error via detection, handling, confinement and recovery methods in order to maintain the service provided by the system.
- Fault Treatment: Requires the capacity to completely remove the identified fault from the system via repair or replacement methods.

The following sections investigate these areas in detail and suggest methods for addressing them.

2.2.3.1. Fault Avoidance and Removal

In order to avoid the introduction of faults into a system involves the following components:

- Use of reliable components
- Reliable techniques for interconnection and assembly of components
- Screening out of interference
- Correct and unambiguous requirements specification
- Proven design methodology
- System engineering environment to manage complexity

A solution to the above requires a good structure of procedures of system design. Fundamentally, this needs to involve a modular design philosophy (Elmqvist et al., 2008; Evans, 2003; Schavey & Duba, 2008). Mapleston (Mapleston, 2006b) suggests an Architectural Design Approach (ADA) to address these issues, as shown in Figure 2-14.

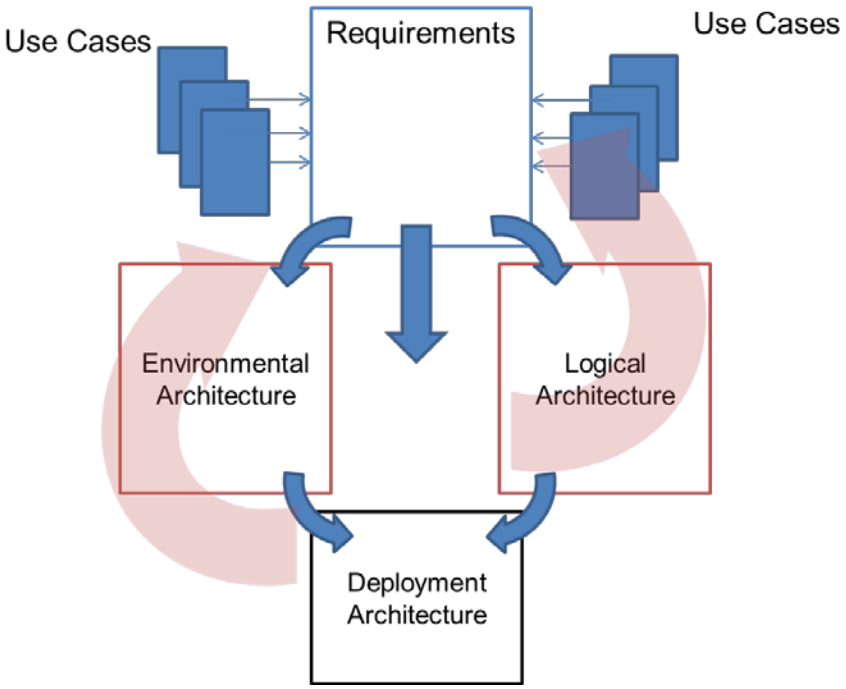


Figure 2-14 Architectural Design Approach (Mapleston, 2006a)

This proposed solution is derived from a process developed to assist the management of battlespaces. The commonality of the problem between the management of a flexible scope of a battlespace to an IMS is evident in that each has a definable, yet flexible, number of differing components that have the capability of performing different tasks depending on current assignment.

Furthermore, it is suggested that the system could be design using an Architectural Analysis and Design Language (AADL), a shown diagrammatically in Figure 2-15. The benefits of performing this process are that AADL has the ability to:

- Specify software and hardware systems architectures
- Specify component interfaces and implementation properties
- Analyse systems timing, reliability, partition isolation, etc.
- Enable system integration with tool support
- Verify source code compliance and middleware behaviour.

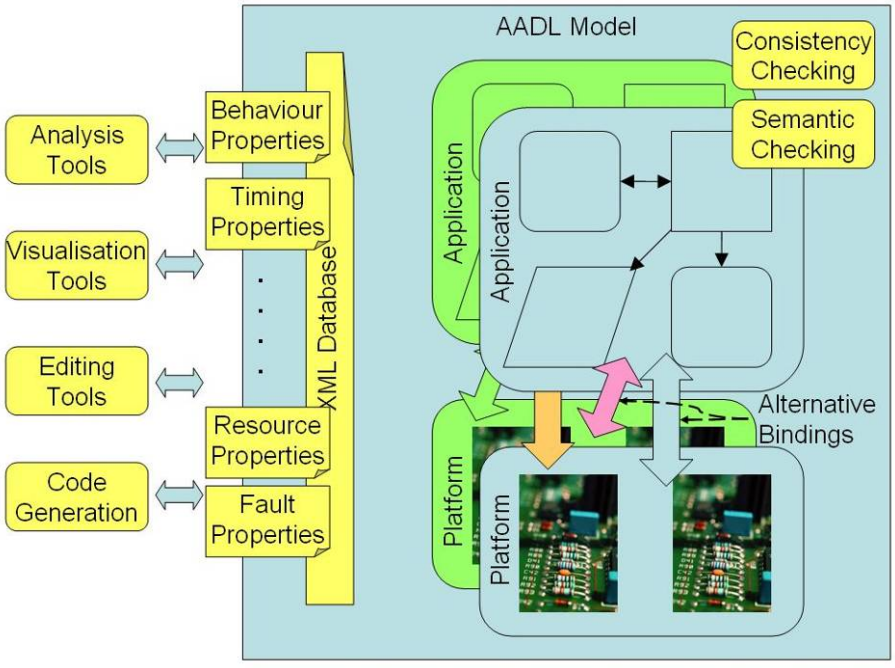


Figure 2-15 AADL Overview (Mapleston, 2006b)

Recent extensions on this work include generic model based approaches (Morel, 2014) that look to use the integration of physical and functional views to perform safety and

health assessments. An additional recent study (Si, Wang, & Liu, 2015) use AADL to model the partitioning involved in an IMA to correctly analyse the safety of the system (by focussing on the temporal aspect of partition) in the design and development phase.

2.2.3.2. Fault Tolerance

Fault tolerance is defined as: "...the capacity of a system to continue to provide a service in the presence of faults" (Avizienis, Laprie, & Randell, 2000). Fault tolerance is achieved by implementing the following functions within a system:

- Error detection
- Error handling
- Error confinement
- Error recovery

The general theme of each of these methods is the concern that each fault arising requires the prevention of the fault propagating through the system. The following suggest how each of these methods could be implemented in an IMS.

Error Detection

There are currently a number of methods for identifying errors in a system. The likely faults to occur within an avionics system are identified as (Mapleston, 2006a):

Error Detection Mode	Description
Replication	Parallel channels not providing identical results
Timing	Worst case execution time exceeded for software component, events occurring more often than expected and data timeout via communication
Reversal	Inversion of process using output does not replicate original input.
Coding	Errors within data
Reasonableness	Check that value falls within a range of acceptability
Structural	Check that all structures defined are filled with correct amount of data
Hardware Built-in Test	A series of checks that are designed and built into hardware

	components in order to be able to detect problems with the hardware.

All these methods are achievable to some extent within an IMS with existing, understood techniques.

Error Handling

An ideal fault tolerant component capable of error handling is shown in Figure 2-16 (Burns & Wellings, 2001).

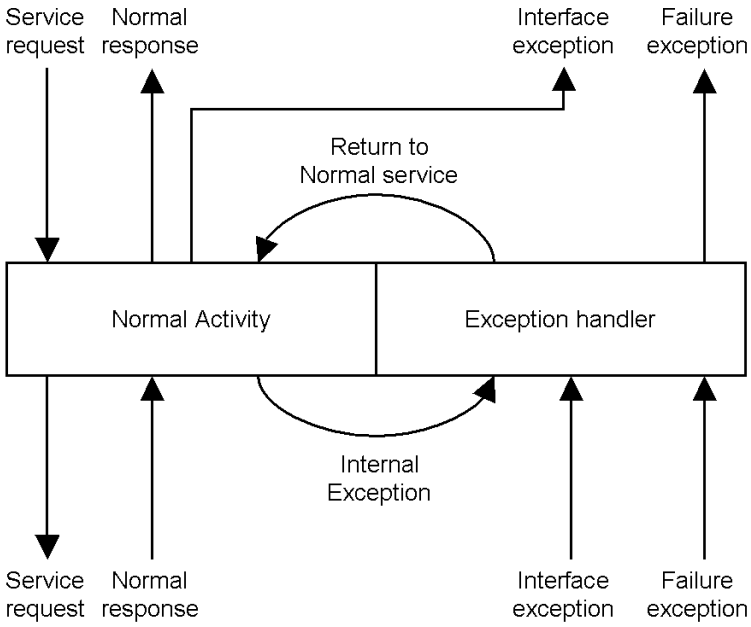


Figure 2-16 Ideal Fault Tolerant Component (Burns & Wellings, 2001)

This model suggests that in the event on a detectable error (or exception) the normal processing stream is interrupted by; and internal exception, an interface exception, or a failure message indicating exception. The component itself should then be able to handle the exception and then return as before to the normal processing activity.

Within an IMS, two error handling techniques are identified that should be implemented in conjunction (Mapleston, 2006b). These are designed to respond to errors within the application and errors within the environment.

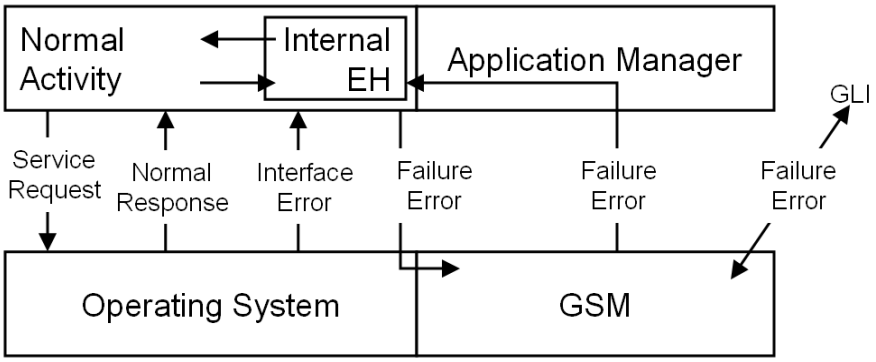


Figure 2-17 IMS Environment Error Handling (Mapleston, 2006a)

Where:

GSM = Global Systems Manager

EH = Error Handler

MSL = Module Support Layer

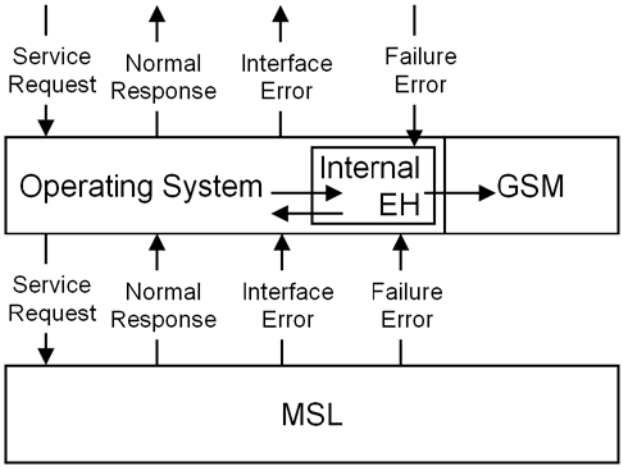


Figure 2-18 IMS Internal Error Handling (Mapleston, 2006a)

Where an error that affects normal operation is raised by another process or by the environment, this arising is always handled by the GSM. The GSM has an oversight to the complete system and can therefore advise applications via the application manager about the specific course of action to take, such as ignoring erroneous inputs. Handling errors may then be thought of as a series of atomic actions across individual systems, with a hierarchy of error handling to prevent error propagation.

Error Confinement

Within an IMS, there are many separate elements all working together to perform a task. If one of these units were to fail, there is a chance that this error will quickly pollute the processing stream and cause widespread failures as a result of a single fault. It is an essential part of fault tolerance to confine the damage caused by errors before any error recovery can be attempted.

A potential solution to the problem of error confinement as the implementation of atomic actions (Burns & Wellings, 2001). Atomic actions are defined as:

"An action is atomic if the process performing it are not aware of the existence of any other active process, and no other active process is aware of the activity of the process during the time the processes are performing the actions" (Burns & Wellings, 2001)

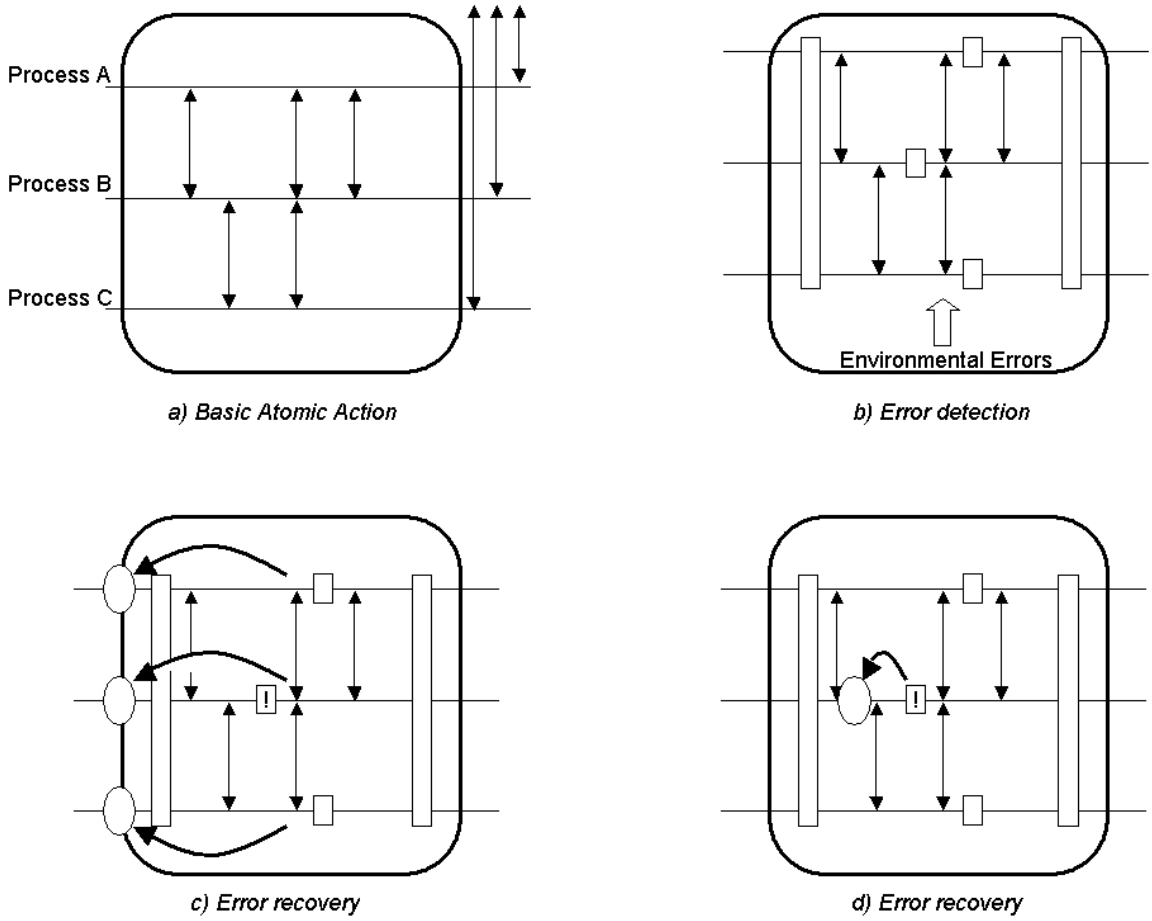


Figure 2-19 Atomic Actions Examples (Mapleston, 2006b)

Figure 2-19 shows an example of atomic actions implemented within a process. Processing streams A,B and C are working together to achieve some common goal. The atomic, defined by the box in bold, has a well-defined start and end point and clear sideways boundaries where no other streams are involved. Figure 2-19 a) shows an atomic action where no errors have occurred, and A,B and C have shared information correctly to arrive at their conclusions. Figure 2-19 b) shows where error detection mechanisms have been included into the process. These are implemented as it is highly unlikely that a processing stream can be truly isolated, and will be subject to environmental disturbances. By including this error checking it is possible to implement error recovery within the atomic action itself. Figure 2-19 c) shows how, at the identification of the error, all processing streams are returned to the start point where it is assumed that all inputs were free of error. By repeating the process from the beginning, the atomic action can be sure that there has been no propagation of the fault. Figure 2-19 d) shows how just one lane can roll back in the attempt to recover from the error, without affecting the other streams. Roll-back, or backwards error recovery, is subject to time constraint issues where the process may be required to be deterministic. An alternative method would be to use a forward error recovery method where by upon the identification of an error, the process carries on straight away using a known correct state. The next section will look at error recovery methods in more detail.

Error Recovery

One of the most tried and tested methods of error recovery is the reversion to a redundant process channel in the event of the detection of a fault with the primary channel. Other methods include parallel processing and with voting stages to remove erroneous results (Moir & Seabridge, 2008).

Within an IMS there are both opportunities and challenges to implementing redundancy. The first challenge is that IMS concepts do not highlight how multiple lanes are implemented (Mapleston, 2006a). Redundancy could be achieved by including the following fundamental requirements:

- Applications should not be aware of replication (i.e. voters and adjudicators should sit outside)
- Messages should be sent from 1 to n in that outputs are replicated seamlessly to all channels

The diagrams in Figure 2-20 shows various ways in which data provided from a sensor can be replicated to multiple channels in different ways.

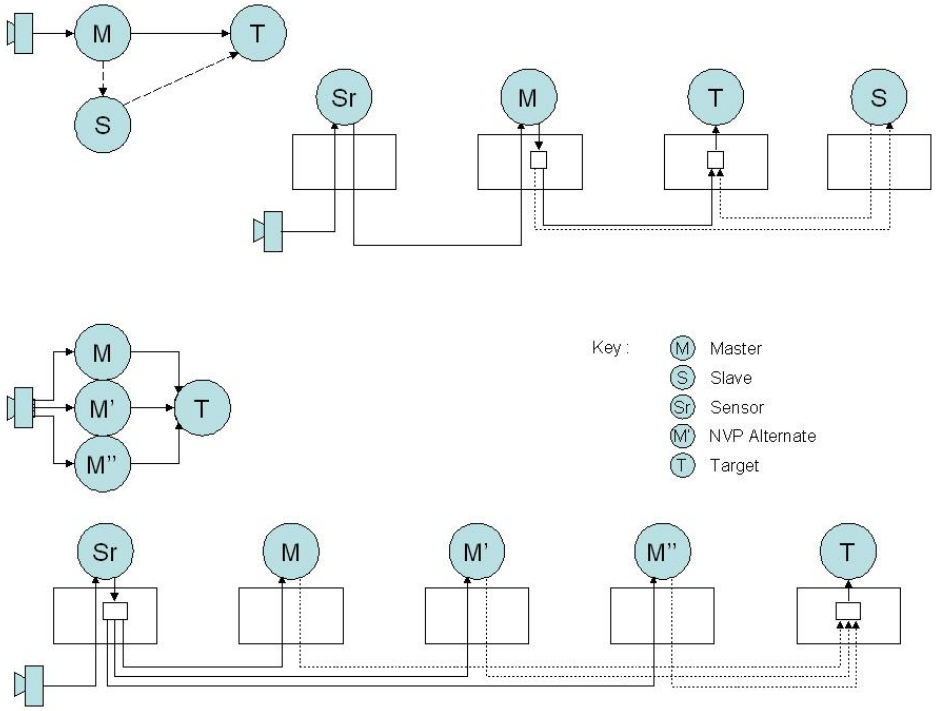


Figure 2-20 Implementing Replication in IMS (Mapleston, 2006a)

In addition to this, there is the opportunity to take advantage of the inherent flexibility of an IMS. Figure 2-20 shows a system in a set configuration. If redundant processing units are available the system could reconfigure in the event of a failure to restore a lost processing element to a new physical location. Although this would not recover from the error occurrence, it would mitigate the effect of the error by restoring all redundant channels.

2.2.3.3. Fault Treatment

Fault treatment is defined as the repair or replacement of faulty system components (Mapleston, 2006a). The hardware identified as potential solutions to IMS requirements will have a very limited scope on how they can be repaired during flight and the likely solution will be tolerate the fault until the component can be replaced at the next maintenance.

An outstanding issue highlighted (Little, 1991) is the requirement of future systems to have guaranteed maintenance free operating periods. This cost saving measure will allow aircraft to fly repeated missions with no maintenance required in between, despite the occurance of 'arisings' or fault occurances. The side effect of this issue is the improvement required in aircraft systems to guarantee a level of performance by tolerating occurring faults. In addition to this, aircraft will start to have improved diagnostics systems that can identify faults and perform basic preparations for maintenance, reducing the down time of the aircraft.

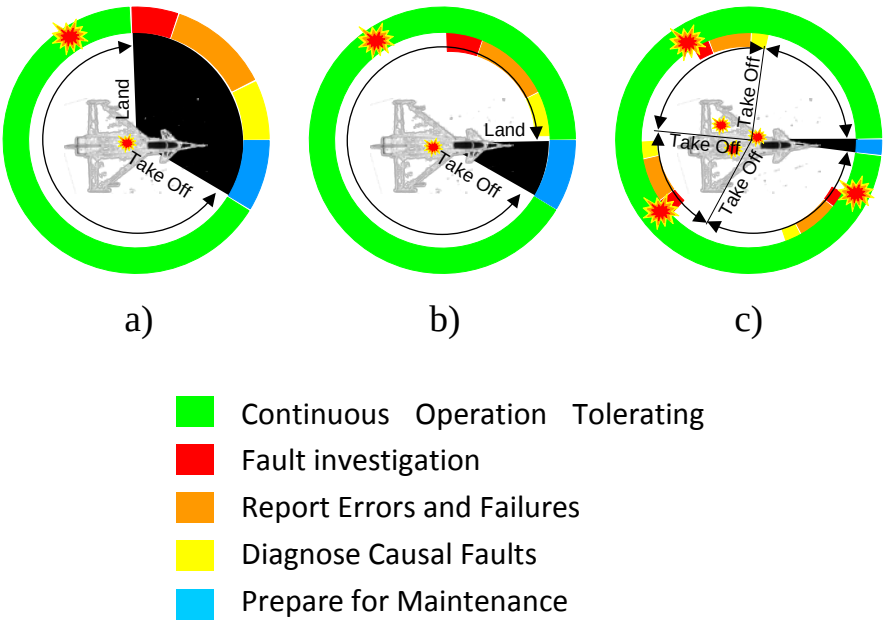


Figure 2-21 Future Fault Treatment Model (Mapleston, 2006a)

Figure 2-21 shows the model for future fault treatment. Figure 2-21 a) shows a traditional maintenance approach. During operation, a fault occurs that is critical enough to require

the pilot to immediately return to base. The aircraft is then subject to a period of fault investigation that generates reports on the error signals and failures. This is then used to diagnose the fault (if at all possible) then corrected through maintenance activity. The vision is that during a phased in approach as shown in Figure 2-21 b), an advanced on-board diagnosis system would identify the cause of the occurrence, and along with appropriate redundancy and safety measures, the aircraft is able to complete the intended mission whilst tolerating this fault. During this time, fault investigation, reporting and diagnosis could occur remotely, readying the aircraft for maintenance upon arrival. This reduces the amount of time the aircraft is out of service. A final implementation is shown in Figure 2-21 c). Here, the aircraft is subject to a number of fault occurrences during operation, but onboard diagnosis and system reconfiguration enables full safety margins to be maintained. This extends the operational capability of the aircraft as maintenance activities can be delayed until a convenient period.

2.3. Examples of IMS implementation

Often, the implementation of IMS is propriety information and details are not released. However, there are some documented cases on the use of IMA in current air vehicles.

2.3.1. Genesis IMA

Smiths Aerospace have developed an IMA referred to as ‘Genesis’ (Generic Networked Elements for the Synthesis of Integrated Systems) and has been implemented on products such as the Boeing 777, the F-22 and the Boeing 787 (Watkins, 2006). The system implemented has an open architecture such that third party suppliers can produce products to integrate with the architecture.

The way the Genesis system configuration differs from a traditional federated system is shown in the differences between Figure 2-22 and Figure 2-23. The IMA makes use of ‘virtual systems’ whereby each processing stream, or task, functions as if it were housed on a single processing unit when they actually are physically separated.

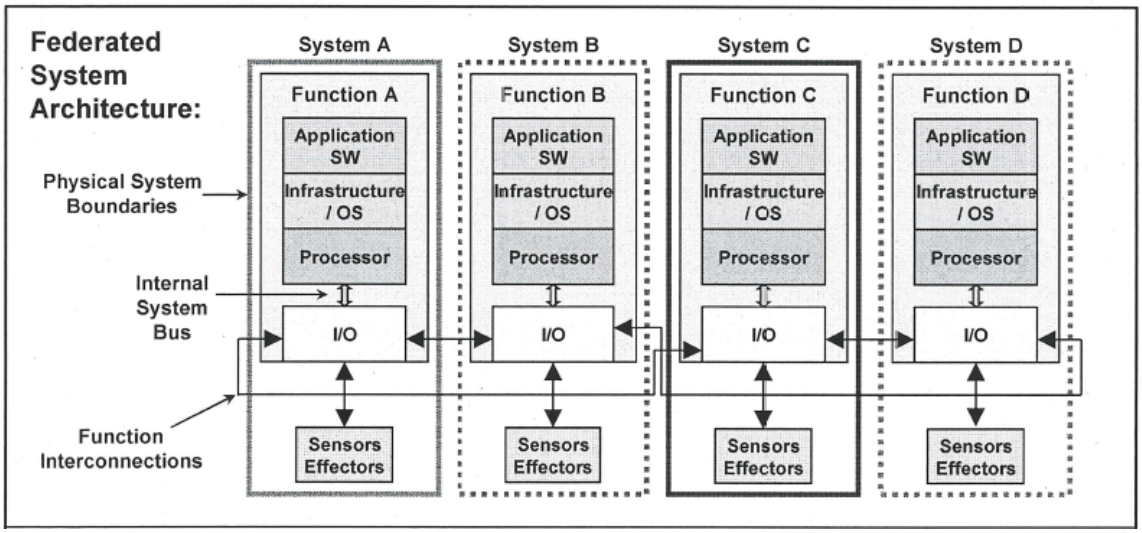


Figure 2-22 Federated Systems Architecture (Watkins, 2006)

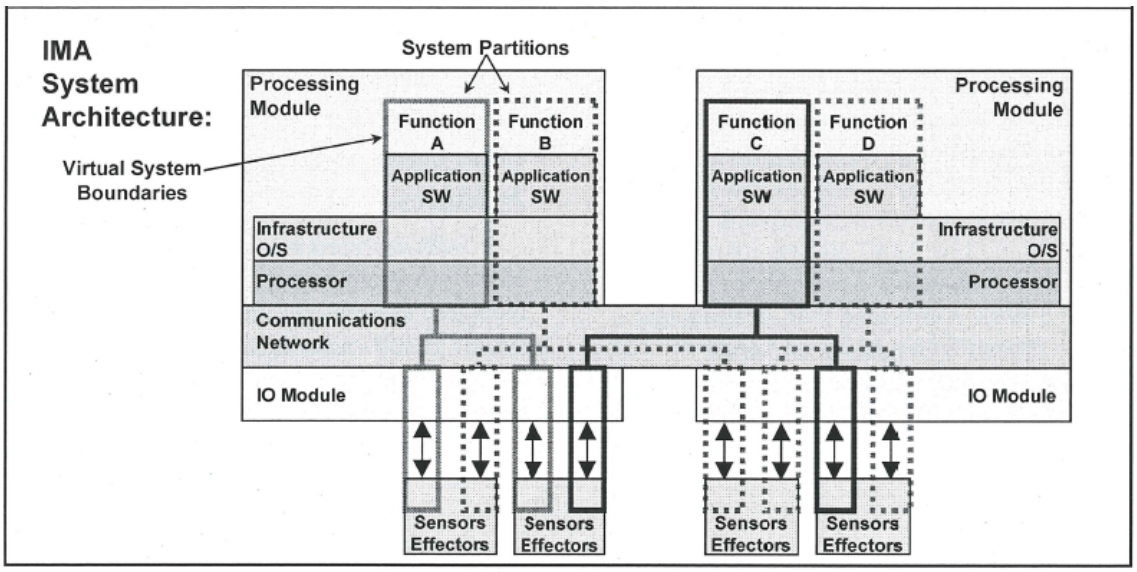


Figure 2-23 IMA System Architecture (Watkins, 2006)

The assignment of functions is performed offline and reconfiguration is facilitated by a multi-static approach. Configurations are derived using a 'contract-based' approach, i.e. by a defined set of functions and interface specifications that each component is required to have.

2.3.2. Modular Avionics Operating System (MAOS)

BAE Systems (along with partner institutions) have a long running research stream into integrated modular systems (Wake, Miller, Moxon, & Fletcher, 1997). The architecture developed for this implementation is based on the three-layer stack method described in section 2.2.1 and shown here in Figure 2-22.

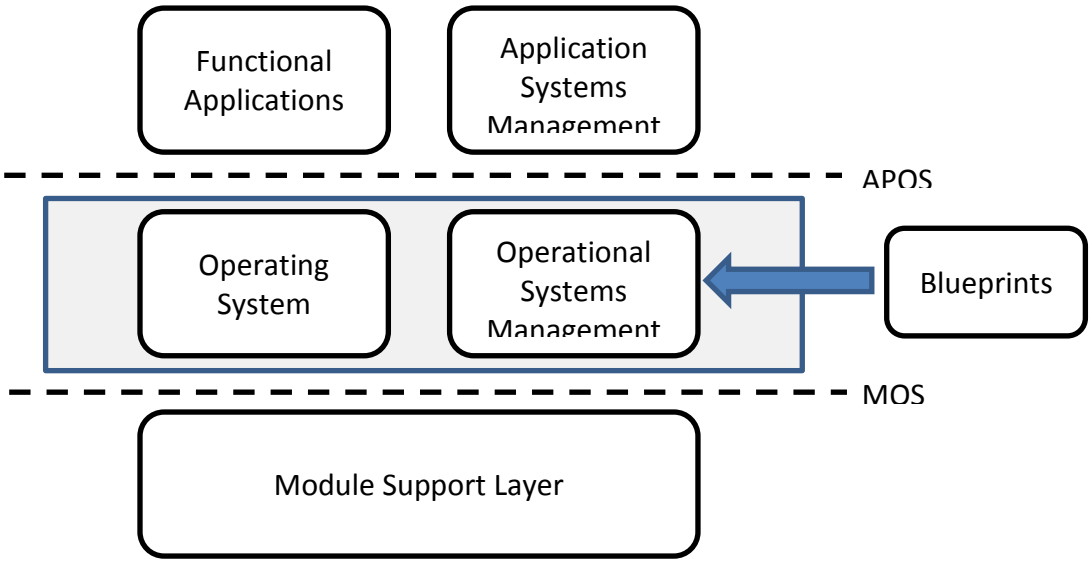


Figure 2-24 IMA Software Model (Grigg et al., 1999)

In this case, the Application Interface Layer defined in Figure 2-6 is called the Application to Operating System Interface (APOS) and the Hardware Interface Layer is the Module to Operating Interface (MOS).

The 'application manager', shown in the applications layer in Figure 2-24, is an essential part of the overall resource and scheduling management process throughout the whole system. It is implemented as an standard application with the addition of management authority as it contains information specific to the distribution. It works closely with the Operational Systems Manager (OSM), found in the middle layer, which is responsible for managed applications system-wide.

The method for deriving configuration is an offline approach that generates a number of blueprints of system configuration. These can then be accessed at system run-time in a

look-up fashion should different configurations be commanded by the OSM. This is essentially a multi-static approach as found with the Genesis system (Watkins, 2006).

2.4. Literature Review Summary

IMA remains a constantly developing field. In the past 20 years it has progress from a conceptual need to in-service products such as Genesis (Watkins, 2006) and MAOS (Wake et al., 1997). In order to achieve this level of progression, techniques and standards have been developed for important parts of IMS such as configuration management and partitioning. The development of ARINC 653 has been an important part of development as it allows a standard for multiple vendors to work towards a common set of interfaces.

Most IMA developers are progressing towards a multi-static solution in that the benefits of multiple configurations are achieved by defining system arrangements off-line and storing them in a lookup table for access during run-time. This has many advantages in terms of certification, safety cases and overall cost benefits from common, re-useable components.

(Gaska et al., 2015) highlights a key research area for IMA as modelling tools for end-to-end temporal allocation and for optimizing spatial resource allocation, both fundamentally key precursors for autonomous configuration toolsets to be realised. Adopting advanced allocation toolsets would provide benefit to the installer as automated allocation of functions to processing resources would ease the task of the systems integrator and would provide large tolerances to faults during operation.

A move towards a more flexible arrangement in IMA places greater reliance on the on-board fault management methods within the system. Any decision making element with the IMA will be reliant on the outputs of implemented fault detection methods to appropriately manage the fault. Such technologies and methods are not yet mature and require research to refine.

The main element missing from the literature is examples of dynamically reconfigurable systems. There is motivation to continue to develop aircraft with extended operational

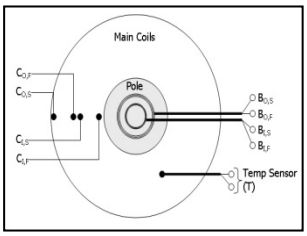
periods between required maintenance activities. As highlighted by (Mapleston, 2006b) for future avionic systems to achieve this they are required to tolerate faults, in that an occurrence of fault cannot reduce the required levels of safety such that a mission has to be aborted and an aircraft return to base. There is potential that utilising dynamically reconfigurable avionics systems would enable this. Having the ability to flexibly utilise redundant processing resource to restore full levels of redundancy upon the occurrence of a fault means that maintenance activity can be delayed to convenient periods in the operational schedule. The operational benefit of this is both in terms of mission success (completion of the mission despite the occurrence of a fault) and economics (as maintenance intervals can be planned - a move to so-called schedule based maintenance). Multi-static solutions can provide some gains in these two areas compared to an avionics system with a large degree of flexibility.

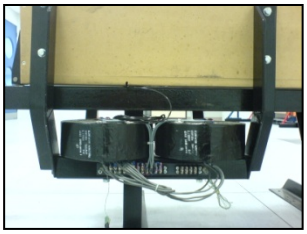
Furthermore, reconfiguring a system requires the generation of all timing characteristics throughout the system from partitioning processing resource to assignment of network bus time for communication processes. The implications of this with regards to the effect on the closed-loop, distributed control activities are required to be understood. Therefore, one of the key aims of this research is to identify a solution for dynamic reconfiguration that assigns required systems functions (i.e. a distributed, real-time control function with redundant processing channels) to available computing resources whilst protecting the functional concurrency and time critical needs of the control actions.

Dynamically reconfigurable systems currently have lack of certification routes to implementation due to the difficulty of proving the reliability of each installation during operation. This is an area of primary concern to weigh against the areas of operational benefit over multi-static solutions. It is conceivable that developers consider dynamic configuration as a ‘marginal gain’ over multi-static configuration methods in terms of operation but would require increased complexity and increased risk in developing certification methods.

When dynamic re-configuration is performed, an increased level of autonomy is assigned to the on-board system. The in-ability to arrange of functions correctly can lead to safety critical problems such as exceeding timing requirements or insufficient multiplexing of tasks for point failures. Interesting thoughts are provided in (Montano & McDermid, 2008) by considering at what point Humans are involved in the decision to reconfigure. This would require automatic generation of explanations and implications for reconfiguration actions in a manner that interfaces with pilot workload. Further research is required to understand the level of automation that can be absorbed by the reconfiguration system, and which would benefit from situational awareness from the systems operator.

The second part of this research addresses the automation problem. Here a systems management strategy is to be researched that utilises dynamic reconfiguration (along with automatic health assessment of the system) to restore an IMA to high levels of redundancy following the occurrence of a failure. The output of this work will bear in mind the requirement to communicate the reconfiguration activity to either a pilot or a remote systems operator in real time.





CHAPTER 3:

IMA Demonstrator System Requirements

3. IMA Demonstrator System Requirements

3.1. Introduction

The purpose of this chapter is to define the capability requirements of the IMA demonstrator in order to investigate and demonstrate the fault management techniques within IMA. This chapter will use information gathered in the literature review chapter that outlined the techniques implemented in IMA and the identified gaps in capability.

These requirements will not form a formally agreed requirements set, but will identify in a logical fashion the outlined requirements for each part of the system. Each section contains will describe the composition for this part of the system and a description of the functional process. This is then summarised as a set of top-level requirements. The goal here is to justify that the developed system is appropriate for verifying techniques in fault management for IMA.

3.2. Top Level System

The system to be designed has the overall aim of providing a test platform for fault management within IMA. The assignment of an IMA to a real time control task will provide proof that the systems management will be operating successfully whilst providing a high level of service. The overall demonstrator system will consist of the three main elements highlighted in Table 3-1.

Table 3-1 Component Composition of Top Level System

Item	Description
Maglev Rig	The vehicle chassis, sensors and actuators in the form of magnetic coils
IMA	The hardware and software elements that make up the distributed avionics architecture
System Operator	The trained operator who at the current time is interfacing with the system

Figure 3-1 shows how these three elements interact. The 'systems operator' (i.e. the operator of the rig) will be able to interface with the IMA via a Graphical User Interface (GUI) and, in turn, the IMA will pass drive signals and receive sensor data from the Maglev hardware. The real time control aspect that is required to maintain electromagnetic levitation will therefore be performed by applications situated in the IMA.

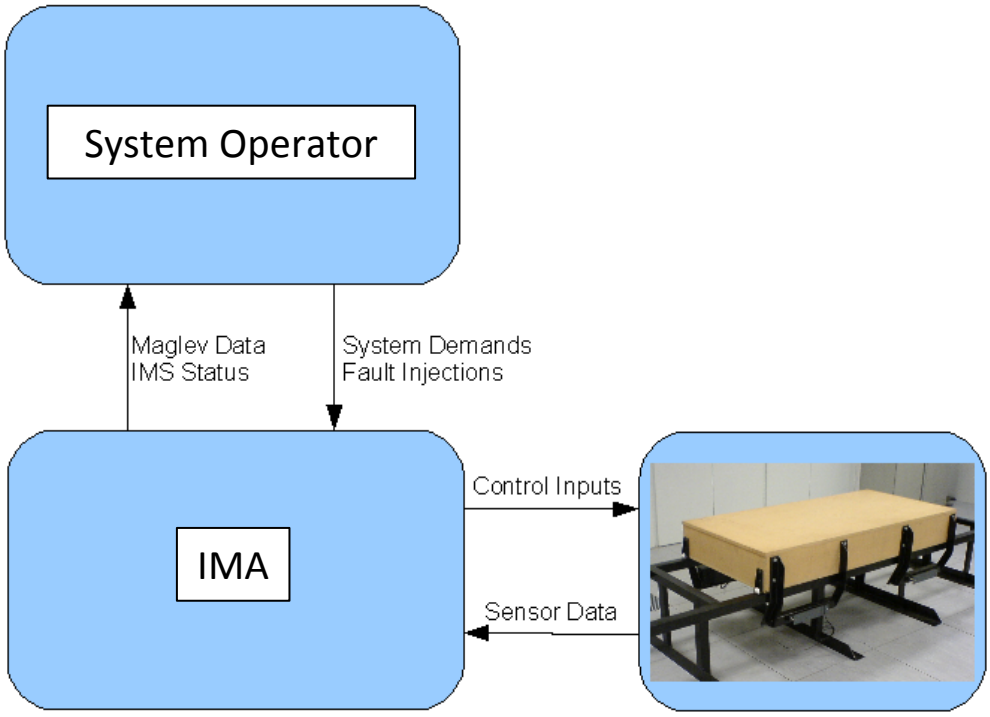


Figure 3-1 Top Level Systems Diagram

The purpose for the development of this system is to provide a tangible demonstration of the concepts under investigation. This demonstration will be a suitable response of the IMA to an injected fault signal. It is worth noting here that because this is a developmental exercise, it is not intended that the system is designed in order to cope with any random fault mode, but be expected to respond appropriately to a number of identified common faults.

To observe a response to a fault, it is intended that the system is capable of having faults or appropriate fault signal inject during operation. This could be done manually via manual interference with signal routing, or by raising a fault code to simulate the identification of an occurrence. The system will have to make available appropriate data to show that a reasonable response to the fault injection has occurred.

Therefore, the overall operating requirements are:

1. The IMA shall be able to display to the operator appropriate internal systems management information (e.g. current functional allocation, and current component health) for verification purposes.
2. The IMA shall be able to display to the operator real-time information regarding the target platform (Maglev rig), such as current position.
3. The IMA shall be capable of receiving control commands for the target platform.
4. The IMA shall perform the real time control functions necessary to maintain magnetic suspension of the Maglev rig.

The requirements for demonstration of fault management within IMA

5. The IMA shall allow user to inject fault signals into the system
6. The IMA shall perform some identified techniques in order to manage the fault
7. The IMA should maintain service in the presence of fault if possible
8. The IMA shall provide a tangible output of the systems management actions during this response.

3.3. IMA

One of the main constraints of the project is that there is no previously commissioned IMA architecture available for further development or modification. It is therefore necessary that a complete design and construction of a representative IMA is required. It was highlighted in Chapter 2 of that an IMA is an incredibly complex combination of equipment and software. As the timescale and resources of this project are limited, an industry standard IMA will not be constructed. The main requirement of this system is that it will represent the key fundamental capabilities of an IMA that are necessary to demonstrate the research achievements.

The general design of each IMA module should follow the three-layer stack architecture, as identified in the literature review and shown here in Figure 3-2.

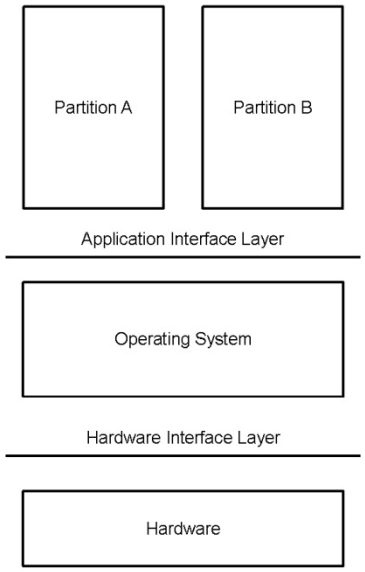


Figure 3-2 Three Layer Stack

It is essential that the system generated represents the actions of IMA, but does not need to strictly adhere to all aviation requirements. For example, it is essential that applications are independently controlled, but it is not essential to employ partitioning schemes that segregate the applications under all instances of operation or failure. As many of the key fundamental principles of IMA should be followed during systems development to ensure the appropriateness of the final system for the verification of techniques employed.

The functional requirements of the IMA are met by the implementation of appropriate 'applications' within the structure. A specific example of an application in this case will be one to control the 'airgap' between the magnet pole and the rail (Chapter 4). The execution of these applications is supervised by the operating system, often referred to as the 'middleware'. The operating system has a number of required functions. It is responsible for managing the access that the applications have to the hardware services such as processing and communication resource. It is also responsible for general systems management tasks such as the boot process and managing faults arising in the system. As such, the hardware needs to be capable of handling the software elements mentioned and provide the hardware capabilities required. In order to interact with the other system elements highlighted in Figure 3-1, the IMA will need to have analogue data input/output capabilities and a suitable interface to allow the operator to control the system elements and observe the internal states of interest.

Figure 3-3 shows how the modules of the system can be interfaced with a network in order to interact with each other. This diagram highlights how additional hardware can be added to the standard module in order to provide interfaces to sensors and actuators. The final element of note is that a Graphical User Interface (GUI) can be added to the network to provide the required data exchange between the system and the operator.

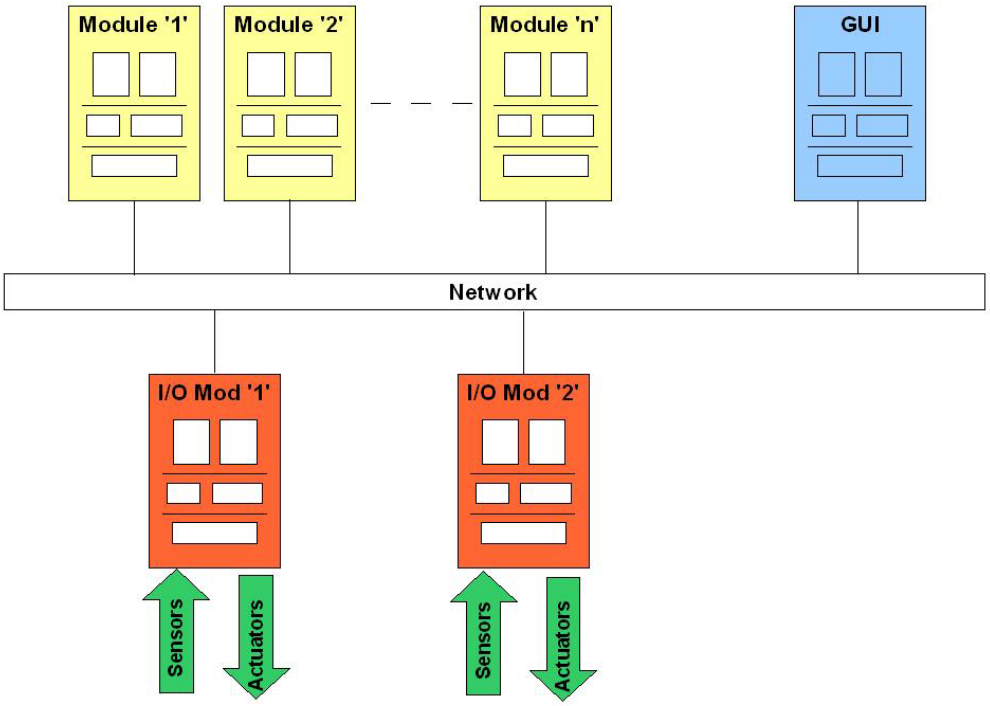


Figure 3-3 IMA Module Integration

The baseline operating requirements are:

9. The design of the system shall follow fundamental IMA principles where appropriate
10. Each IMA processing module shall follow the three layer stack architecture
11. The hardware and software shall be loosely coupled such that, within a limited scope, a change in either shall not infer a change in the other.
12. The operating systems layer, or middle layer, shall manage the availability of the hardware resources to the application layer

The required functionality for demonstration:

13. The operating systems layer shall manage the arising of a limited number of faults within the system
14. A tangible output shall be created showing the management of the fault
15. The service provided by the system shall not be interrupted during the management of the fault

3.3.1. IMA Hardware

All the IMA operating system and applications require hardware resources in order to function. In Chapter 2, examples of avionics hardware were highlighted that would be capable of performing such role. Due to the constraints of the project, avionics standard equipment cannot be incorporated. It is therefore important to identify the key elements that are required and select hardware that can accommodate these.

Essentially, the applications and operating system will require basic computing needs, namely:

- Appropriate access to processing resource to execute function
- Access to physical memory for storage of internal variables
- Access to communications channels
- Access to data storage facility

Further to this, Figure 3-3 highlights that some modules require extra hardware functionality in the form data acquisition and a graphical user interface in order to interact with the operator and the target plarform. An ideal solution here would involve a standard processing board with expansion slots to provide extra capabilities where required.

In the spirit of IMA, the hardware should be capable of being replaced without affecting the written software. This would suggest that the use of a higher-level language (such as C++) compiled on to generic type hardware would provide a reasonable solution. Furthermore, hardware rating to avionics standards of reliability will not be required for the relatively short time scale of operation expected of the test rig.

The overall requirements are:

16. The hardware shall be capable of accommodating the required system functionality in terms of processing resource, communication resource and I/O expectations.
17. To a limited scope, hardware shall be generic and a change in a hardware component will not affect the fundamental design of the rest of the system.
18. The hardware shall have data acquisition capabilities where appropriate

19. The hardware shall have a networking capability of deterministic communications
20. The hardware shall be expandable to allow additional hardware capabilities to be included

3.3.2. IMA Middleware/Operating System

The systems management that forms the middleware of the IMA is the main focus of this project. It is also a critical part of any IMA system as it is responsible for the overall management and decision making for the system. The middleware architecture to be adopted is represented by Figure 3-4.

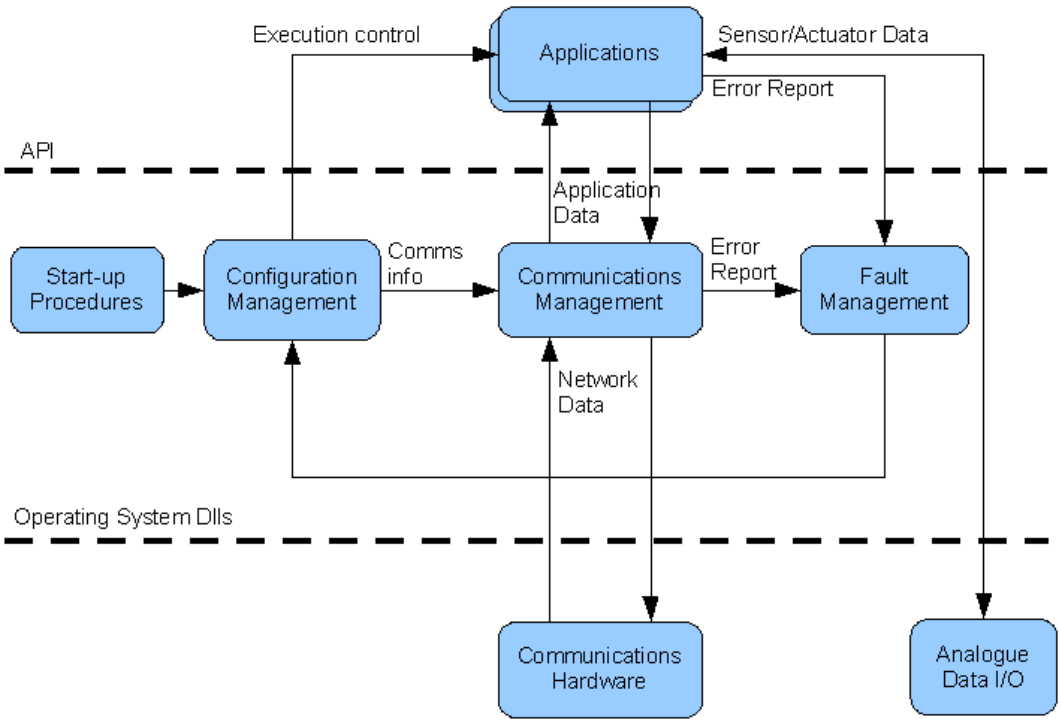


Figure 3-4 IMA Middleware Architecture

An IMA comprises of an interacting system comprising of a number of modules connected over a network (Figure 3-3). Therefore managing the IMA is a system wide problem requiring communication and coordination between the nodes, more so than is highlighted in Figure 3-4. This infers that processing time and communication time will have to be made available for systems management activities along with the real time requirements of the module.

The concept of the system shown in Figure 3-4 is that the configuration of the assignment of the applications to resources can be changed during system operation. The configuration management function will have to have prior knowledge of the overall functional requirements of the IMA in terms of the expected applications, their communication needs and their own timing requirements. It will also require information regarding the current health state of the hardware within the system to prevent allocation of functions to a faulty component. An output of the configuration algorithms will be a set of instructions to be distributed throughout the network detailing the application assignment and the communication structure.

Figure 3-4 highlights how each application accesses communications channels. As opposed to directly interfacing with the hardware, the data goes through a communications manager in the middleware. By doing this, the communications manager cannot only make the network protocols transparent to the applications, but also organises the data such that it can be sent in a deterministic fashion.

An important part of the middleware for this application is the fault management function. The purpose of this investigation is to perform appropriate fault management techniques in order to maintain the service in the presence of faults. This fault manager will need to obtain error messages from all significant components of the system, i.e. hardware, applications, other modules, in order to make a decision on the appropriate action to take. If this is to be an instruction to reconfigure, then this command can be passed back to the configuration manager with the new restrictions on assignment criteria.

The requirements for the IMA middleware are:

21. The middleware shall manage the administrative aspects of the IMA across all modules, inclusive of start-up, communications, configuration and faults.
22. The middleware shall manage accessibility of hardware resources to applications.
23. The middleware shall enable the execution of applications and communications to be performed in a deterministic manner.

24. The middleware shall allow the software and hardware disparate such that a change in one should not infer a change in the other.
25. The middleware shall prevent any unwanted interaction between applications to a reasonable degree.

3.3.2.1. Management of Application Access to Hardware Resources

The common theme of the middleware activities is that of managing the way applications interact with the hardware on which they are housed. The purpose of this is to decouple to software from the hardware such that their development becomes modular. This is the fundamental property that provides many benefits as highlighted in the literature review in Chapter 2.

Most of the interactions between software and hardware come from standard library files and commands of any normal operating system. For example, if the code requests an additional action, this is interpreted by the operating system into low level commands to the computer hardware in order to return a result.

A complication within IMA is that of having a number of applications executing upon the same module. The middleware has to be responsible for ensuring that applications can share resources whilst ensuring that they execute to the timing schedules necessary to perform deterministic functions. Further to this, the middleware should be responsible for ensuring that applications are segregated from the operations or failures of other applications running on the same module. These functions will run alongside the communications needs highlighted in Section 3.3.2.4.

Following the design principles of IMA, the final need from the middleware is that it is designed to be portable and modular such that it can operate on similar but different hardware. This should allow the hardware to be incrementally changed without affecting the operating system, and vice versa.

Avionics standard middleware has to ensure key functional needs such as partitioning and deterministic execution to a very high degree of fidelity. For the purposes of this investigation, these items should be part of the developed code but need not match the same high levels of reliability or robustness.

The requirements of this component of the system are:

26. The middleware shall share systems resources between applications
27. The middleware shall partition applications from all others
28. The middleware shall ensure that applications are executed in order to meet their timing constraints.
29. The middleware shall be modular and portable

3.3.2.2. Configuration Management

The purposes of the configuration manager will be to derive and implement an appropriate assignment of software applications to available resources. In order to achieve this, the configuration manager will require to know:

- Details for each of the functional applications, such as how they interact and any specific assignment requirements
- Details of each available resource on the network, such as signal input/output capability or graphics output
- Constraints for the configuration such as any failure reports of components

From this information, it should be possible to either derive a configuration or report an error message.

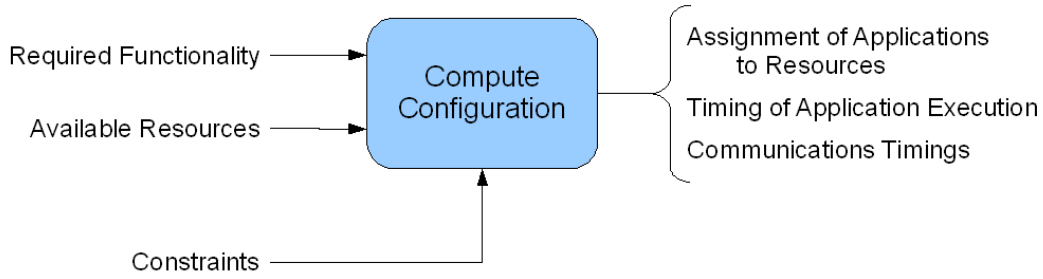


Figure 3-5 Configuration Manager

A further complication to the configuration problem is that of assigning a schedule to the execution of functions. If the system is to be deterministic, there will be a requirement for critical functions of applications distributed across the network to run concurrently and meet execution deadlines. This requires the synchronisation of execution network wide and synchronisation of communications.

The goal for this project it to utilise a method whereby the configuration is generated dynamically. This will mean that for any functional set of applications prepared and any number of processing modules attached, the system will generate either an appropriate configuration where possible or an error message. This also means that should a fault occur in a system component, the configuration computations can be re-executed with the new restraints in place.

The requirements are:

30. The configuration manager shall manage assignment of applications to resource following application assignment requirements
31. The configurations generated shall ensure the applications can execute in a deterministic fashion
32. The configuration/reconfiguration tasks shall be performed in a timely manner
33. The process of configuration/reconfiguration shall not disrupt the application execution flow
34. To recalculate a new configuration based on constraints from fault signals
35. To Implement a new configuration without interrupting the service

3.3.2.3. Fault Management

The literature review in Chapter 2 identified that in order to provide fault management, 4 key items should be considered:

- Fault avoidance
- Fault removal
- Fault tolerance

- Fault treatment

Fault avoidance and removal are largely concerned with good systems engineering practices in design and are not the subject of the investigation here. Fault treatment is often covered by a good support network that can diagnose the fault and replace the failed component. The focus in this investigation is that of fault tolerance, as in how to accommodate the occurrence of a fault whilst maintaining the intended service output. The following items were identified in the literature review in order to perform fault tolerance:

- Error detection
- Error handling
- Error confinement
- Error recovery

It can be seen at this stage that Figure 3-4 over simplifies the process of the fault management system. The inclusion of a fault management mechanism, or more specifically in this case fault tolerance, infers requirements on all aspects of the system.

Ideally, the system will have error detection mechanisms for each component of the system with a structured fault reporting system that feeds back to a top level. At this central location, a decision can be made as to the next best course of action, based on the information provided. If, for example, the solution to the recover from the error is to reconfigure then the configuration mechanism can be activated with the new constraints applied.

A useful addition to the fault manager is that of an error log. This will have a record of any identified faults and the result of the handling decision. This will provide an important output to the verification of the fault tolerance process.

The requirements for fault management for this application are:

36. The system shall be capable of detecting simple faults in various system components, such as applications or hardware modules.
37. The system shall be capable of handling the occurrence of certain errors.
38. The system shall be capable of recovering from error by initiating the reconfiguration mechanism with new constraints.
39. Where possible, the system shall confine faults such that their occurrence does not propagate system wide.
40. The system shall maintain a log recording the error that has arisen and the resultant action taken.

3.3.2.4. Communications Management

A large part of the IMA problem is managing this loosely coupled relationship of hardware and software in a deterministic fashion. From the application perspective, data is expected at a certain instance in time and makes the output of the function available upon completion of execution. The communications manager, as shown in Figure 3-6, will be responsible for taking this data and sending it to the appropriate application for which it is intended.

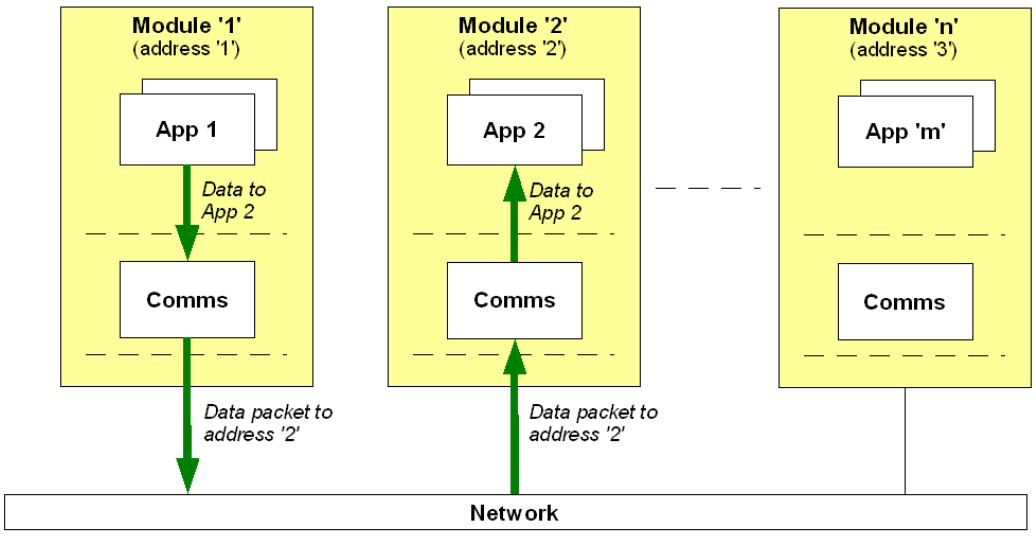


Figure 3-6 Application Data Exchange Example

As an output of the configuration algorithms, the communications manager requires to be informed of the timing characteristics for the functionality of the system. It is then responsibility for managing the series of communication to achieve real time, deterministic

communications whilst maintaining transparency of this process to the housed applications. This can be done by interacting locally with the applications housed on the same module to retrieve the necessary data and then send this data via a network to the appropriate module that houses the target application. The data can then be processed by the local communications manager on the receiving module to be passed to the correct application.

The communications strategy requires flexibility such that should the functional assignment change, the communications structure will also be required to change. It is important that should this occur, the transition should be smooth and cannot be allowed to cause disruption to the service.

The communication structure is not only responsible for sharing information between applications across the network. Previously, it has been mentioned that aspects of the systems manager will require communications between modules in order to obtain a global picture. It is important that the systems management communications, that may be non-deterministic, does not impact on the time critical communications.

The requirements of the communications algorithms are:

41. It shall provide deterministic communications between applications for assigned configuration
42. It shall ensure that the network communications methodology is transparent to the applications
43. It shall provide communications between systems management components
44. It shall Ensure that communications of systems management do not impact upon the deterministic communications
45. It shall ensure smooth transition between configurations

3.3.3. IMA Applications

The applications within the IMA are the components that will be performing the functional tasks assigned to the system. The code within the applications will be task specific, in this

case it will be the control algorithms and any supporting functions desired to maintain the airgap on the maglev test rig.

The challenge here is to ensure that the modularity of the system is protected such that, should the applications or their functional structure change, there should be no change inferred on the underlying middleware. The application will have to conform to an interface standard such that they can be implemented on any IMA module. The main part of this refers to the communications structure, as mentioned in section 3.3.2.4.

Where possible, applications have to be written such that they are not hardware specific. In some certain circumstances, applications will require input from specific hardware component such as a sensor or a keystroke, and therefore the application requires locating on a module with this capability. The application will be require to state this to the configuration manager such that this assignment is achieved.

In addition to this, applications have to be controllable in that the systems manager has the capability to start/stop applications, schedule the execution and initiate their execution in another part of the system.

The requirements of the applications are:

46. They shall perform a specified task
47. They shall be controllable by the system manager
48. They shall communicate with other applications via the communications structure
49. They shall have an assignment specification

For the purposes of the demonstration, the primary requirements for the applications will be to maintain the airgap of the Maglev vehicle.

3.3.4. Graphical User Interface

The GUI in this application will be required to have two main functions. The first will be the ability for a user to control the inputs associated with the Maglev rig and to monitor any

sensor output. The second is to provide an input/output capability to facilitate a tangible demonstration of the IMA systems management capabilities.

In order to perform the input/output requirements of controlling the rig itself, the GUI will need to take an airgap demand from the user, pass this to the control algorithms for processing and output data from all the associated sensors. It may also be required that test inputs, such as a sinusoidal input, be used for analysing the control algorithms.

A more important output with regards to this investigation will be the interface detailing the systems management activities of the IMA. The GUI should show the current configuration of the system in terms of allocation of functional components to processing resources. It should also show the relationships between the applications, as in the flow of data and parallel processing channels. Further information to display can include information about each system component and if a component is in a faulty condition. The most important consideration is to make the information tangible to the observer

The baseline requirements are therefore:

50. The GUI shall provide an input capability for the demand of the Maglev rig
51. The GUI shall provide an output of sensor data from the Maglev rig
52. The GUI shall provide a test input capability

For the purposes of the demonstration Fault Management within the system:

53. The GUI shall display the configuration of the system in a tangible manner
54. The GUI shall highlight the process flow of the functions
55. The GUI shall display the health of the system components
56. The GUI shall display detailed information about each component
57. The GUI shall provide a fault injection capability

3.4. Maglev

The main requirements instilled on the maglev rig are that it is controllable and measurable. The rig should respond to the drive signals given to it, and output reliable sensor data.

The rig is designed to be controlled by low voltage PWM signals, one for each magnet. These signals are required to be amplified in order to provide sufficient power to the magnets in order to magnetically suspend the vehicle.

The sensor outputs of the rig have different levels of critically, in that some signals are used for the real time control aspects, and others are used for general monitoring purposes. Table 3-2 below shows the sensors already installed on the rig and their purpose for use.

Item	Description	Purpose of Use
Flux sensor	Coil of wire embedded onto magnet poleface. Signal is integrated to a voltage proportional to Flux density.	Critical - used for real time control.
Gap	Sensor embedded to the underside of the chassis. Outputs a voltage proportional to gap size.	Critical – used for real time control of airgap.
Current	A Hall effect sensor that outputs a voltage proportional to current.	Can be used for real time control, but more likely used for monitoring purposes
Temperature	Thermocouple embedded into the coil.	Non-critical - Used purely for monitoring.

Table 3-2 Maglev Rig Sensor Descriptions

For all sensors it is important that a reasonable level of reliability is obtained such that sensor failure does not impact development. The sensors should provide as clean and repeatable signal as possible otherwise this will impact on the control design. It is a reasonable extension here to design the signals such that they can be failed in order to observe how the fault management system responds.

The rig has been designed with redundancy in mind, by duplication of certain elements. Each magnet is dual wound, such that it has two separate excitation inputs. Each poleface has two search coils embedded into the poleface to measure magnetic flux and two temperature sensors. It is therefore possible to take advantage of this arrangement in that sensor channels or actuation channels can be failed, and it will be up to the IMA to identify this fault mode and respond to it. Figure 3-7 shows the possible duplex sensing arrangements for interfacing with the magnet. Full details of the design and build of the experimental rig can be found in Appendix C

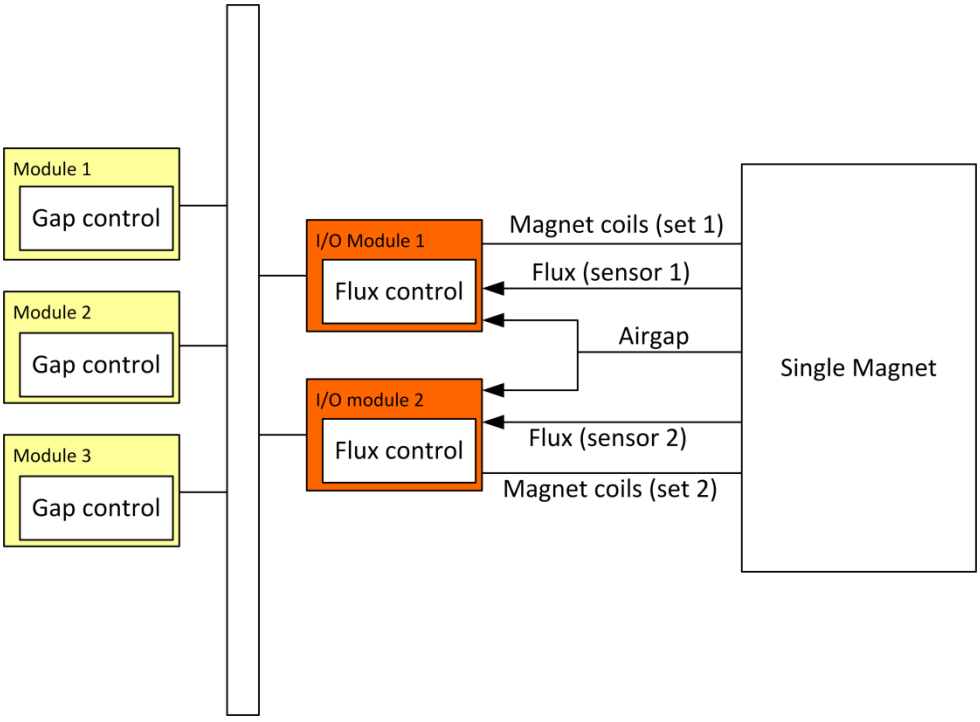


Figure 3-7 Suggested sensor interface between IMA and Magnet

Therefore, the requirements are:

58. The Maglev rig shall be controllable and respond as expected to inputs.
59. The Maglev rig shall provide clean, repeatable signals.

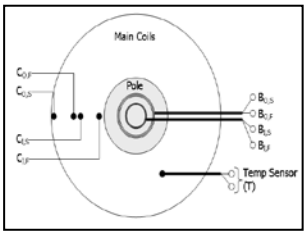
For the demonstration

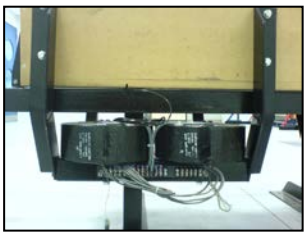
60. The Maglev rig shall allow faults to be injected into it.

3.5. Summary of Requirements

The requirements presented here summarise the major considerations for the development of a test rig that can demonstrate the main research outputs. A key challenge is that of creating a network of computing modules that represent the fundamental functionality and attributes of IMA in terms of facilitating the execution of application whilst maintaining transparency of the underlying technology and processes. The aim of defining and then replicating the features that are essential to fault tolerance is to ensure the scientific findings are appropriate to a robust standard architecture.

The requirements are shown to have been satisfied by the Requirements Compliance Matrix that can be found in Appendix B.





CHAPTER 4:

IMA Implementation

4. IMA Implementation

This chapter describes the design and function of each component of the physical implementation of the representative IMA for this project. A key aspect of this chapter will be the description of how software applications are assigned and executed on the available processing resources. The ability to sensibly and reliably assign different networks of functions to varying levels of resource will be tested in Chapter 6.

Although fault management within this system is intrinsic to its design, the methods of managing faults and how the system responds to these will be discussed at depth in Chapter 7.

4.1. Top Level Design

A fundamental choice of the project was that of the computing capability. The solution chosen was to use a real time operating system provided by National Instruments running on a standard PC. This real time operating system (RTOS), called Pharlap, is also known as LabVIEW ETS (Embedded Tools Suite). When installed as an operating system on a PC, it will execute standard LabVIEW code in a deterministic fashion. The library files contained as part of the RTOS provide the interface needed for the software to access hardware resources such as networking capabilities or sensor inputs.

The issue highlighted with this solution is that LabVIEW ETS is not designed as a middleware solution to an IMA implementation. Therefore a certain amount of augmentation with custom designed code is required to fulfil the functions necessary for this project. The implementation of this solution, as a comparison to the 3 layer stack philosophy, is shown in Figure 4-1.

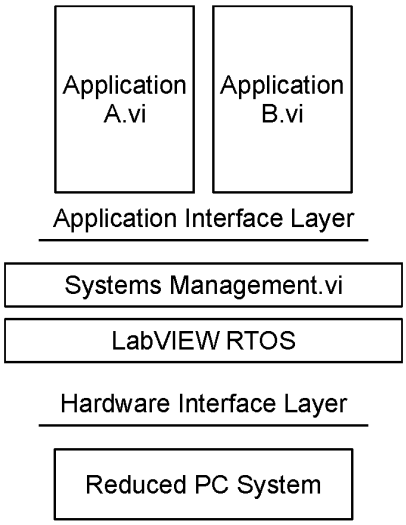


Figure 4-1 Practical Implementation of IMA

The PC itself contains as few extra components as possible. The complete hardware list consists of:

- PC motherboard
- Intel Celeron D Processor
- RAM
- Hard disk drive
- Intel network PCI card
- Data acquisition card (where appropriate)

One stipulation using LabVIEW's Pharlap as the operating system is there is a restriction inferred on the choice of components used above. Pharlap is only compatible with a certain number of processors, network PCI cards and National Instrument's own data acquisition cards. This fundamentally works against the principles of an open architecture in which there are no restrictions on compatibility. For a commercial implementation for IMA, this level of constraint would be unacceptable, but for an investigation into systems management activities, the solution is adequate.

Aside from this restriction, the solution will allow the demonstration of fundamental IMA principles. During development of the system, the original motherboard selected was outmoded and hence irreplaceable. Instead an upgrade of similar specification was

purchased and this was inserted as a replacement into the system with no change required to the rest of the system.

In summary, the benefits and issues of using this arrangement are highlighted in **Error! Reference source not found.** below:

Benefits	Issues
• Provides a flexible, workable solution to the implementation of a representative IMA • Upgradeable (with certain restrictions) • Cheap • High performance computing	• Reliability • Augmentation required to perform middleware functions • Not 'future proof' • Not an open architectural solution

Table 4-1 Analysis of LabVIEW implementation of IMA style middleware

The overall selection above provides a reasonable solution to the problem. A number of key functions, as highlighted in Chapter 3, will be required to be written to mimic a true IMA middleware. The following sections will detail how these functions are implemented.

4.2. Systems Management

An IMA can be considered as a distributed system in that the functional components are spread across a number of computing resources connected by a network. One of the questions this raises is that of decision making and, more importantly, which element makes which decision. Major decisions which encompass the whole IMA such as configuration selection will be made at a central location with the result communicated to other modules. This results in two considerations, firstly the need for a good reporting structure and secondly the introduction of a critical single point failure.

4.2.1. Systems Management Reporting Structure

The structure of management within the system takes a hierarchical approach whereby there exists a global systems manager, responsible for making system wide decisions,

working with local systems managers that control and manage each module. Local systems managers will be responsible for reporting faults within these modules to the global systems manager and acting upon any instructions received, as shown in Figure 4-2.

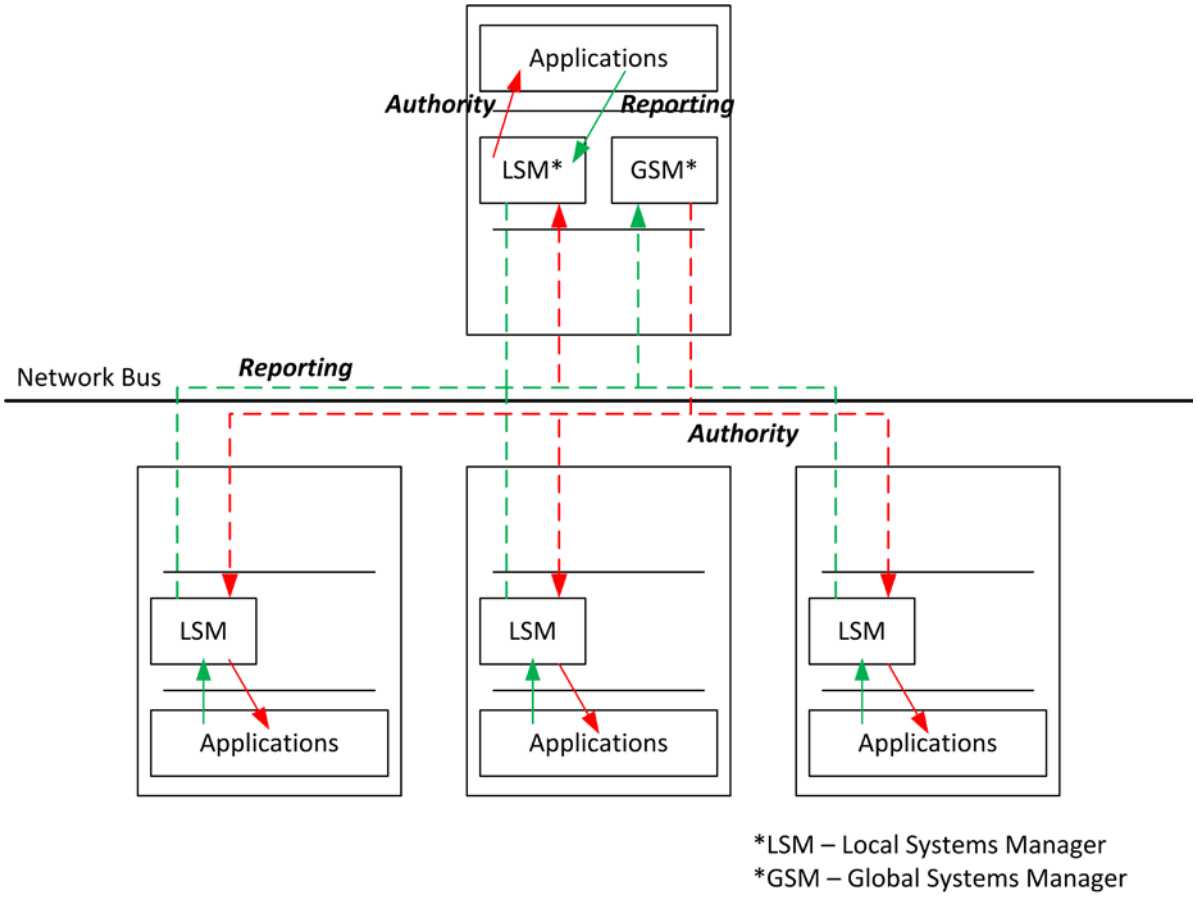


Figure 4-2 Systems Management Hierarchy

The global systems manager constantly monitors the health of the hardware components and applications of the system. For simplicity, these items are considered healthy or faulty. The status of each of these items are retrieved from a number of sources, and collated here to gain a systems wide view. Should the status of the system change, such as a processing module or application is reported to have failed, the global systems manager initiates the reconfiguration algorithms to optimise the allocation of required functionality to remaining resources. The new configuration is then communicated to the rest of the system and implemented by the local systems managers.

4.2.2. Timing and Synchronisation

The reporting and communication of systems management activities require bus bandwidth to function. It is important that this communication does not interfere with the 'hard' real-time functions responsible for real time control loops.

It is assumed that the initial system configuration contains appropriate levels static redundancy. This ensure that at the point of a failure occurring during a mission profile, the system can continue to provide service during the time it takes to assess the new system capability and decide the appropriate course of action to take. At any point, if a failure occurs and no reconfiguration of the system will improve current capability as all redundant options have been exploited, then the health assessment can be used to inform the decision to continue the mission, return to base or ditch the aircraft.

Systems management activities are not 'real-time' functions in that the ability of them to gather data and function is not affected by transient problems (such as 'jitter', signal delay and requirement of regular data transfer) as real time control functions are. Communication of systems management functions are allocated time on the network bus following the completion of real-time functions, as shown in Figure 4-3. This is performed by dividing bus time into regular time frames, synchronised by a data packet from the global systems manager. Each time frame is segmented such that the systems management activities are assigned bus communication bandwidth where interference with real time functions is not possible.

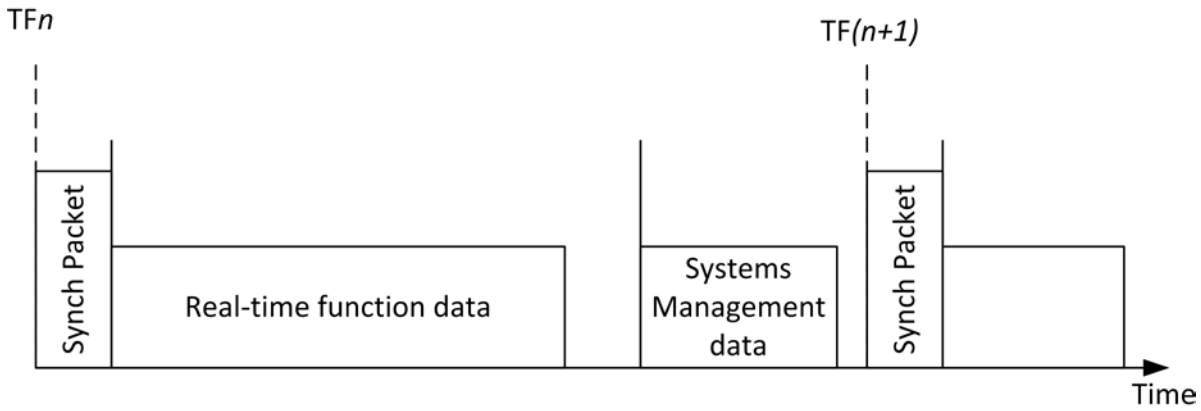


Figure 4-3 Time Partitioning of Network Bus using Segmented Time Frames (TF)

This segmentation enables reconfiguration activities to be arranged in the background whilst the system continues to operate. Following the identification of a failure, information regarding the health is reported via the communication structure to the global systems manager. At this stage the global systems manager attempt to identify a new configuration that would restore higher levels of redundancy. As mentioned, if all options have been used due to previous failures of available system components then the system is reliant on the levels of static redundancy in the current configuration. If reconfiguration is advised, the new configuration is communicated to the local systems managers whilst the system continues to operate in its current state of reduced redundancy. After the new configuration is prepared system wide (in that each module is aware of what the new assignment and timings of applications and communication packages will be), the new configuration is implemented by a message embedded in the synchronisation packet in the time frame. Each module instantly discards the previous configuration and implements the new. Changes have to be instant and occur at system wide synchronously to ensure that no loss of service occurs during transition between configuration states.

4.2.3. Redundancy in Systems Management

Single point failures are traditionally compensated for by the implementation of redundant processing channels. A similar solution could be applied here to provide a reversion option should the module acting as the global systems manager fail.

As all processing modules in the system are very similar, the choice as to which one performs the global systems management is arbitrary. In this case a node at a fixed IP address is chosen and this module is assigned the role of Global Systems Manager at start-up.

The proposed conceptual solution to the problem is to not only have a global systems manager processing health and systems information, but for its actions to be replicated and monitored by a number of modules aptly named Shadow Systems Managers. These will mimic the actions of the global manager and look for any discrepancies or bad outputs. Should the global manager be deemed to have failed, one of the shadow managers will be able to assume authority with little or no delay.

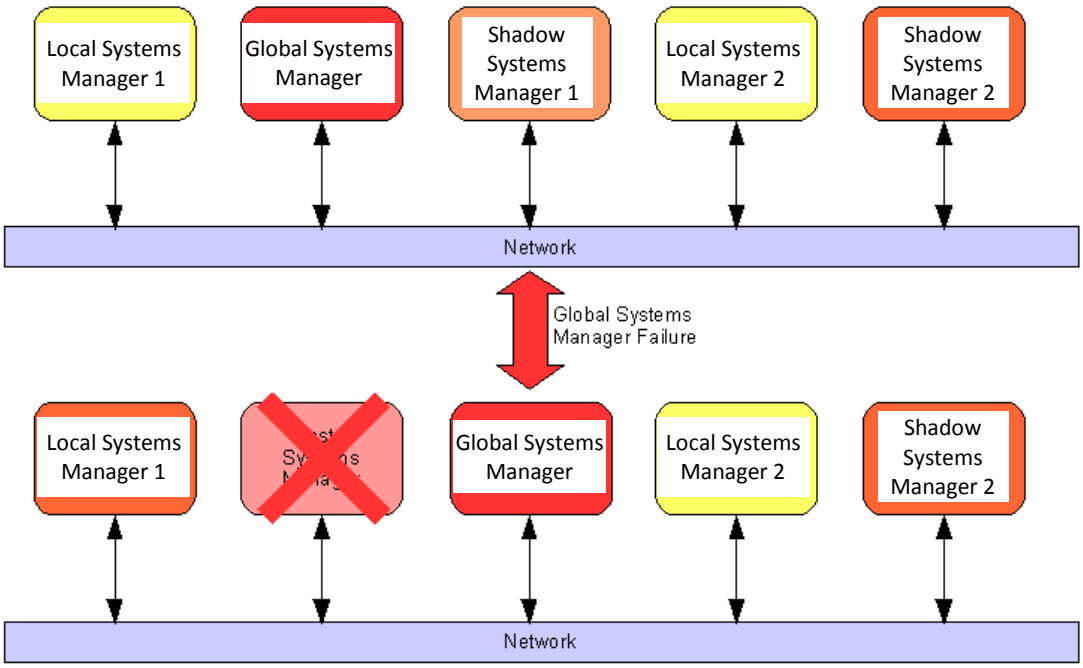


Figure 4-4 Systems Management Tolerating Failure

Figure 4-4 focuses on the middleware sections of the IMA. Reconfiguration in this instance will have to occur in this system on two levels. First of all, the systems management level itself needs to be reconfigured dynamically. Following this, the application level needs to be reconfigured in order to restore the system to an optimum level of functionality and redundancy.

The solution proposed is a conceptual one in that it has not been employed in the developed system. It is described here to justify the chosen solution of a global systems manager is a viable one, but in a more demanding environment than this project further considerations must be made to guard against the occurrence of a failure.

All of the management level reconfiguration must occur in a transparent manner to the applications.

4.3. Application Design

In order for the system to control the application with regards to the execution and assignment, the system will have to know about the application. Statistical information regarding each function requires storing in a location accessible to any decision making element within the middleware. This information, along with the assignment of each application, forms what is known in IMA as the system Blueprint.

The information required to be stored in the Blueprint can be found in Table 4-2. Most of the information is a textual description of the task orientated functions of the system and can almost be captured directly from an architectural systems analysis. These items are filled in automatically as part of the start-up procedure. The rest of the content is populated as a result of the configuration algorithms and will be discussed later.

Element	Description
Application Name:	The descriptive name of the functional component
Application Reference #:	A unique number assigned to the application
Assigned to:	The processing module identification number to which the application is assigned
Inputs from:	Details the name and source application for each of the inputs
Outputs to:	Details the name and target application for each of the outputs
Assignment criteria:	Contains information relating to the possible assignment of each application. This is based on the functional description of the system, but essentially ensures that the applications are assigned in such a way not to jeopardise any redundant channels and to ensure that applications that require certain hardware are assigned to the appropriate hardware resource. These will be ranked according to importance and the least important criteria will be removed should there be no logical solution to the configuration. Example criteria: !app1 - cannot run on same module as application 'app1' !Input 1 – the application that is the source of input 1 must be on another module
Criticality:	States the level of importance to the system. The application is flagged as time critical if it is the last function in a network of distributed real-time functions. This enables priority over less critical functions (such as data display) and could be graded depending on the task. (See section 4.4.1.1 for more information)
Run time:	The time it takes the application to execute based on worst-case execution time assessment
Start time:	The time the application is due to start in the time frame (NB It is assumed that each application runs once every time frame)

Table 4-2 IMA Blueprint Outline

4.3.1. Application specification example

Figure 4-5 shows a network of functions that have to be performed in a set time frame.

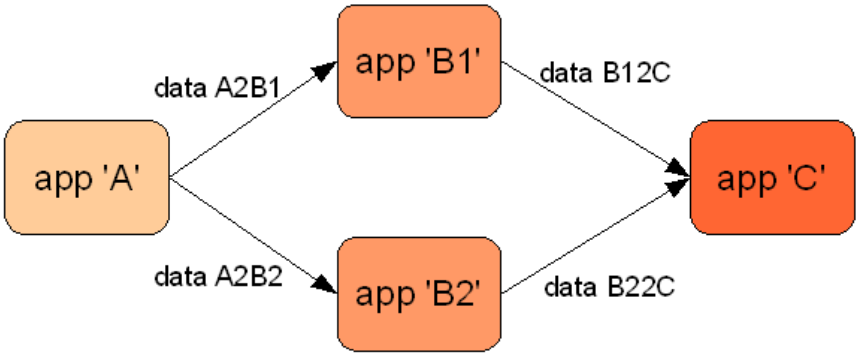


Figure 4-5 A Simple Network of Functions

This diagram is similar to a critical path analysis where each element is unable to execute until its data input is available. Applications B1 and B2 represent a duplicated function where B represents the function, and the numeral indicates the channel number.

From here, relevant information for the system blueprint can be captured. Considering application 'B1', the following can be captured:

Element	Value	Explanation
Application Name:	app B1	
Application Reference #:	#	Assigned automatically, but will be a unique identifier.
Assigned to:		Value defined by configuration algorithms
Inputs from:	From: app A; Data name: data A2B1	A table is shown here in case more than one input is expected
Outputs to:	To: app C; Data name: data B12C	
Assignment criteria:	!B2	It is not desirable to locate the redundant process on the same physical resource
Criticality:		Value only entered if this is the last process in the functional tree

Run time:	<us>	This time is taken from experimentation. A more ideal measure is making an assessment on the amount of computing resource required, but a time in micro seconds is generally accurate enough
Start time:		Value is defined by configuration process

Table 4-3 Example Capture of Application Information

This textual description is stored with the application such that on start up, the global systems manager can assimilate this information.

4.4. Configuration/Re-configuration

The purpose of the configuration algorithms is to derive an appropriate allocation of applications to computing resources. This assignment should take into consideration:

- Available processing time on each module
- Hardware requirements of the applications
- The requirements for real time, deterministic execution of distributed tasks

The goal is to achieve automatically an appropriate distribution of functions that would mimic a manual design of a distributed real time control system.

Functional distribution throughout a standard non-reconfigurable distributed control system is normally done by deriving the relative time each activity can function without interference within each time frame. A time frame (TF) is a regular division of time in which activities are scheduled to occur, and then repeated for as long as necessary. An example of how functions might be scheduled to run across the distributed system is shown in Figure 4-6.

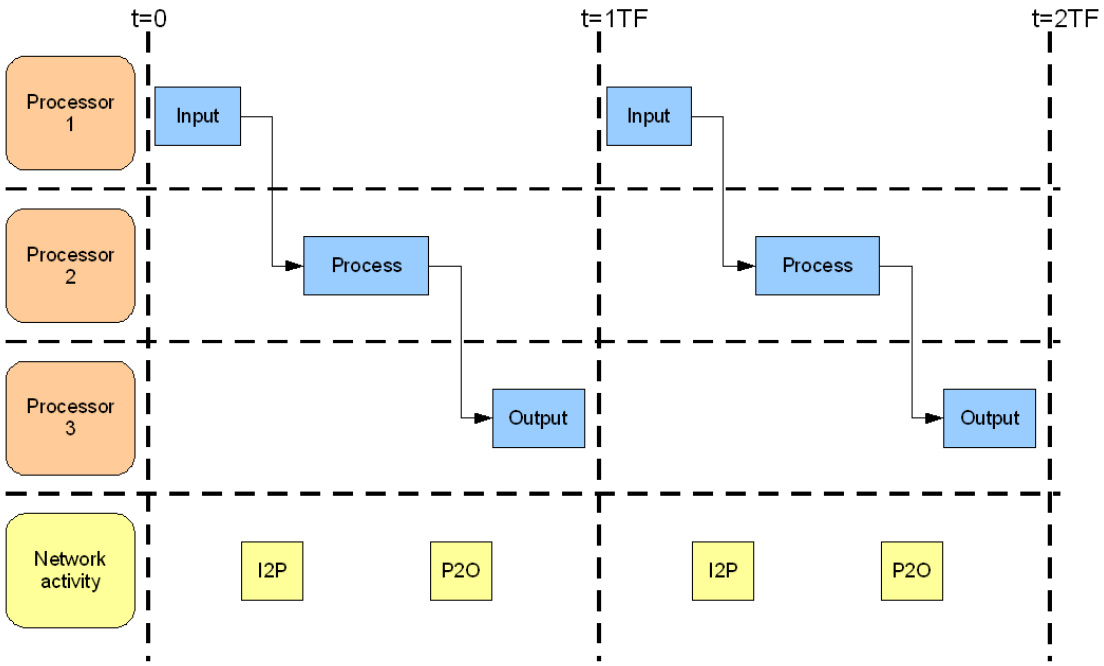


Figure 4-6 Organising a Real Time Distributed Control System

Figure 4-6 shows how each functional element is assigned not only to a resource, but also a slot of time in which to execute. This is true also for communications where each data packet has to be scheduled on the network such that collisions are entirely avoided.

This method of arrangement is a classic technique when arranging distributed real time systems and as such forms the basis of the design of here. It is essential to maintain the concurrency of events to uphold the integrity of the real time control. The designer of the system can measure or approximate the time required for each of the tasks or communications to take place such that these can be incorporated into the design of the control problem.

Previous examples of configuring an IMA have used a multi-static approach where a number of configurations are designed, stored in the systems blueprint and reverted to as a response to either a change in functional need or a fault. In this project, the configuration will be derived dynamically. The algorithms are executed and an assignment set is derived based on current information regarding the state of the system. Should the

system require a different functional set, or require response to a fault, the configuration algorithms will be repeated using current system health inputs as assignment restrictions.

The configuration methodology adopted is based on that highlighted by Yann-Heng (Yann-Hang, Daeyoung, Younis, Zhou, & McElroy, 2000) but largely focuses on the temporal partitioning problem. Yann-Heng highlights the importance of spatial partitioning to prevent the propagation or errors between operating applications. Further to Yann-Heng's work, this investigation looks into the additional problem of functional networks where a single application may have many inputs derived from an interacting web of functions before it can perform its task. The configuration is then stored in the system blueprint and communicated to the local systems managers.

Lee (Lee, Kim, Younis, & Zhou, 2000) furthers previous work by discussing how the temporal and spatial partitioning drives the configuration of the system. Each task and communication must be arranged such that they do not interfere with neighbouring functions. Lee demonstrates a system where this is performed in a static manner. In this investigation similar principles are studied but with the aim to produce a method of autonomous configuration and a resultant robust schedule as part of the system design.

The resource allocation problem is based on the CPRTA (Constraint Pro-gramming for solving Real-Time Allocation) method investigated by Hladik (Hladik, Cambazard, Daplanche, & Jussien, 2008). The progression from this work is that in this case the algorithms are executed at run time, and will be employed to facilitate reconfiguration during the system operation. For this to occur, the appropriate systems descriptions normally chosen in a static sense will have to be derived automatically in order to bound the configuration algorithm. The methods employed here to facilitate this will be described at a later stage.

A number of plausible methods exist where a configuration can be derived, such as the use of neural networks or Bayesian belief networks. For this project, a recursive placement algorithm method is used where applications are placed where they do not contradict

assignment criteria provided in the application information (Table 4-3 Example Capture of Application Information)

The configuration process implemented is explained using a simple example in section 4.4.1.

4.4.1. Configuration Example

This example will describe the process the system follows in order to assign the network of functions described in Figure 4-5 to a set of standard modular processing resources, shown in Figure 4-7.

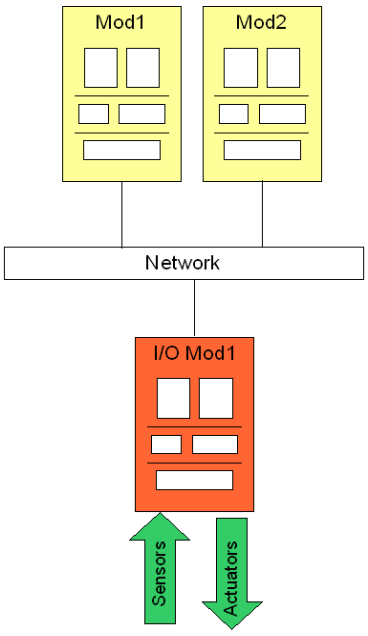


Figure 4-7 Simple Network of IMA Resources

The functions shown in Figure 4-5 will have interdependency as they are all part of a process flow. A complete system is expected to have any number of functional networks operating alongside each other. The configuration algorithm developed places each functional set in turn to ensure concurrency of the process flow is maintained. As this is an initial system configuration, if at any point no solution to the configuration problem can be found then essentially the system fails to boot and informs the design team or system maintainers that there is a mis-match in available processing and resources to required application functionality.

The top level process that is followed is shown in the flow chart in Figure 4-8. For further detail regarding the algorithm, a pseudo code version describing the whole configuration algorithm is included in Appendix D.

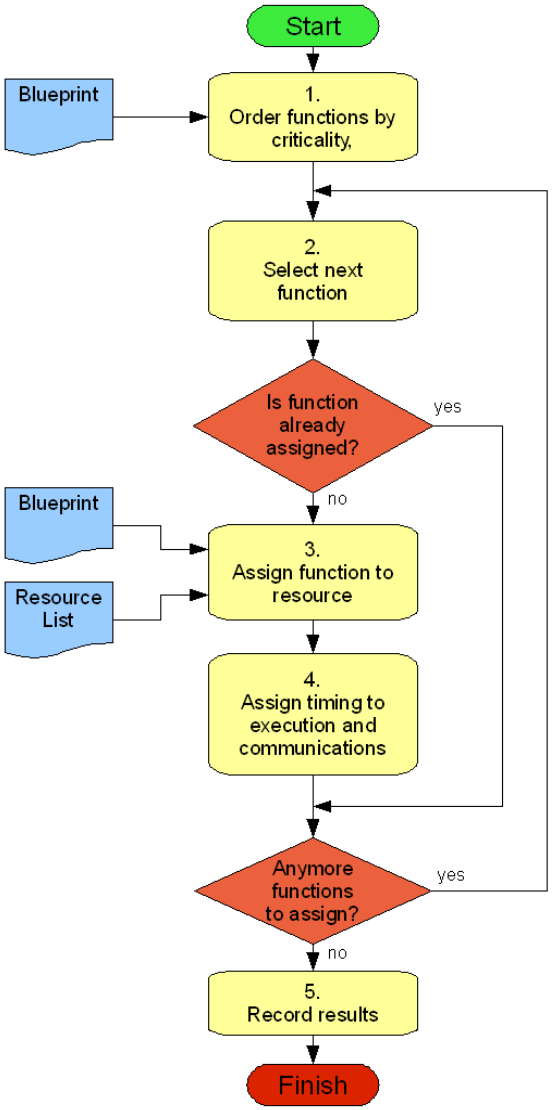


Figure 4-8 Recursive Placement Algorithm Process Flow

The general philosophy behind the process is that the most critical function is identified and placed first. The algorithm then attempts to place all necessary pre-requisite functions (i.e. all those preceding it in its functional tree) to this critical function, adjusting execution orders and timings along the way. Upon completion of the placement of the functional set, the allocation and timings are then fixed and are not allowed to be adjusted by later allocations. The process is repeated for subsequent functional sets, placing the remaining applications around those already assigned. This process is repeated until all functions are

placed. If at any stage, a process cannot be completed, an error is returned and the configuration fails.

The individual processes (numbered 1 to 5 in Figure 4-8) are explained in the following relevant sections.

4.4.1.1. Process 1: Order Functions by Criticality (and Identify Resources)

The level of criticality of the functions in this case is a simple solution where a function is declared 'time critical', by entering 'TC' in the appropriate field when defining the application specification. Although all real-time functions have deadlines to meet, data related to a safety critical items such as a flight control function for example, should take priority over display data or less critical control activities. In more complex networks of functions, criticality will be required to be graded and the process to be described can be modified in order to accommodate this.

The process for sorting the functions into criticality is actually quite a simple one, and summarised in Figure 4-9.

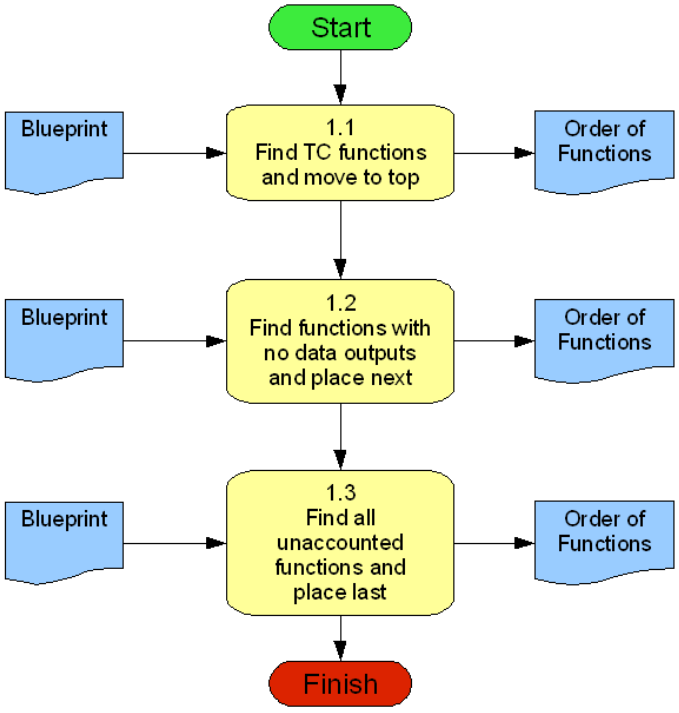


Figure 4-9 Order Functions by Criticality

Figure 4-9 contains a data store named 'Order of Functions'. This is an internal variable designed to store references to the functions in order of criticality.

Aside from functions marked 'time critical', it is also considered that the next most important functions are in fact those that perform the actual output required of the system. These functions normally have an output to an actuator or to a display screen and therefore will have more specific allocation requirements in the IMA hardware. These functions are considered more important in order to be placed first. These terminal functions are often easily identifiable by analysis of the functional specifications as they have no communication data to be output on the network alongside their physical interactions with the environment. Process 1.2 "find function with no data output" in Figure 4-9 above refers to this search. This process may not provide an optimal solution, bus does enable a configuration to be realised that satisfies timing requirements.

4.4.1.2. Process 2: Select Next Function

Selecting the next function to place requires more thought and care than is first envisaged. This is brought about by the desire to place functions by following the process flow along the functional network. As mentioned earlier, the general procedure is to start by placing the last function in the set and working backwards. Where multiple inputs into a function occur, each branch has to be pursued in turn in order to be sure that all pre-requisite functions are placed. The process followed is similar to a blind search algorithm where the first input is followed each time until an initiating function (one with no data inputs) is found. The search resumes by selecting the next input at the lowest level branch until all avenues have been explored.

The process is described in Figure 4-10. A key feature of this process is the "breadcrumb array" - an internal variable that keeps track of the path taken through the functions such that it can be retraced when the current avenue has been fully explored.

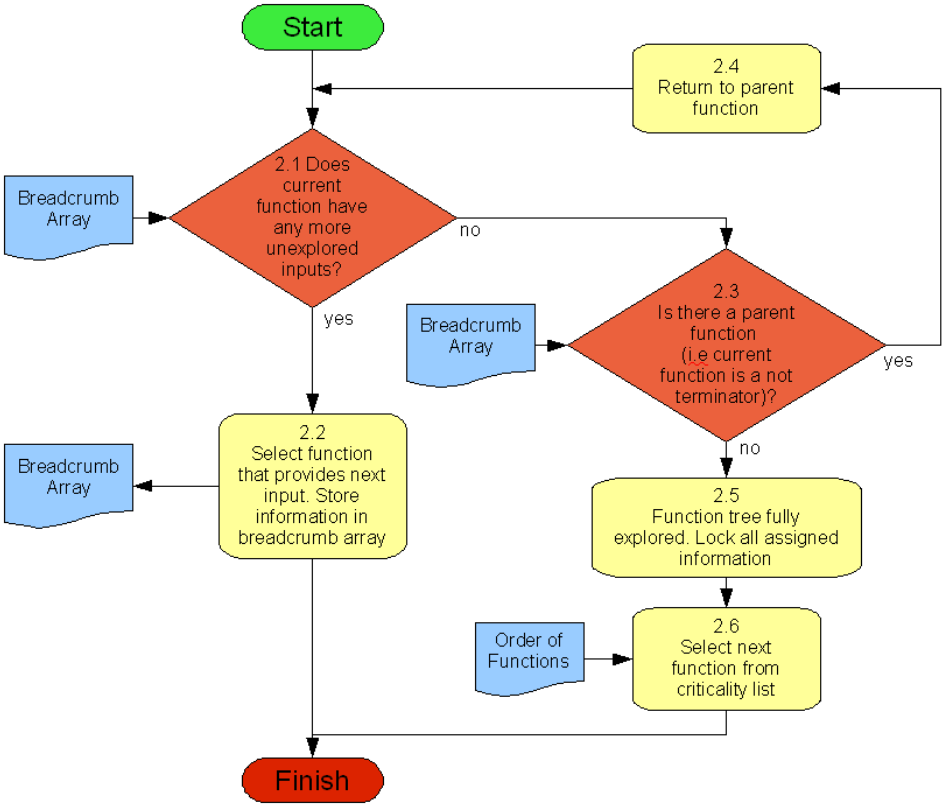


Figure 4-10 Find Next Function to Place

When a functional tree is complete, defined by the decision block 2.3 in Figure 4-10, the attributes relating the assignment of these functions is fixed. Other functional sets have to be assigned around these functions in order to ensure that the timing specification derived is not compromised by later assignments. When this occurs, the algorithm looks for the next most critical function calculated earlier and described in section 4.4.1.1.

4.4.1.3. Process 3: Placing Function on Resources

Placing a function on a resource has to be performed carefully in order to make sure that the assignment criteria are satisfied. The general process is that the algorithm attempts to place the function on each resource in turn, moving on only if a contradiction is found in the assignment criteria. The process is described in the flow diagram of Figure 4-11.

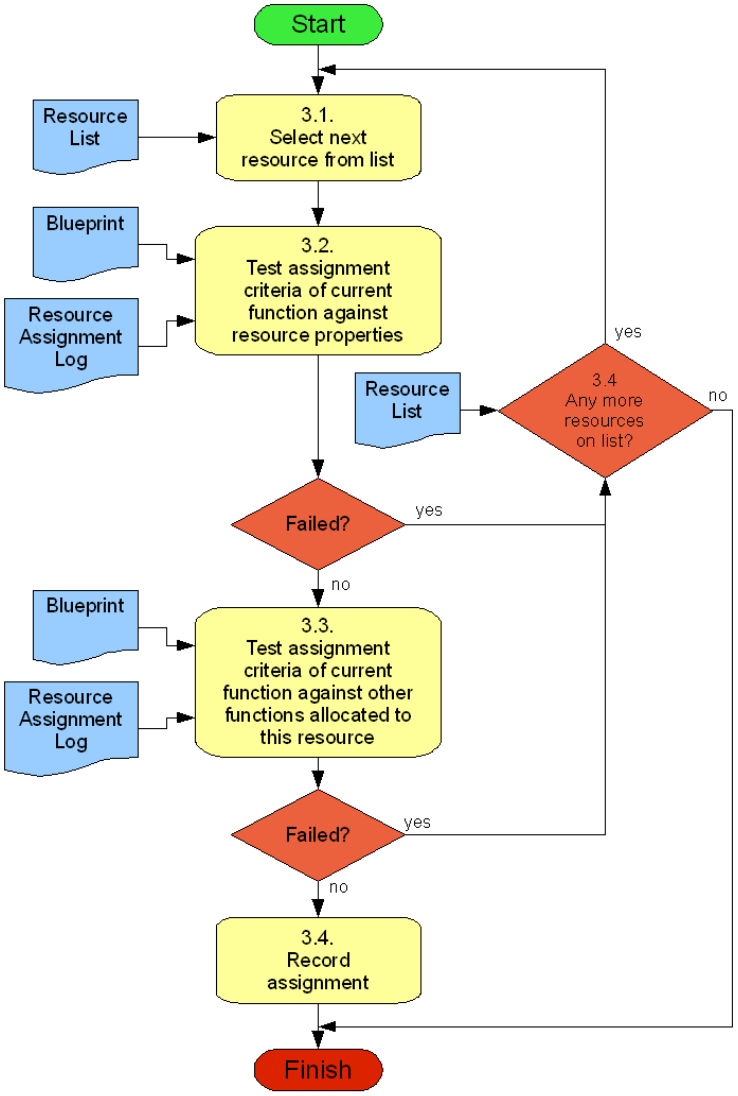


Figure 4-11 Assign Function to Resource

Generally, functions have the requirement to be assigned to a specific type of hardware, such as an input/output module, or require to be segregated from redundant processing. Further requirements come from the desire to not overload a single resource with too many functions.

Should no resource be found that does not contradict the assignment criteria, the entire configuration process fails with an error output.

4.4.1.4. Process 4: Schedule of Application Execution and Communications

A critical aspect of managing the configuration is ensuring that the resultant arrangement is a concurrent process that satisfies the functional flow described in design, such as outlined in Figure 4-5. It is important that the process takes into account the arrangement on each module such that they execute in the correct order, but considers a system wide perspective to take into account pre-requisite functions running on another resources. The process therefore follows two main functions, correcting the order of functions placed upon the current module, and the correction of timing of each of these functions. Both of these processes are executed each time a new function is placed on a module.

Correcting the execution order upon the module

This part of the process is the implementation of a sorting algorithm in order to ensure that pre-requisite functions are scheduled to execute on the module before later functions. The sorting process is described in Figure 4-12.

The resource assignment log, mentioned in the previous section, not only keeps track of the assignment of functions to resources, but also records the order and timing of each of the functions on each resource. The information is very similar to that recorded in the system blueprint but with a view taken from the resources as opposed to the functions.

The process begins by placing the newest function of interest at the front of the module. The algorithm then checks if any application scheduled to execute later on the module is

actually a pre-requisite of a function scheduled to run sooner. Clearly, this arrangement would cause concurrency problems in the overall execution of the functionality.

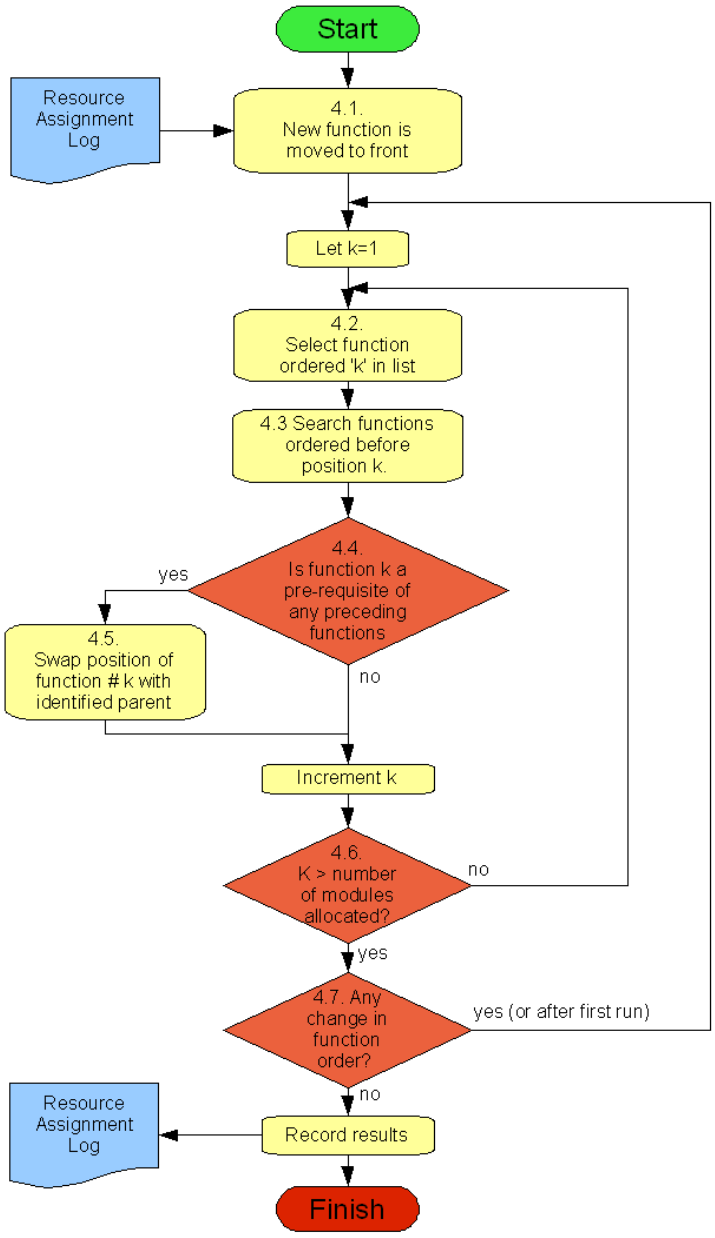


Figure 4-12 Order Function on Module

Processes 4.4. and 4.5 describe the identification of an out of order function, and the process of swapping them around. A prerequisite function is identified as being out of order if it realised that its execution is scheduled to complete after the scheduled start of a

subsequent function. Process 4.4 also involves a check to confirm that the functions being compared do exist within the same network of functions. This check could be done by another 'blind man's search' of the data interactions leading up to this module. This would provide a more accurate picture of the relationship but be more timely and complex. However, in this instance a more simple check is made in the following way:

- Is this function part of the same functional tree?
- Is this function at a lower respective level in the tree according to the breadcrumb array?

If both of these tests are true, then the function is considered a prerequisite. Although not entirely accurate and represents an area for improvement, this was found to be an appropriate solution.

If a newly added function is not affected by this sorting algorithm because it is not recognised as part of the functional set, the next stage in the process arranges it into an appropriate time slot in order to allow access to processing time upon the module. This is described in the following section.

Assign Timing to Execution and Communications

A critical part in the configuration process is the accurate assignment of timing to the execution of the functions and the communications. Without this, the system will not be capable of correctly performing distributed real time control algorithms. It is vital that functions are scheduled to run concurrently and that communications are arranged to support this need and timed to avoid collisions on the databus.

One way of considering the timing problem is working out the earliest possible time available for the function to execute. This is dependant on the latest time of two considerations:

- The time the last input is due to be received by the function from its immediate prerequisites.
- The earliest available block of processing time for executing the function on the module.

A similar pair of considerations applies to the communications structure. The earliest data transmission time will be the latest time taken from the following:

- The time the function completes from which the data originates.
- The earliest available block of time on the network for transmitting the data packet.

Both these sets of constraints infer that the pre-requisite function has been placed. It can be seen that this is slightly contradictory to the process outlined in Figure 4-8, as this describes a process that works backward along the functional tree. The timing adjustment is actually triggered as the algorithm retraces its steps back up the functional tree. This way, as each pre-requisite is placed, the timing of the subsequent function and the associated communications are re-evaluated, as shown in Figure 4-13.

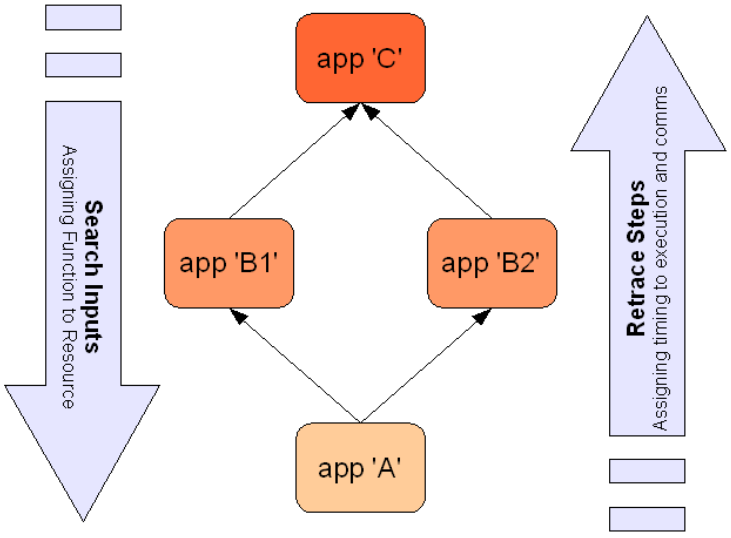


Figure 4-13 Relationship between search direction and function

The overall process for assigning a time is as shown in Figure 4-14.

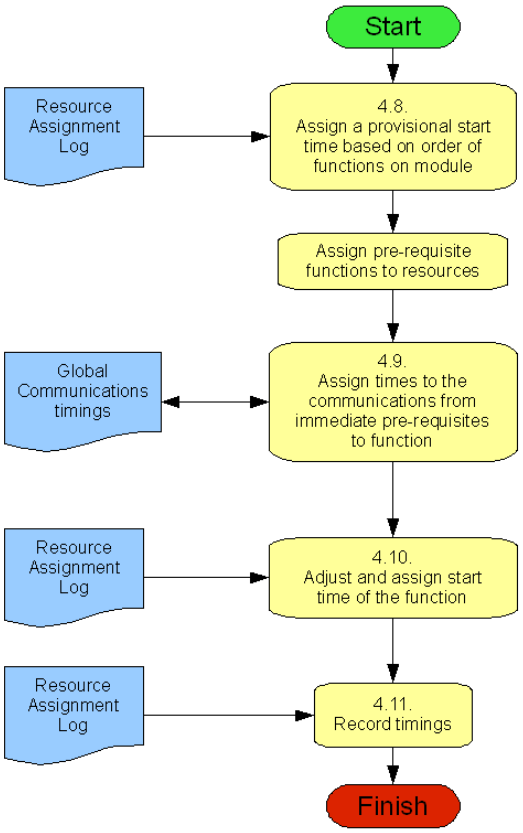


Figure 4-14 Process to Assign Timings to Communications and Execution

Process 4.8 assigns a provisional 'earliest possible' execution time for the function assigned. This is based on the order in the execution queue that it has been given from the process outlined previously. This algorithm derives from the assignment information already stored the time at which the current function can execute. This will be equal to the end time of the preceding function.

After the pre-requisite functions have also been assigned, it is now possible to schedule the communications. This is done by searching an internal variable storing the global communications timings. This variable is an array that stores all planned traffic travelling along the network. By analysing this array, starting from the time that the preceding function is due to end, a gap can be found that is large enough to fit the data transfer in. This new data packet information is then added to the global timings array.

At this stage, the earliest time that the function of interest can be executed can be now found by adjusting the start time of the function to occur after the end of the planned data arrival. This information is then recorded.

4.4.1.5. Process 5: Record all Details in Blueprint

The final part of the process is to write all values from the important internal variables to the system blueprint. At the end of the configuration process, there are two internal variables that contain the information required. These are:

- Resource Assignment log
- Global Communications Timing

The Resource Assignment log contains all the schedule and timing information for the functions, with a view taken from the resources. This is converted into the blueprint format which takes its perspective from a functions point of view. The global communications timing variable array contains the information for all the data packets upon the network. This is now interrogated and the appropriate timing information is written to the input/output data pack information stored in the blueprint.

This information is then communicated to the entire system ready to be implemented.

4.4.2. Start Up Procedures

Upon powering up, the system has a number of tasks that require performing, namely:

- Assignment of management roles (global systems manager, local systems managers, etc)
- Identification of hardware elements and capabilities
- Identification of applications to be assigned and requirements
- Assign an initial configuration and begin functional tasks

Before the system is powered up, the hard drives of each processing module are installed with software that comprises the IMA middleware and the associated applications. This technique is implemented as it avoids any transfer of programming code when

applications are re-assigned. Instead applications can be activated or de-activated depending on assignment.

The first module powered up is assigned the responsibility of performing global systems management duties. As the processing module is already loaded with all the applications that could be implemented systems wide, it can already begin to populate the blueprint with information. The global management module then begins to wait for other modules to make themselves known.

The connection process is designed as an opportunity for each module to introduce itself to the systems manager. Not only does it register itself as an available processing module on the network, it makes the global systems manager aware of its hardware capabilities, i.e. Data input/output. The capability of each module is currently done by defining the name of the module upon installing software, but could be done via an automatic self check on system start up.

The global systems manager uses this information to populate resource information for the configuration algorithm to interrogate, as described in section 4.4.1. It will then wait for more modules to introduce themselves, or until the user issues the command to initiate configuration. Upon receiving this command, it will attempt to assign the functions installed on the system to the resources that have appeared on the network by initiating the configuration process.

4.4.3. Using this Algorithm for Reconfiguration

Reconfiguration is employed in this system as a method of handling and tolerating faults that occur in the IMA. As such, the system will undergo a reconfiguration when it identifies an appropriate event that can be solved by reconfiguration. The system will attempt a reconfiguration when it is recommended by this systems manager to do so.

A new configuration can be generated by repeating the above process, but placing further restrictions on assignments to avoid re-using faulty components. For example, if a module

fails, it is removed from the available resource list before the configuration process is run. This way the resultant configuration will not have assigned any functions to this resource.

The reconfiguration process during run-time raises key safety concerns. Developing a new configuration will take time and is unlikely to completed within a single time-frame. At the instance of failure, the IMA must have fault tolerant mechanisms that do not rely on reconfiguration to continue to provide service. The solution to this problem is in using traditional multiplexing techniques and rules such that the placement of applications on modules will not result in the loss of all redundant processing channels should a single module or a single application fail. Other inherent design characteristics of IMA, particularly good partitioning and fault containments, will prevent a single failure propagating throughout the system.

Implementing a new configuration is also an identified area of high operational risk as it requires current processes to be stopped and new processes to be started. In terms of a safety case, the failure of this process is a critical problem that requires to be addressed.

The particular problem of stopping a set process and starting a new one cannot be removed from this system as it would prevent reconfiguration being an option in event of failure. However, the risk of failure has been reduced by taking certain steps within the design and implementation of the IMA. The first step is in the installation of applications. In this architecture, all applications are installed on all modules ready to be executed. Implementing a new configuration is performed by instructing which applications to execute and which to not. This removes any complications and failure modes that could be introduced by physically installing applications during run-time.

This also means that implementing a new configuration can be performed at the end of a time frame and be ready to go as the new frame starts by implementing two substitutions of data on each module following the communication of the new configuration from the Master unit. The first is a variable within the local systems manager that instructs which application to execute on this module. The second is the array mapping of data outputs to

physical network addresses within the communications manager within each module, and changing the times within the frame that these are expected to happen.

The resultant levels of redundancy restored will be dependent on the available remaining processing modules. It is possible at this stage to prioritise critical processing channels at the expense of those less critical in order to restore higher levels of redundancy.

The reconfiguration process suggested here will mean that, although reconfiguration cannot be achieved within a single time frame of a few milliseconds, the system will only be operating at a lower level of redundancy for the few time frames it takes to restore critical channels.

4.4.4. Configuration Summary

The configuration presented here provides a rigorous process for assigning simple networks of functions to resources based on a logical assignment method that ensures the applications are not placed in contradiction to their operating requirements.

There are a number of ways in which this process could be improved to maximise the resources available. An example of this would be the inclusion of an optimisation method for the execution timings whereby the most critical functions could be adjusted, as long as they still met their delivery time. This may allow less critical functions to run more efficiently, or even enable a configuration to be realised at all.

An example of how optimisation can be beneficial is shown in Figure 4-15. The top part of the diagram shows the result of a basic configuration of a process stream, with functions for input, process and output. This is executed alongside a redundant set of functions with lower criticality named Input 2, Process 2, Output 2. By following the above configuration process, a configuration that does not contradict placement requirements is obtained, but the solution is non-optimal. It can be seen after manual optimisation that the execution deadline of both streams can be arranged to meet a hypothetical deadline, as opposed to just the primary stream.

Although not covered in this research, it would be possible to study and create an adjustment method to take advantage of opportunities for overall system improvement at the expense of some specific performance criteria.

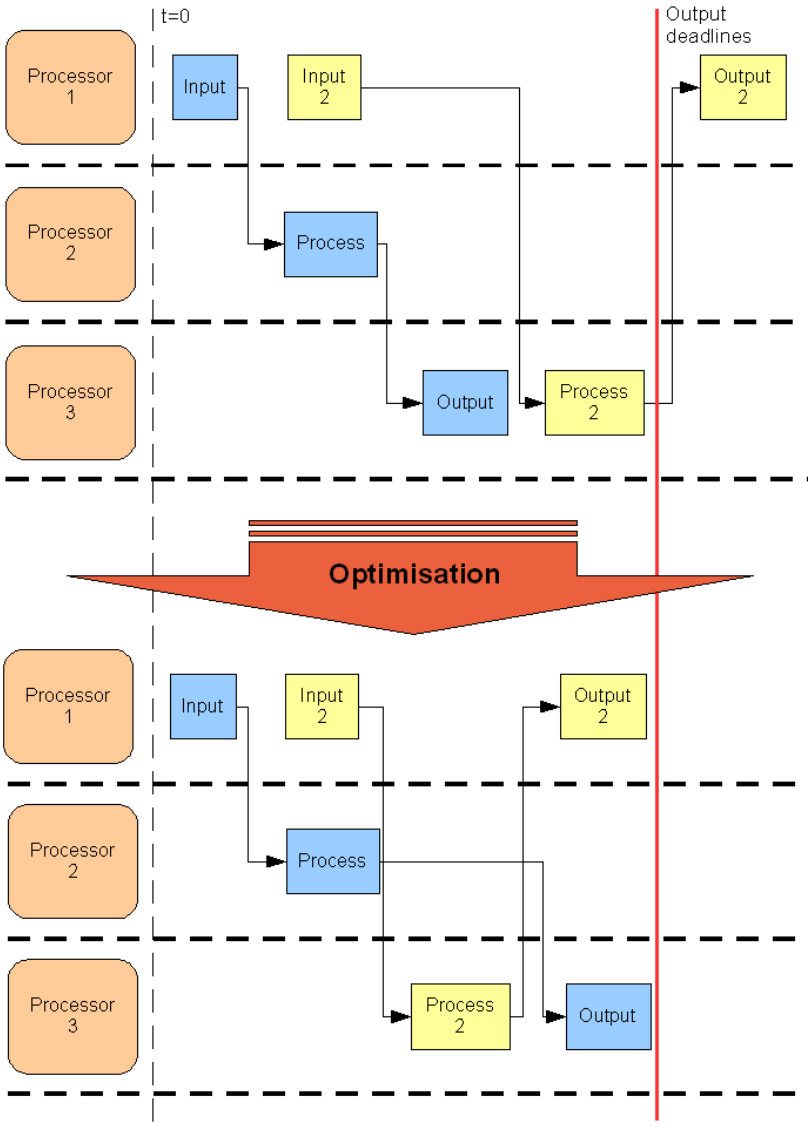


Figure 4-15 Benefits of Optimisation

4.5. Synchronisation and Communications

The ultimate goal of the implementation of the IMA it to provide a flexible architecture that meets the requirements of a real time controller. Well-founded methods and procedures exist for so called traditional distributed control systems in which the system design is static. These methods are designed for systems whose hardware and software

components are largely bespoke and are not intended to be changed throughout the system's lifecycle. It is the designer's responsibility to arrange the communication timings and synchronisation of modules in order to maintain a concurrent real time process. In a flexible system where the physical location of software components and the time within the frame at which these might execute is all changeable, the synchronisation and communication will have to be arranged and checked autonomously. Most of this issue has been dealt with in the design of the reconfiguration algorithm but it remains to be defined how the execution of applications and the timings of communications are performed such that these timing needs are met.

The following sections will review the critical aspects of distributed real time control systems and comment on how these issues are addressed in this architecture.

4.5.1. Synchronisation of Modules

A fundamental issue of running separate processing modules is the synchronisation of the processing clock on each hardware module. To ensure concurrency, it is required that each time frame within each module is begun at the same instant in time. If this is not the case, timeout faults or data packet collisions will occur as variables will not be available to be transmitted in their globally allotted time frame.

Although highly complex clock synchronisation techniques exist, in this application it is possible to take a higher-level solution to the problem. The key requirement here is to ensure that each module begins its time frame at the same instant, allowing for a small margin of error that would result in an acceptable level of 'jitter' in terms of the resultant sample time.

Rather than adjusting the underlying processor clock, the solution undertaken here is to use a single data packet as a global synchronisation message to denote the start of a time frame. The master module in the system is responsible for the timing and distribution of this message. The other modules must have completed any outstanding activity and be awaiting this message, ready to begin a new time frame.

This method is not without its limitations. Firstly, there is the issue of accuracy of timing. This method largely relies on a swift, uninterrupted message to be sent to all modules on the network synchronously. Due to the latency of transmission and any intermediary software delays, this will never truly be the case. Secondly, there is the introduction of a single point failure. If the message fails to send for whatever reason, for example a master module failure or a software blip that means this message is missed, then the whole time frame is jeopardised.

Despite these shortcomings, experimentation with this method has shown it to have very reasonable reliability and accuracy. Although not suitable for high-end applications, it has proved to be effective enough for the purposes of this investigation.

4.5.2. Synchronisation of Applications

The configuration algorithm assigns the physical location and start time of each application. It is now important to ensure that these applications are run at the correct moment in order to meet these timing criteria.

The execution timing is achieved in this setting by making each application wait until its prerequisite data has arrived. This method guarantees that the most recent data is being used in processing and not an out of date value. By not having a separate method for controlling the timings for applications and allowing the communications to control the process flow, conflict of timings between the two and the need for extra synchronisation is removed. Late or missing data can be handled by including appropriate ‘time-outs’ in any functions that are waiting for data, with the generation of appropriate error messages.

4.5.3. Synchronisation of Communications

As with the application synchronisation, the timing and arrangement of the communication to satisfy the concurrency of the functional flow has been defined in the reconfiguration algorithm. This section describes the mechanism involved to ensure that these timings are achieved.

As part of the configuration process, the communication structure is defined. Details of when each data packet can be sent is recorded as part of the overall blueprint of the system – effectively stored as an attribute of each application in terms of the input 'Rx' time and the output 'Tx' time. When a new blueprint is distributed, each module can search the attributes mentioned to identify when it is required to perform a data transfer. This information is used to control the communication timings of each module from the application layer of the 3-layer stack.

As mentioned previously, the communications timing controls the execution of the applications. For incoming data packets, the software will wait until its allotted time slot and then prepare to receive data. When a data packet is received, this triggers the execution of the application. The output data of this function is stored until the allotted time frame to transmit is reached.

4.5.3.1. Collision Avoidance

Ethernet is a serial communication method based on a collision detection mechanism. The hardware is designed to identify when two nodes on the network are attempting to send a packet, and to wait a random amount of time before attempting a resend. In order to achieve as close to real time control as possible a collision avoidance technique must be used, as is the case with the MIL-STD-1553 or ARINC 629 bus topologies.

Collision avoidance is normally done with low level programming of the communication hardware. Specific time slots for each data packet are designed off-line and programmed into the system. This provides a rigorous solution and optimises the use of the databus allowing high levels of data transfer. However, Ethernet cards are widely commercially available and very low cost in comparison to other methods. This makes them an attractive solution if they can be used in a collision avoidance manner.

The way the solution was achieved in this example is to control the timing of data transfer in a higher level program. By using the data packet timings from the blueprint, each node

can be prepared to send and receive at the appropriate time, ensuring collisions are avoided. However, Ethernet cards are not designed to be used in this way and as a result introduce extra delays in the system. They are relatively slow to begin a communication as each card will always perform a check to see if the databus is clear before attempting to transmit. This introduces a delay of around 1ms to each data packet transfer. As a result of this, the amount of data transferred along the databus is not restricted by the quoted bandwidth of the system but by the number of data transfers that are required to be performed within a set time frame. The resultant system, albeit appropriate for this example, actually uses a fraction of the available data bandwidth normally available.

4.5.3.2. Managing Time Critical and Non-Time Critical Data

Data between applications is not the only information to be sent over the network. Already it has been mentioned that a synchronisation signal is being transmitted, as well as systems management data used to communicate between the command aspects of the system. It is important to manage the so-called overhead data around time-critical data to avoid any interference to the real time application.

The time frame in which the applications are scheduled to run concurrently is split into 3 sequential parts. The first sub-frame allows for the transmission of the ‘drumbeat’ or timing-pulse signal from the master module to all others on the network. The second sub-frame allows enough time for all the data between applications to be passed and the third allows for the systems management communications to occur. This division of time is shown below in Figure 4-16. Here, 3 modular processors are running a simple concurrent series of functions, with Processor 2 designated as the master module.

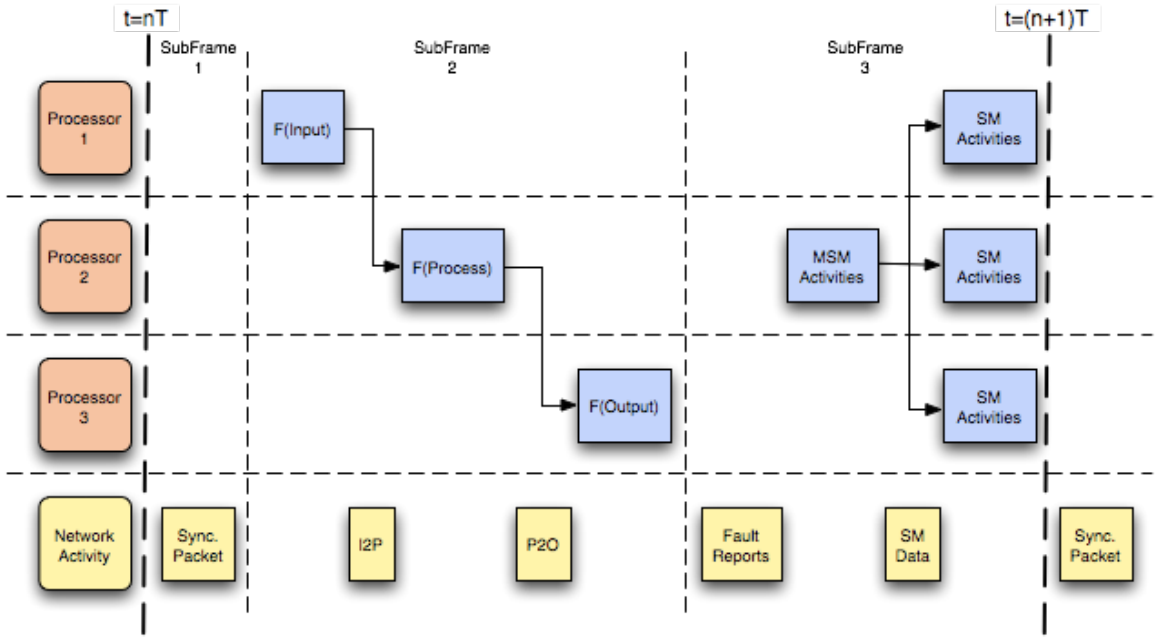


Figure 4-16 Subdivision of Time Frame

In this system, this order is deliberately chosen with the intention that as much overhead processing and data transfer as possible is saved until after the real time functions have been completed. As well as helping these real time events to stay concurrent and repeatable, it also allows the delay from the acquisition of an input to the output of a command signal to be reduced to a minimum, which is particularly beneficial in terms of control system performance.

4.6. Error Recovery

The focus of this study is in the use of dynamic reconfiguration as a method of responding to the occurrence of a fault, in order to restore higher levels of redundancy and capability.

The most pertinent errors to observe in this study will be those that are associated with the IMA operation itself, as opposed to errors experienced in the target platform. Although it is theoretically possible to use a variety of fault detection methods installed as applications to identify faults within the Maglev system, it is more useful to study those that will affect the processing or communications within the IMA.

For the purposes of this investigation, case studies regarding specific failure modes will be studied. Two specific cases will be taken forward, namely:

- Failure of a processing module
- Failure of an application to execute

These failures have been chosen as they capture two levels of systems failure. The failure of an application is likely to interrupt a single processing stream whereas the failure of a module could affect many, dependant on the allocation of tasks.

The precise modes of these failures will be discussed in the following section. Each failure mode will be defined along with the expected effects of the failure.

4.6.1. Processing module failure

Failure Mode: In this case the module is simulated to have failed to a silent state, as if power has been cut to it completely, or the system has 'hung'.

Effects: Any application executing on this module will no longer respond, including any system management activities. The module will not output any random communication to consume bandwidth unexpectedly.

Failure Detection: In this case, as the failure is simulated, an automated error message is also generated to be sent to the systems manager. Although failure detection methods are possible in this circumstance they are not generated here. This message will inform the system manager of the specific module that has failed and initiate the reconfiguration process.

Expected System Reaction: At the instance of the fault the system will maintain service provision. This will be possible as the allocation of any redundant processing channels for this service will be arranged such that a single module failure will not prevent these executing. The IMA will operate at a lower level of redundancy until reconfiguration has

occurred. The system will reconfigure to the highest possible capability using new information about available resources.

4.6.2. Application Failure

Failure Mode: In this case application is simulated to have failed to a silent state. This is as if the application has 'hung' and is no longer executing.

Effects: The assumption here is that correct partitioning on this module will prevent other applications that are sharing the resource from failing. The effects seen will be the absence of any data results from the application. Subsequent applications further along the processing channel will receive 'null' data packets.

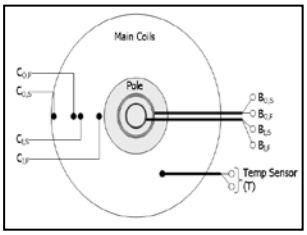
Failure Detection: As the failure is simulated, an automated error message is also generated to be sent to the systems manager. Although failure detection methods such as timeouts and data checking techniques are possible they are not generated here. This message will inform the system manager of the specific application that has failed and initiate the reconfiguration process.

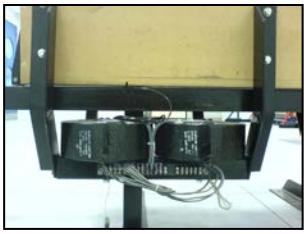
Expected System Reaction: At the time of the fault, the application will stop sending data results for communication. Service provision is expected to be maintained as the initial design of a duplex channel will mean there are further channels replicating this service. Appropriate data checking methods will allow the selection of the remaining true data whilst ignoring any null data. The system will the reconfigure and attempt to place this particular application on a different processing module in an attempt to rectify the failure.

4.6.3. Summary

Both the failure modes described above can be tested by raising an error code manually to simulate the occurrence of the fault. This will then trigger a reconfiguration request with any new configuration restrictions included in the call. Suitable logging of the events will

occur such that a view of the reconfiguration process can be reported. The results of these tests are reported in Chapter 7.





CHAPTER 5:

Configuration and Real-Time Robustness Testing

5. Configuration and Real-Time Robustness Testing

The purpose of this chapter is to test the appropriateness of the configuration algorithm described in Chapter 5. The algorithm will be given a series of functional networks and processing resources to assign them to. In the following chapter, a series of failures will be introduced to the system to attempt to observe the successful re-assignment of functions such that as much service as possible can be maintained, and graceful degradation is achieved.

In addition to these tests, a simple functional network will be implemented on the IMA test rig and used to assess the performance of the communications between applications.

5.1. Validation of the Configuration Algorithm

This section contains the results of a series of functional allocation problems. The purpose of these tests is to demonstrate that the configuration algorithm developed will allocate networks of functions to available resources. The applications should be placed such that service will be maintained (where applicable) at the instance of:

- A complete processing module failure
- A failure of an application

The configuration algorithm will be presented with increasingly complex functional networks (i.e. duplex and triplex arrangements) and should place these networks whilst maintaining the independent processing channels in the event of a single failure. At this stage, reconfiguration is not being assessed, only the ability to sensibly configure simple application structures. This will demonstrate that the system arranges applications in order to tolerate faults at the instant of occurrence via a traditional redundancy-based method. The resultant algorithms are generated only to test the configuration algorithm and are not implemented nor execute on the developed IMA network. This allows a freedom to experiment with different hypothetical configurations.

A hypothetical real-time processing task is defined for the purpose of this benchmark test. It is assumed that data is to be collected from a sensing element, processed during each sample, with the result informing some actuation activity. This scenario can be extended to include duplex or triplex processing channels, multiple sources for sensing or increased redundancy in actuation. Escalating the levels of redundancy within the system complicates the assignment process for the configuration algorithm.

The following sections show the result of the configuration algorithm for a number of application networks based on the above hypothetical description. Each section will include:

- a brief description of the necessary system design in terms of hardware requirements
- A description of the assignment criteria for the functions
- A diagram showing the allocation of function

In order to restrict the testing to the configuration methodology, any systems management processes and communications are ignored. It is also assumed that each application takes 2 milliseconds seconds to execute and each communication packet is allowed 1 millisecond to transmit and receive.

5.1.1. Single Sensor, Single Process and Single Actuator

The first functional set of applications to assign is a simple 'sense, process and actuate' structure as defined in Figure 5-1. The rectangular blocks represent the application and the labelled arrows represent the required data communication.

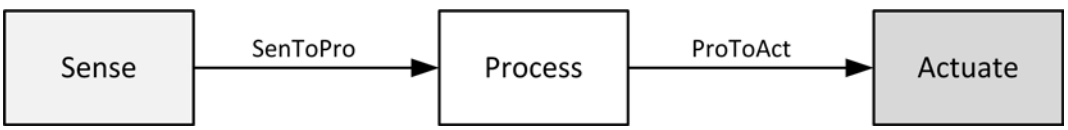


Figure 5-1 Single Sense, Process and Actuation

As this represents a real time process, the three applications in Figure 5-1 must occur concurrently and within the same processing interval (or time frame). Ideally, there should be a minimum amount of time between the 'Sense' and 'Actuate' function.

In order for a configuration to be realised, two processing modules are required:

- An input/output (I/O) module (Named '*IOMOD*') – this module has physical links to the required sensing and/or actuation hardware
- A standard processing module (Named '*Module 1*')

It could be argued that in this case, there need only be a single module equipped with sensors and actuators that can also house the processing task, but the arrangement in Figure 5-1 is chosen as it is more readily expanded in later scenarios.

Each application can be assigned configuration criteria to ensure it is located on a module with the correct capability. These are defined in Table 5-1, and input into the application definition as defined in Chapter 5.

Application	Criteria	Description
Sensor	*IOMod*	Placement on the I/O module
Process	*Mod**	Placement on any generic processing module
Actuate	*IOMod*	Placement on the I/O module

Table 5-1 Assignment Criteria

The communication requirements and assignment requirements are input to the application definitions (section 5.3.1) and information regarding the available modules is manually input (as opposed to having the information automatically generated on a full system boot). The configuration algorithm is then executed on a standard PC, with the allocation results diagrammatically illustrated in Figure 5-2.

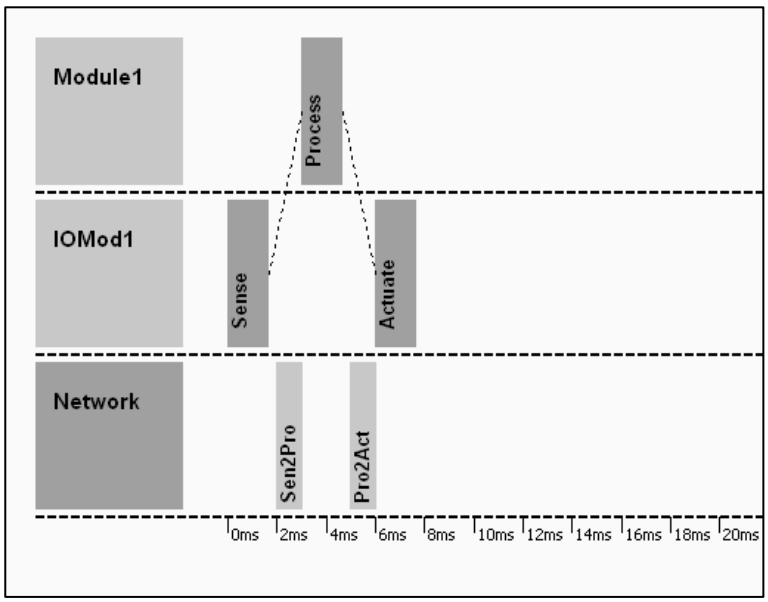


Figure 5-2 Allocation results of Single Sensor, Single Process and Single Actuator

Figure 5-2 shows the available resources down the left hand side (including the network as a resource) and a time base along the base. Functions are shown as allocated to a processing module and to a period of time in which to execute. Communications are allocated to the network and also as a block of time. It can be seen that concurrency of events is maintained in that the 'sense' application executes before the 'Sen2Pro' data packet is sent, and resultant applications and communications follow suit. If a

communication or application was scheduled to execute before its pre-requisite function had finished, then concurrency of events within the time frame is not being facilitated. Figure 5-2 shows that the complete time frame in which these functions can repeatable act is 8 ms.

5.1.2. Single Sensor, Duplex Processing, Single Actuator

This arrangement of functions introduces a redundant processing channel within the network. Here, the 'sense' application is required to send two communication packages; one to each of the 'process' applications. In turn, the 'actuate' application is required to have received two data packets from each of the 'process' functions before it can execute without raising an error.

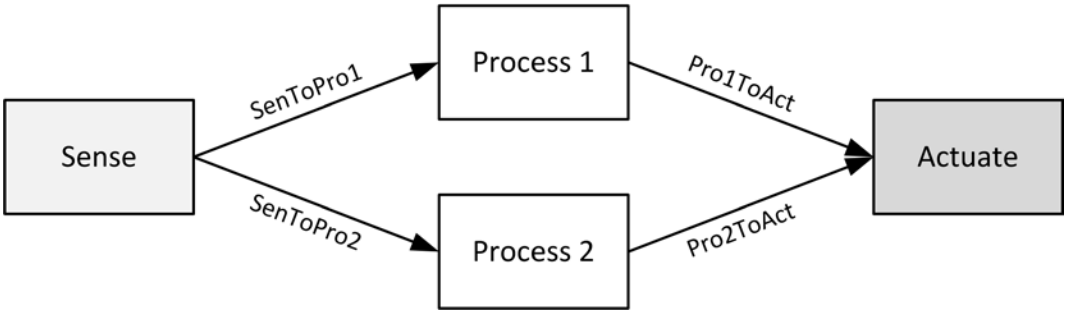


Figure 5-3 Single sensor, duplex processing, single actuation

It is clear in this circumstance that to maintain redundancy within the 'process' applications, two generic processing modules will be required alongside the input/output module as before. Therefore, the required modules are:

- An input/output (I/O) module (Named '*IOMOD*')
- Two standard processing module (Named *'Module 1'* and *'Module 2'*)

The assignment criteria for each application is summarised in Table 5-1. Here, the configuration algorithm is instructed to maintain redundancy between the 'process' applications by the '*!ProcessN*' instruction which enforces 'Do not place this application on a module that also hosts ProcessN'.

Application	Criteria	Description

Sensor	*IOMod*	Placement on the I/O module
Process1	*Mod**	Placement on any generic processing module
	!Process2	Not with Process 2
Process2	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
Actuate	*IOMod*	Placement on the I/O module

Table 5-2 Assignment criteria for duplex processing

The result from the configuration algorithm is shown in Figure 5-4.

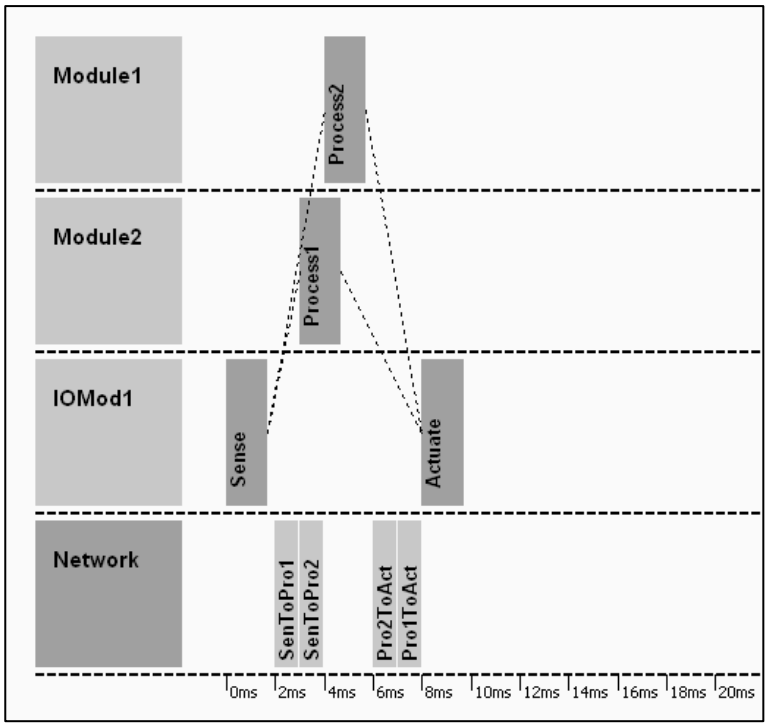


Figure 5-4 - Single Sensor, Duplex Processing, Single Actuator

It can be seen that once again, the applications and communications are placed concurrently. Although 'Process2' is a parallel function to 'Process1', it has to wait an extra millisecond for the delivery of data packet 'SenToPro2' to execute. Conversely, the 'Actuate' function has to wait for receipt of both datapackets (or appropriate error messages) before it can execute.

5.1.3. Single Sensor, Triplex Processing, Single Actuator

The arrangement of these functions is similar to that in the previous section, with the addition of a third parallel 'Process' application, as shown in Figure 5-5.

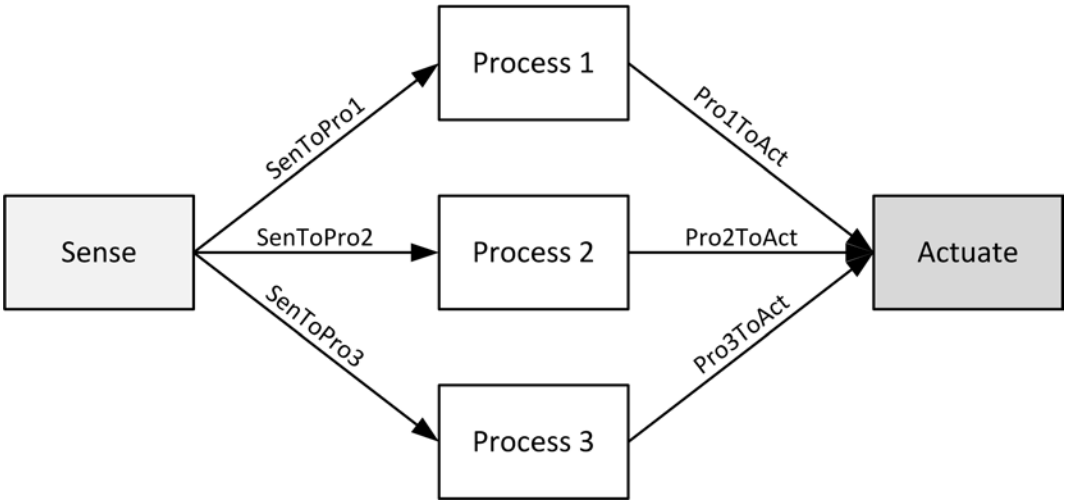


Figure 5-5 Single sensor, Triplex processing, single actuator

The third processing channel introduces additional allocation requirements in order to preserve the redundancy in the event of a module failure. This is shown in Table 5-3.

Application	Criteria	Description
Sensor	*IOMod*	Placement on the I/O module
Process1	*Mod**	Placement on any generic processing module
	!Process2	Not with Process 2
	!Process3	Not with Process 3
Process2	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process3	Not with Process 3
Process3	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process2	Not with Process 2
Actuate	*IOMod*	Placement on the I/O module

Table 5-3 Assignment criteria for triplex processing

It can also be surmised that in order to allow this system to be implemented effectively, a third generic processing module is required. The result of the configuration algorithm is shown in Figure 5-6.

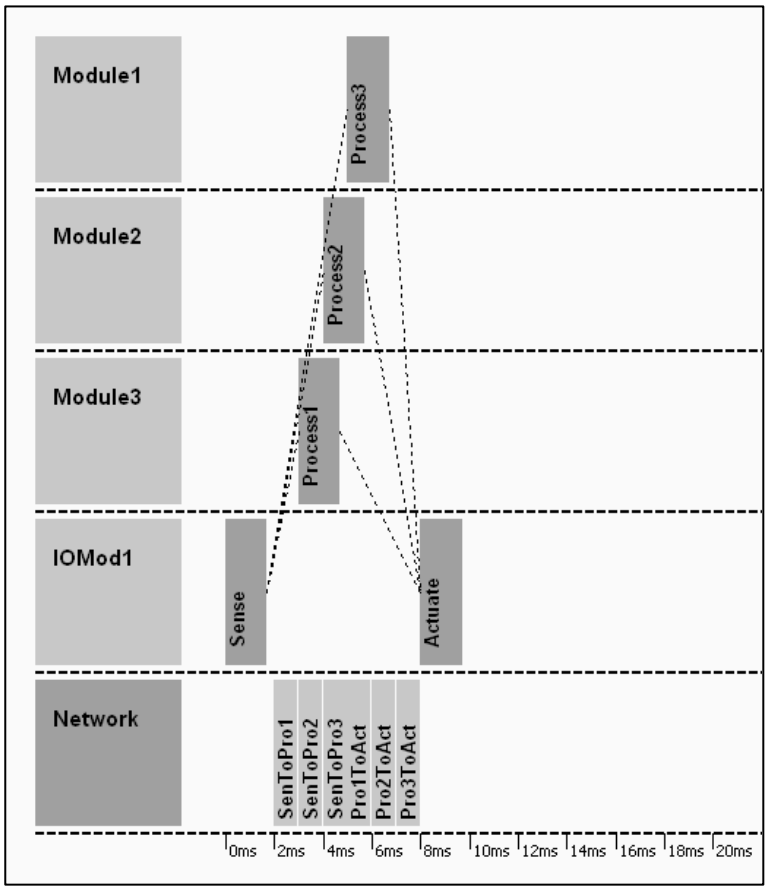


Figure 5-6 Allocation Results of Single Sensor, Triplex Processing, Single Actuator

With this particular arrangement of function, there is some argument that the third module may not be necessary. Detailed failure mode analysis may show that, with appropriate partitioning in the modules themselves, good enough levels of reliability could be achieved even when placing two 'Process' applications on one module. However, for the purposes of this exercise, a more simple view is adopted.

Comparing Figure 5-6 and Figure 5-4, it can be seen that the expected end time of the 'Actuate' application is the same. This is a result of two things: in the duplex case the process is slowed as some applications have to wait for data to be delivered before they

execute; In the triplex case, some applications or data transmissions can execute whilst others are waiting.

5.1.4. Dual Sensors, Triplex Processing, Dual Actuator

To further complicate the processing and communication requirements, in this case it is assumed that both the sensing and actuator functions are duplicated. The resultant network of functions is shown in Figure 5-7.

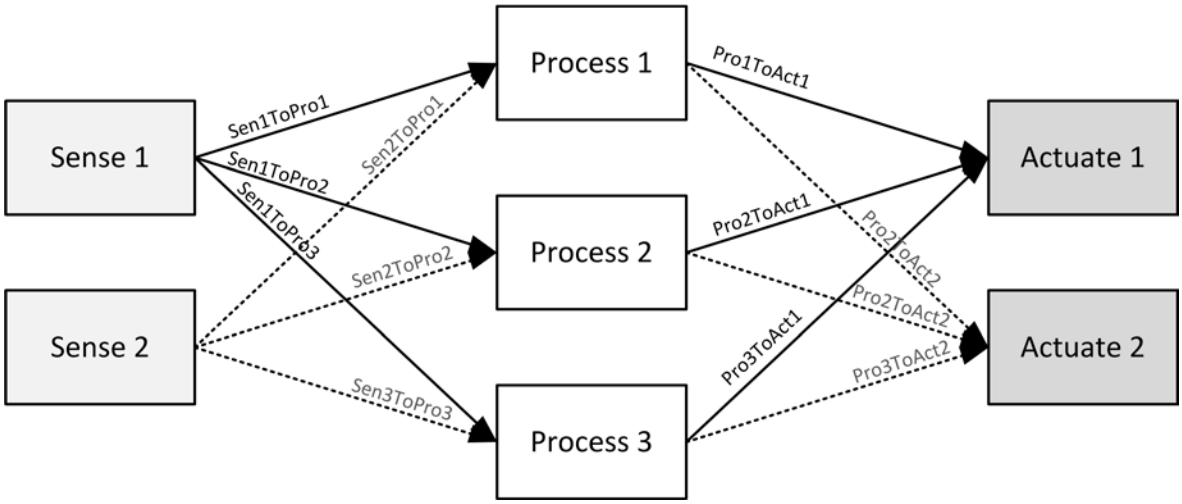


Figure 5-7 Dual sensors, triplex processing, dual actuation

It is also assumed in this case that the second sensor and actuator require a second input/output module on the network. Therefore, the required modules are:

- Two input/output (I/O) module (Named '*IoMod1*' and *'IOMod2'*)
- Three standard processing module (Named *'Module 1'* ,*'Module 2'* and *'Module 3'*)

The assignment criteria for this system is defined in Table 5-4.

Application	Criteria	Description
Sensor1	*IOMod1*	Placement on the I/O module 1
Sensor2	*IOMod2*	Placement on the I/O module 2
Process1	*Mod**	Placement on any generic processing module
	!Process2	Not with Process 2
	!Process3	Not with Process 3

Process2	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process3	Not with Process 3
Process3	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process2	Not with Process 2
Actuate1	*IOMod1*	Placement on the I/O module 1
Actuate2	*IOMod2*	Placement on the I/O module 2

Table 5-4 Assignment criteria for triplex processing and dual I/O

The resultant configuration generated is shown in Figure 5-8.

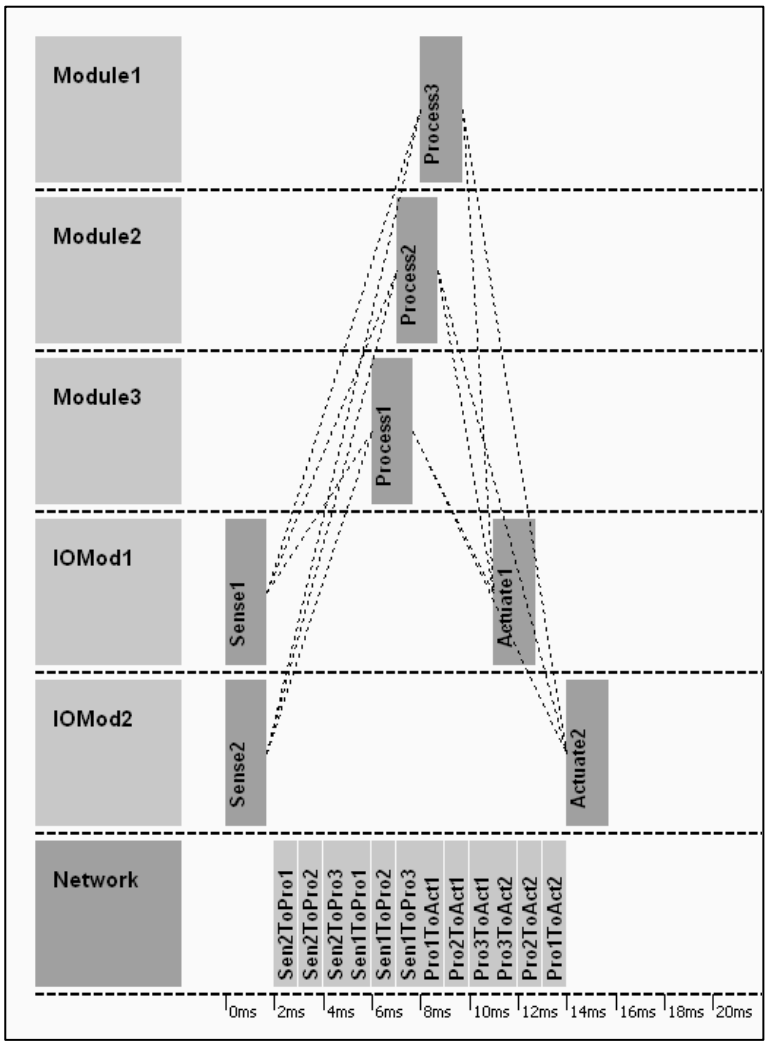


Figure 5-8 Allocation Results of Dual sensors, triplex processing, dual actuation

The algorithm has successfully arranged applications and communications in a concurrent manner. The complete timeframe finishes at 16 ms, a further 6 ms slower than the standard triplex processing scenario. It can be seen that the main cause of this delay is the availability of the network time in which to transmit the required data packets.

5.1.5. Dual I/O, Triplex processing and parallel function

In this case, a function of lower criticality is added introduced to the scenario whereby some data is to be collected from the sensors by a data logging application, and sent to a user interface for viewing. This functional network is shown in Figure 5-9.

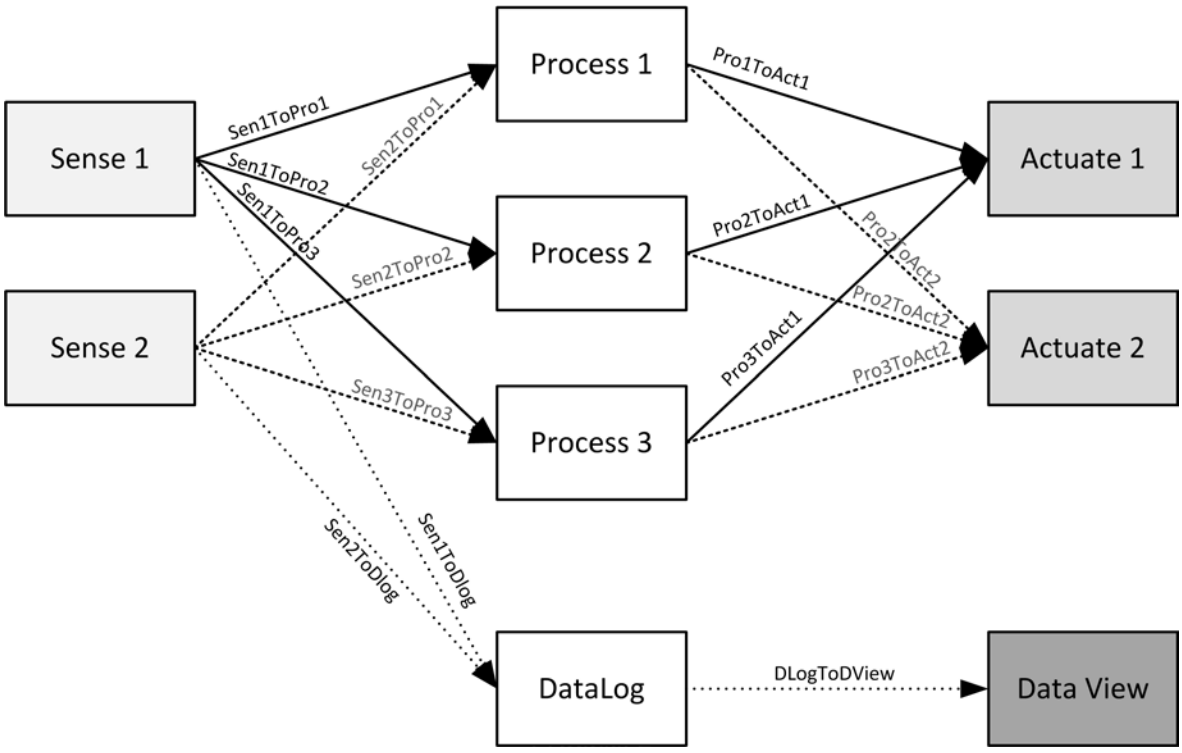


Figure 5-9 Dual sensing, triplex processing, dual actuation plus non-critical functions

In this scenario, a further module is required on the network to display data to a user. Therefore, a minimum set of modules required for this network is:

- Two input/output (I/O) module (Named '*IoMod1*' and *'IOMod2'*)
- Three standard processing module (Named *'Module 1'* ,*'Module 2'* and *'Module 3'*)
- A graphical user interface module (Named '*GUI*')

The allocation criteria is summarised in Table 5-5

Application	Criteria	Description
Sensor1	*IOMod1*	Placement on the I/O module 1
Sensor2	*IOMod2*	Placement on the I/O module 2
Process1	*Mod**	Placement on any generic processing module
	!Process2	Not with Process 2
	!Process3	Not with Process 3
Process2	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process3	Not with Process 3
Process3	*Mod**	Placement on any generic processing module
	!Process1	Not with Process 1
	!Process2	Not with Process 2
Actuate1	*IOMod1*	Placement on the I/O module 1
Actuate2	*IOMod2*	Placement on the I/O module 2
DataLog	*Mod**	Placement on any generic processing module
DataView	*GUI*	Placement on the GUI module

Table 5-5 Assignment Criteria for Dual Sensing, Triplex Processing, Dual Actuation and Datalog

The resultant configuration is shown in Figure 5-10

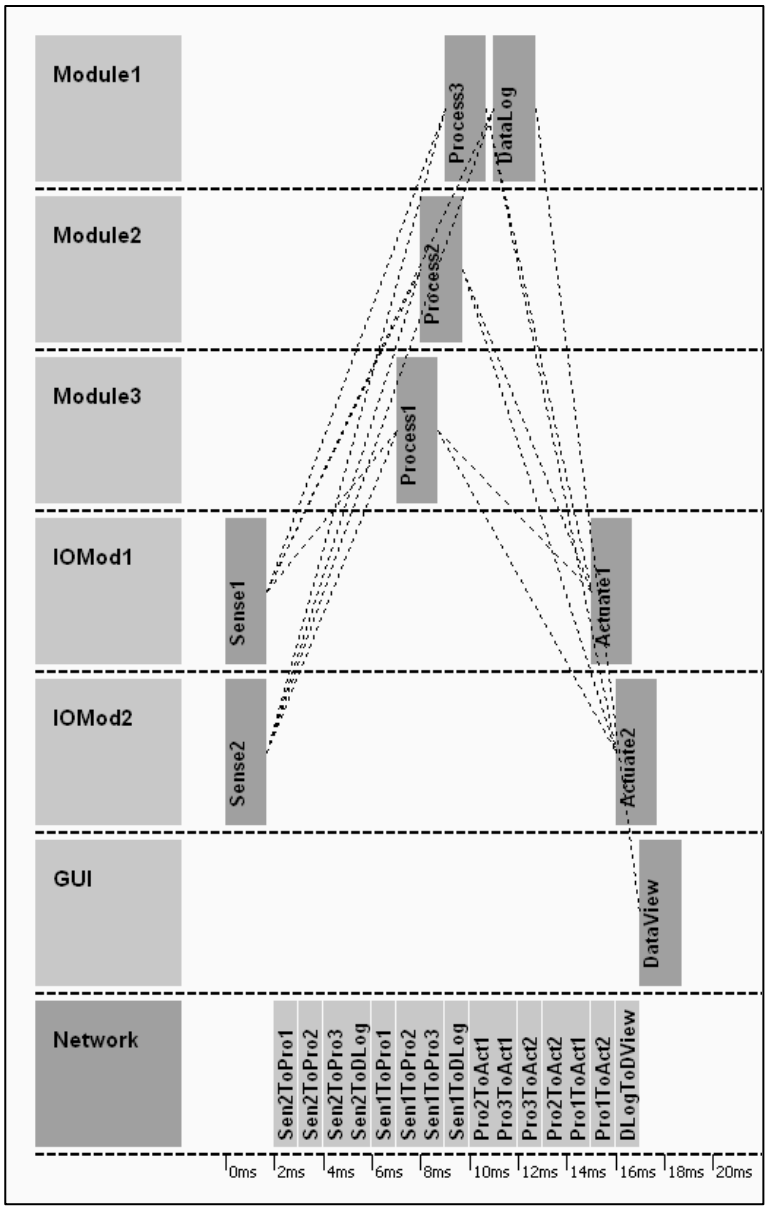


Figure 5-10 Allocation Results of Dual Sensing, Triplex Processing, Dual Actuation and Data log

The goal of the allocation function is to place the non-critical function with minimal disruption to the time-critical task. It can be seen that this is largely the case in that communications and applications concerned with the time-critical task operate first, then the data log task operates afterwards. The only interruption occurs in the transmission of 'Sen2ToDlog' and 'Send1ToDlog'. There is no logical reason as to why these cannot wait until later in the frame to execute. The reason they are placed earlier is that they are associated with the applications 'Sense1' and 'Sense2', which are both time critical

functions. By association, some weight is placed to the criticality of the output of these functions. The mis-placement of this data packet does cause some delay in the time-critical function, but the impact is considered minimal. It can be seen that the 'Actuate' functions finish by the 18ms mark, only 2 ms slower than the previous case which contained no data log function. The configuration algorithm could be optimised to remove this effect, but the resultant configuration generated is still a reasonable one and the automatic generation algorithm is considered fit for purpose for this investigation.

5.1.6. Summary

The above networks contain a simple example of how the configuration mechanism derives an appropriate allocation of functions based on the assignment definition defined and the available processing resources. For each of the basic functional topologies presented a configuration is automatically realised in a way that protects the static redundancy channels and maintains appropriate functional partitioning by allocation to available resources and temporally to avoid data packet clashes on the network bus.

There have been areas highlighted where the assignment process may be optimised further, but these tests have shown that it does ensure concurrency of events and maintain appropriate segregation to preserve redundancy in processing channels.

5.2. IMS as a Distributed Real Time System

The restricting element of implementing a real-time structure in this form of IMA is the reliability and repeatability of the communication. As described in Chapter 5, the system is designed such that the communication and the application execution timings are linked to ensure concurrency of events.

The goal of this section is to observe how well the IMA, in particular the Ethernet structure, handles the real time communications tasks.

5.2.1. Real time attributes testing

In this experiment, the IMA is provided with a simple token passing function, the design of which is shown in Figure 5-11. Here a number is sent from the GUI module and passed via numerous applications on other generic IMA modules before being returned to the GUI.

A further required function is a data packet between the designed GUI and the systems management function. This allows the user to interact with various systems management activities. Although these items are not discussed here, the function is included for completeness as it is required to be included for operation.

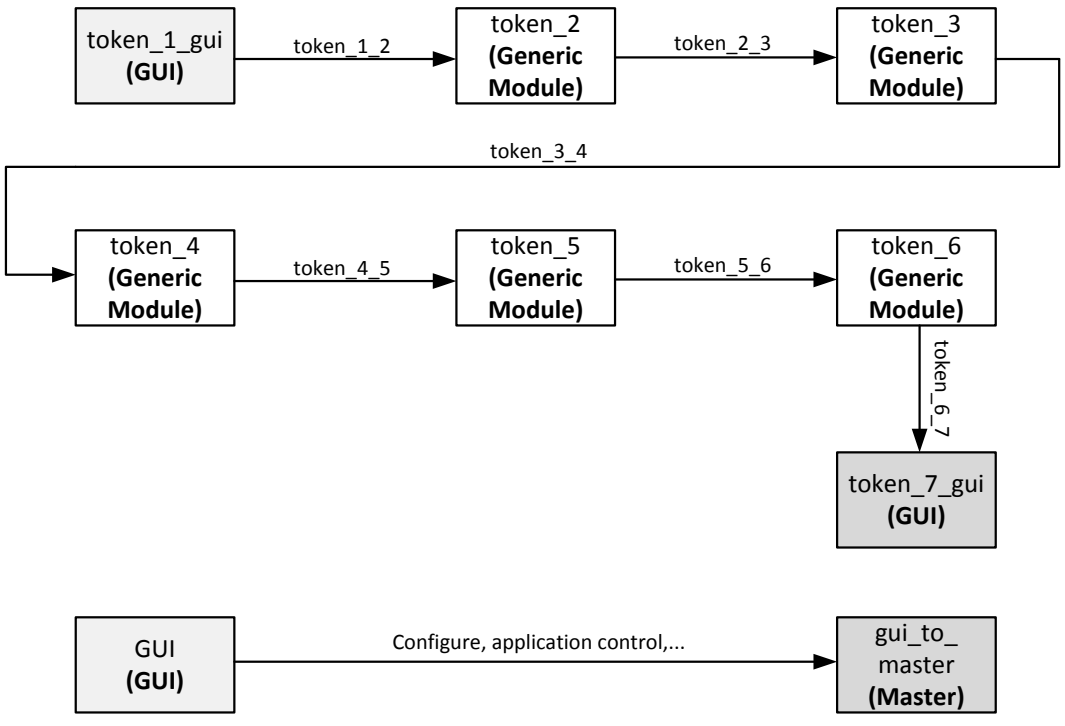


Figure 5-11 Function Network Design for Token Passing Exercise

The applications are assigned to specific modules in order to allow the token to cross the network as many times as possible during its transit. The allocation of applications is shown in Figure 5-12.

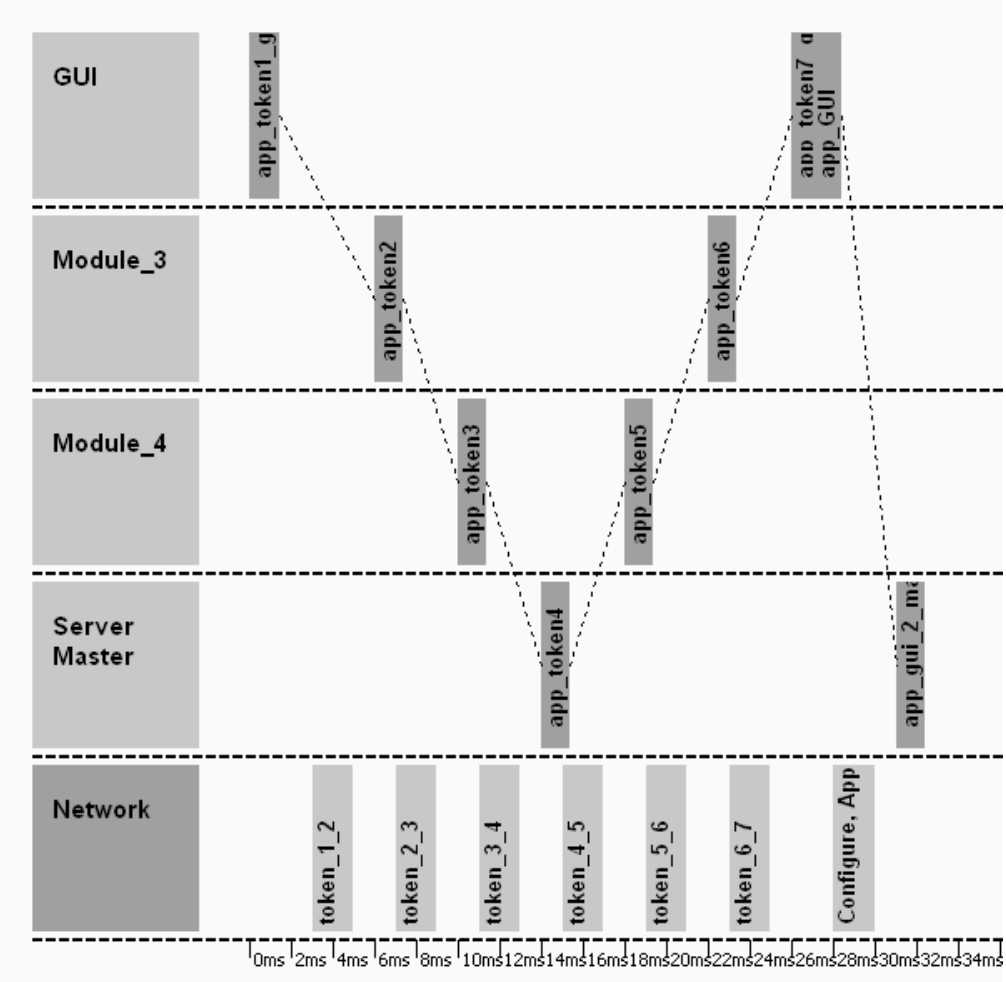


Figure 5-12 Allocation of Token Passing Functions to IMA (3 ms communication time allowed)

Within each module, communication timing data is captured. Each communication activity is recorded relative to the start of the global time frame, and each time frame has a time stamp for identification defined by the clock on the 'Server Master' module. This time stamp is communicated as part of the synchronisation signal. Following the execution of the above function structure, communication data logs from each module can be downloaded and analysed to generate analysis for the entire network. A sample of data from a single time frame is shown in Table 5-6. The information captured is:

- **Timestamp of frame (ms)** – the millisecond timer value captured by the master module at the start of the frame. This is sent to all modules.
- **Time in frame (ms)** – the time within the frame the recorded activity takes place. This is timed from the receipt of the 'go' data packet.
- **Error Code** – records any error occurring with the activity. The most common is a timeout error which is captured here.
- **Rx/Tx** – Receipt or Transmit. Records the type of activity into sending or receiving.
- **To/From I.P.** – records the target IP address of the data packet. This can be cross-referenced to identify the module name.
- **Variable Name** – the name given for the data packet. This is taken from the original application definition.
- **Ready From (ms)** – shows at what point in the frame the activity is waiting to act. The time is measured from the receipt of the 'go' data packet.

The data shown in Table 5-6 has been amalgamated from all modules.

Timestamp of frame (ms)	Time in Frame (ms)	Error Code	Rx/Tx	To/From I.P.	Variable Name	Ready From (ms)
85417	3	0	Tx	192.168.0.3	token_1_2	1
85417	3	0	Rx	192.168.0.1	token_1_2	1
85417	7	0	Tx	192.168.0.4	token_2_3	3
85417	9	0	Rx	192.168.0.3	token_2_3	1
85417	11	0	Tx	192.168.0.2	token_3_4	9
85417	12	0	Rx	192.168.0.4	token_3_4	0
85417	15	0	Tx	192.168.0.4	token_4_5	12
85417	17	0	Rx	192.168.0.2	token_4_5	11
85417	19	0	Tx	192.168.0.3	token_5_6	17
85417	19	0	Rx	192.168.0.4	token_5_6	7
85417	23	0	Tx	192.168.0.1	token_6_7	19

85417	24	0	Rx	192.168.0.3	token_6_7	3
85417	28	0	Tx	192.168.0.2	Configure	24

Table 5-6 Sample Communication Data from a Single Time-frame

Using a MATLAB algorithm for the complete amalgamated communication data, the following attributes for each communication can be calculated:

- **Average time of occurrence (ms)** – average time within the frame the communication activity takes place.
- **Variance** – the variance in arrival time. This is calculated by:

  $$\sigma^2 = (x_i - \bar{x})^2/n$$

  Where $\sigma^2$is the variance, x_i is the time of the sample, $\bar{x}$ is the mean arrival time of all the samples and n is the number of samples.
- **Number of timeouts** – how often the data packet fails to arrive in time. A timeout occurs if the allotted communication time has passed, and the data packet has not been received.
- **Latest recorded time** – the latest time (within the timeout period) that the data package is recorded to have arrived.

The following sections record a series of tests where the communication is reduced until not enough time is being allowed for the data to be exchanged.

5.2.1.1. Communication time: 3 ms

In this experiment, a generous communication time of 3 ms is allowed for each data packet transaction, and 200 ms is defined for each time frame. This provides opportunity to record timing information on late packets as opposed to counting them only as occurrences of timeouts. Furthermore, although the data is recorded in coarse values of milliseconds, over 700 time frames are captured for analysis. By analysing a large number of samples, a reasonable resolution for timings can be achieved.

Table 5-7 shows a summary of the results. Analysis was only performed on the receipt of data packets along with the transmission of the first, as transmission of the others were

controlled in such a way that they always occurred exactly on time. The first transmission is found to only vary slightly but is thought to be a result of the preceding synchronisation data packet.

	token_1_2	*token_1_2*	*token_2_3*	*token_3_4*	*token_4_5*	*token_5_6*	*token_6_7*
	Tx	Rx	Rx	Rx	Rx	Rx	Rx
Expected Time (ms)	3	3	7	11	15	19	23
Average Time (ms)	3.233	3.245	8.079	**12.291**	16.000	20.078	24.084
Variance (ms)	0.181	0.826	0.990	0.702	**0.993**	0.989	0.680
Latest (ms)	5	6	9	14	17	22	26
Timeouts	0	1	0	0	0	0	**18**

Table 5-7 Communication timing results – 3 ms communication time (824 time frames)

The results in Table 5-7 show that the average receipt time is normally between 0.2 - 1 ms after the expected time, with a variance of about 0.75 seconds. Therefore it is expected that most data packets are received around 1.75 ms after the expected time, but some packets are arriving as much as 3 ms late.

The worst results in Table 5-7 are highlighted. The most timeouts occur on the GUI module. This is expected as in this particular experimental setup as this function is hosted on a 'Windows' PC (for purposes of user interface) as opposed to the specific real-time module developed. It is therefore subject to interruption from the operating system. The worst performing module is the Master module as it is the latest to process the receipt of token_3_4 and the transmission of token_4_5. This fault was found to be in the synchronisation of the time frame. The master module broadcasts a message to signal the start of the frame, but this message is subject to the same delay and variance as is seen in the above communications. It is therefore easier to co-ordinate two servant modules than it is to estimate the delay experienced on the master module.

5.2.1.2. Communication time: 2 ms

The allocation of the token passing function with 2 ms allowed for communication time is shown in Figure 5-13.

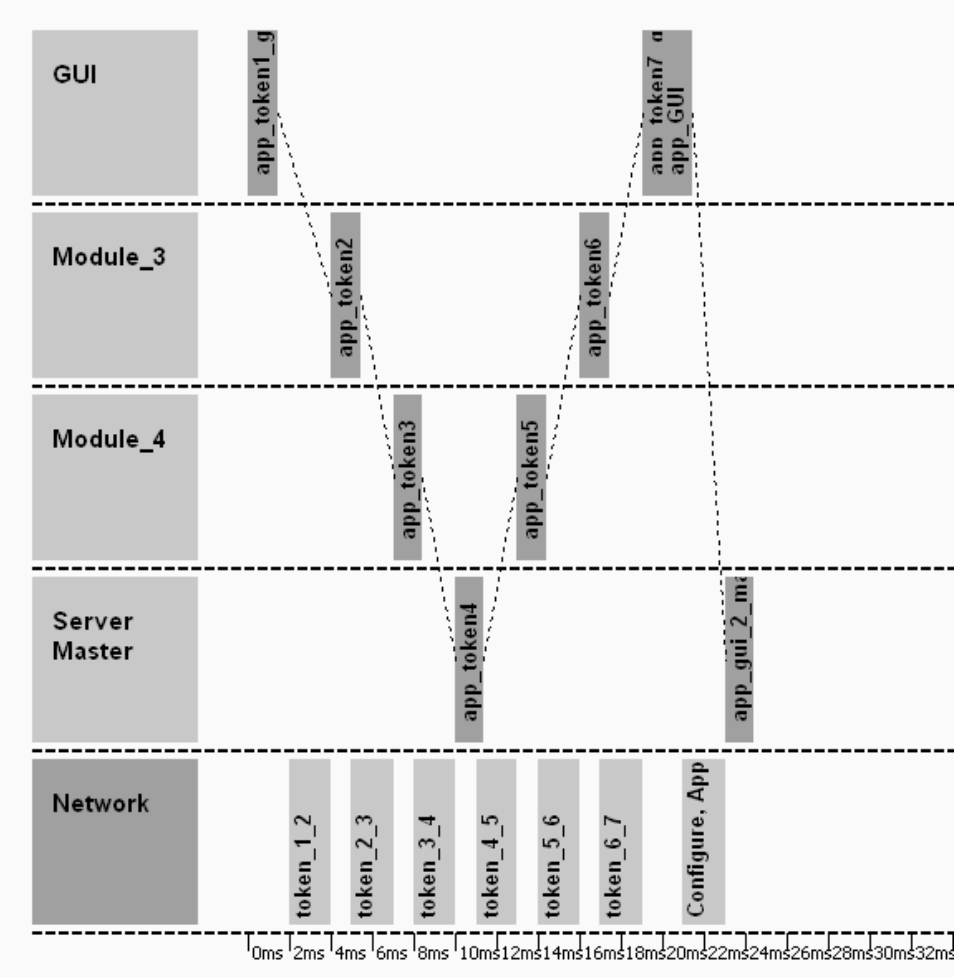


Figure 5-13 Allocation of Token Passing Functions to IMA (2 ms communication time allowed)

The summarised timing data for the implementation of this configuration is shown in Table 5-8.

	token_1_2	*token_1_2*	*token_2_3*	*token_3_4*	*token_4_5*	*token_5_6*	*token_6_7*
	Tx	Rx	Rx	Rx	Rx	Rx	Rx
Expected Time	2	2	5	8	11	14	17
Average Time	2.793	2.706	6.121	**9.202**	11.933	14.977	18.059
Variance	0.169	0.678	0.982	0.520	**0.989**	0.475	0.627

Latest	4	4	7	10	13	16	20
Timeouts	0	0	0	0	0	0	**197**

Table 5-8 Communication timing results – 2 ms communication time (902 time frames)

It can be seen that the IMA manages the 2 ms communication time reasonably well in that the average time and variances of data receipt are similar to the previous 3 ms test. The main difference identified is that the GUI module experiences a timeout error nearly 22% of the time (197 times out of 902 time frames). Again, the Master module experiences marginally worse performance than its peers (i.e. slower to receive token_3_4 and slower to transmit token_4_5) which is due to the synchronisation issues highlighted previously.

5.2.1.3. Communication time: 1 ms

The final test in this series reduces the allowed communication time to 1 ms. The allocation of functions is shown in Figure 5-1.

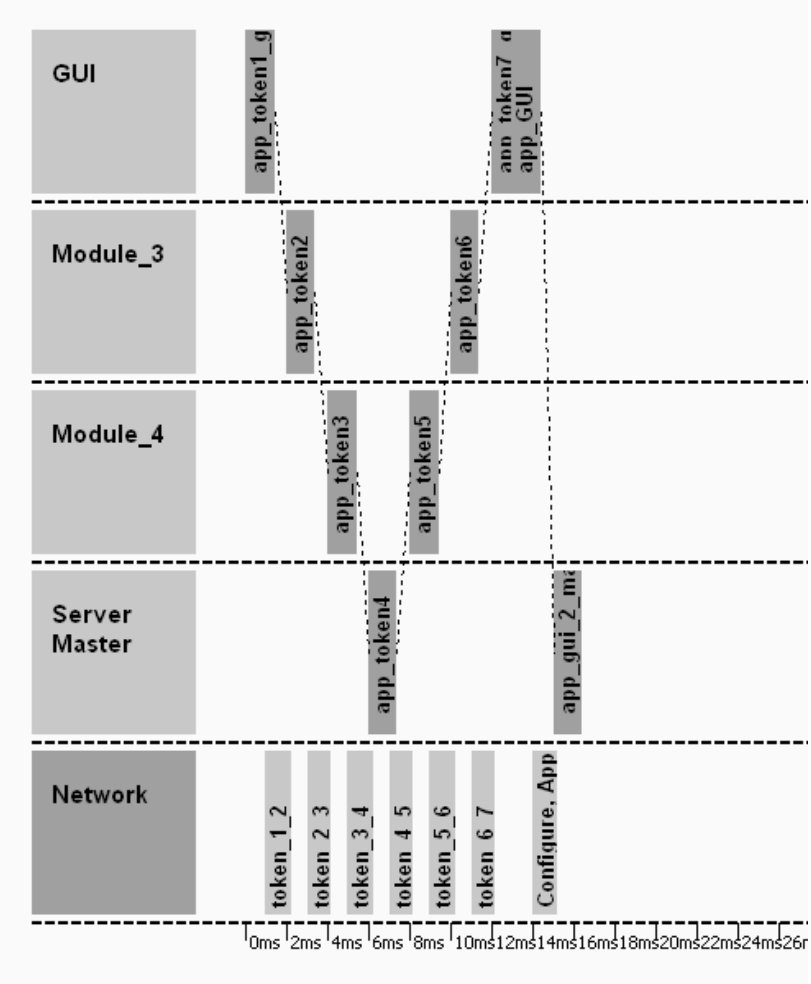


Figure 5-14 Allocation of Token Passing Functions to IMA (1 ms communication time allowed)

The amalgamated timing data is summarised in Table 5-9.

	token_1_2	*token_1_2*	*token_2_3*	*token_3_4*	*token_4_5*	*token_5_6*	*token_6_7*
	Tx	Rx	Rx	Rx	Rx	Rx	Rx
Expected Time	1	1	3	5	7	9	11
Average Time	1.178	1.189	3.549	6.000	6.679	9.955	12.040
Variance	0.147	0.153	0.250	0.000	0.571	0.247	0.568
Latest	2	2	4	6	8	10	13
Timeouts	0	0	148	**589**	2	159	367

Table 5-9 Communication timing results - 1 ms communication time (828 time frames)

It can be seen that in this case, the system experiences a large number of timeout errors on a number of modules. This level or unrecorded data invalidates the timing analysis as most of it is discarded. Although some communication is transmitted and received, the reliability is much too low for any distributed real-time control application.

5.2.1.4. Summary

These experiments have shown that the use of Ethernet within the communication structure developed does allow for repeatable message communication in a concurrent fashion. This is an essential attribute of a distributed real time control system. Although the allocation of function was performed in a manual fashion by setting appropriate values in the application definitions, the timing structure was generated automatically using the configuration algorithm. These tests have shown that the system is capable of generating this structure and then maintaining control of it during execution.

In its current form, the IMA is not capable of fast (i.e. greater than ~10Hz sample frequency) distributed real time control. Although Ethernet has a theoretically high bandwidth, it has been shown here that standard off-the-shelf components cannot provide the functional capability for this type of application. Ethernet is designed to send large

packets of data in a single direction. This application requires many small packets of data to be transmitted and received in a short time scale. The delays highlighted by analysing timing data occur as the Ethernet cards require 1-2 milliseconds to assess if the bus is empty. Customised Ethernet hardware may be able to remove this inbuilt property.

A further issue in this case is that timing functions within the system can only operate up to 1 ms resolution due to use of LabVIEW as a development environment. This in itself causes further discrepancy in timings and is troublesome when attempting to synchronise different modules.

The implication of this is that distributed real time control of the Maglev rig is difficult as the sample rate is too slow. Initial assumptions were that a single time frame containing a number of communications could be readily fit into 10 milliseconds. It has been found that it is more appropriate to assume that a single time frame will take 100 milliseconds; an order of a magnitude slower than expected.

5.2.2. Air Gap control

In order to test the ability of the IMA to operate a real time application, a software simulation of the rig dynamics will be used. Here, the parameters of the Maglev rig are adjusted such that the dynamic properties react 10 times slower than in reality. The distributed IMA can then be tasked with controlling this system.

The full design of the Maglev Rig and airgap control can be found in Appendix A and C. The control loop to be implemented is shown in Figure 5-15.

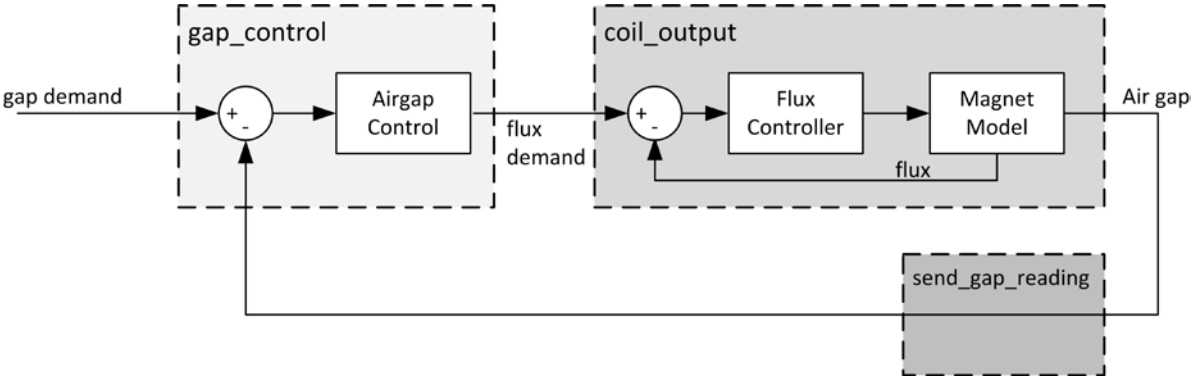


Figure 5-15 Air gap control schematic

The dashed boxes identify the real time applications that are defined to perform this function. For the purposes of the demonstration, the gap control is required to be distributed on a separate module to the 'coil_output' function.

5.2.2.1. Application Design

The control laws that need to be executed within each timeframe have been calculated in Chapter 4 and it is these functions that will be implemented for airgap control.

Figure 5-15 shows the required network of functions to be implemented.

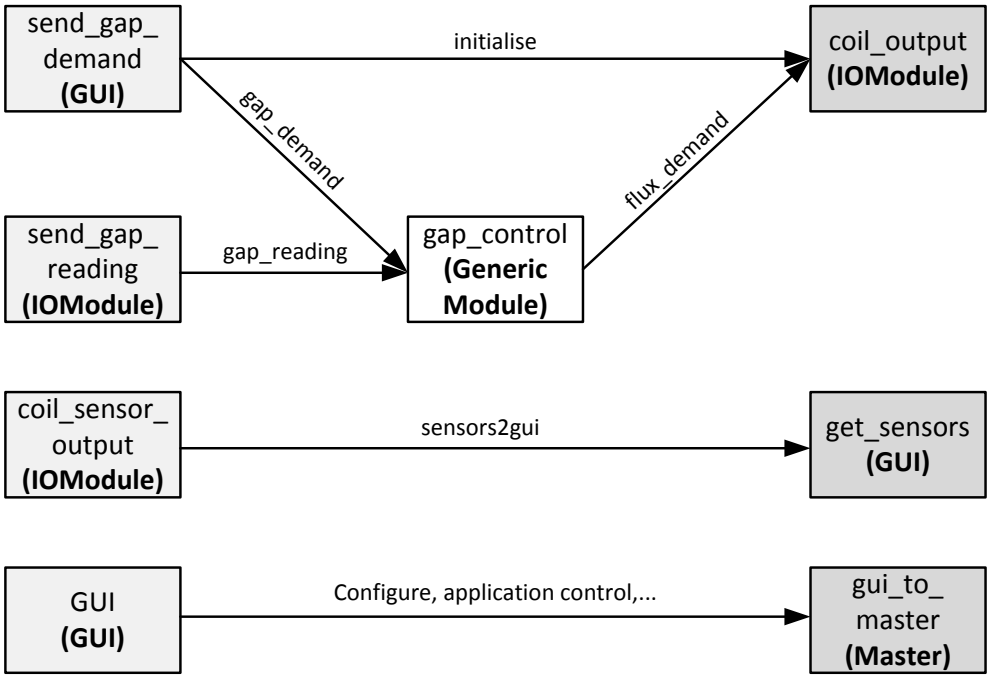


Figure 5-16 Function Network for Gap Control Structure

In this realisation of the designed controller, the flux control action is placed within the 'coil_output' application and the gap control action is placed remotely in the 'gap_control' application. The flux loop described has a high bandwidth so its function will be allocated to the input/output module on which the simulated coil is housed. The air gap control loop has a lower bandwidth and can be allocated on any of the generic processing modules. In this case, the allocation process will be forced to place this function on neither the GUI module nor the I/O module in order to make sure the data packets have to use the network.

For the purposes of a user interface, the coil sensor readings are reported back to the user via the 'coil_sensor_output' application. This is not directly part of the control network described above as this data is only 'for information' and not part of the time critical loop.

Further to the control action, the required system management communication between the 'GUI' and the 'gui_to_master' app is present.

For the purposes of testing the control action, data will be recorded from the 'send_gap_demand' application and the 'coil_output' application. By using the global time

stamp, this data can be collected, amalgamated and analysed after the test has been executed to observe the response of the system as a result of the changing input.

5.2.2.2. IMA Implementation of Distributed Gap Control

In this implementation, each application was given 1 ms to for execution completion and each communication was allocated 3 ms to transmit and receive. The I/O module to which the coil is physically connected is also allocated as the server master module. The time frame allocated is 50 ms for each iteration. The result of the configuration algorithm to these criteria is summarised in Figure 5-16.

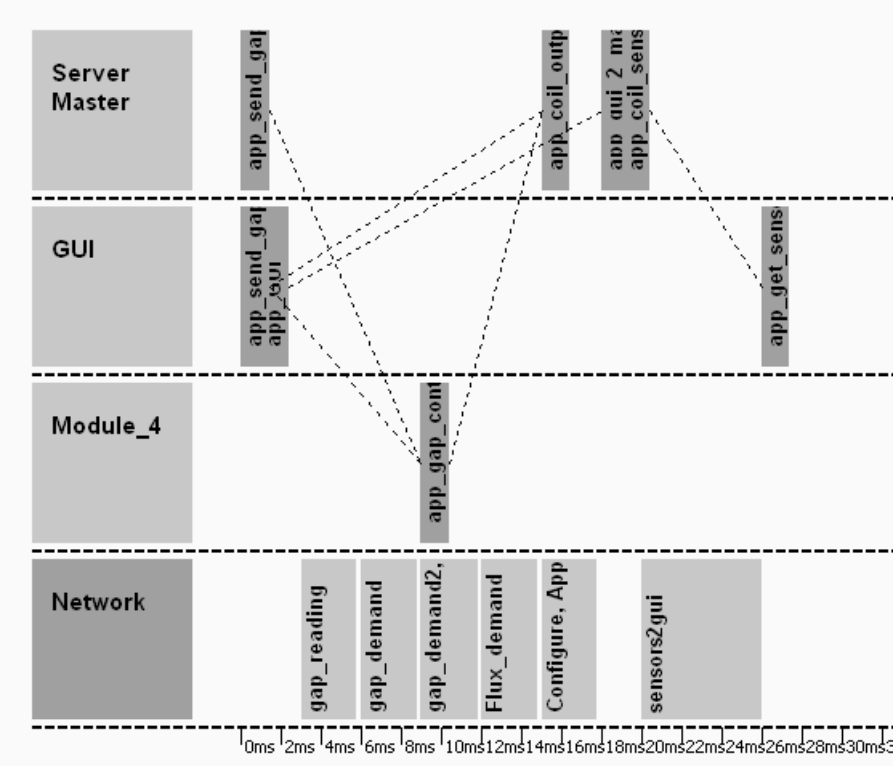


Figure 5-17 IMA Implementation of Gap Control Functional Network

The system will be tested around the nominal operating point of a 10 mm air gap. This will be done by a +/- 2mm step wave around a centre point of 10 mm.

5.2.2.3. Results of Gap Control Implementation

A snapshot of data from this test is shown in Figure 5-17. The full length test lasted for over two minutes, and this data is taken from 47 seconds into the test and lasts for a positive and negative step change in airgap demand. The blue dashed line represents the

gap demand, which was recorded at the point of input on the GUI module. The green solid line represents the measured air gap and is recorded on the I/O module. The time of each data point is recorded by monitoring the global time stamp of the start of the frame (using the central 'Server Master' time reference) then adding the recorded offset within the frame itself. Due to the communication delays analysed in the previous section, it was found that it took approximately 13 ms from the start of the frame for the updated flux demand to be implemented.

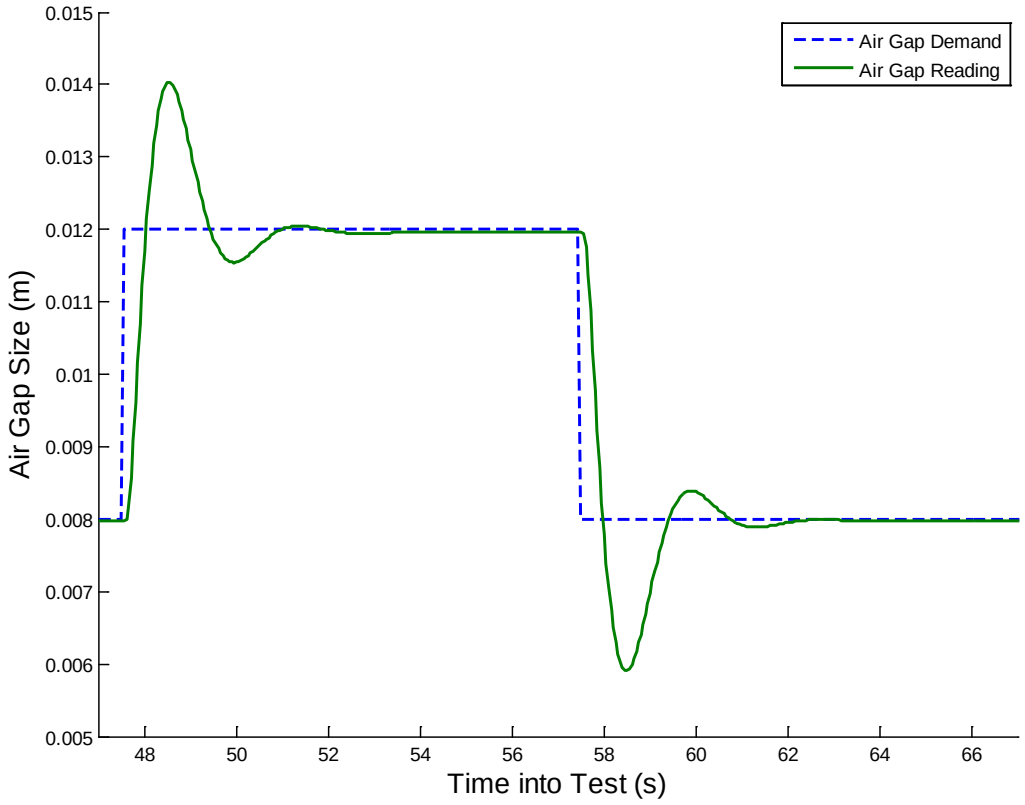
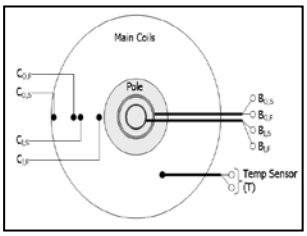


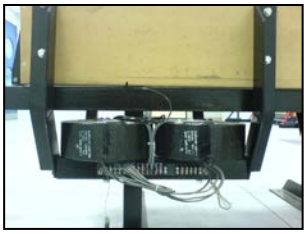
Figure 5-18 Time Response of Simulated Air Gap to Step Input

5.2.2.4. Distributed Gap Control Summary

It can be seen in the results presented Figure 5-17 that the communication method implemented is consistent enough to facilitate distributed control architecture. In this case, a simulation model has been used as a target test subject as the overall time frame could not be reduced sufficiently to operate the real system.

However, it has been shown that the design of the IMA is able to automatically allocate functions to resources in a manner that ensures concurrency of events and allows the establishment of real time, distributed architecture. The next chapter will test the ability of the system to maintain these properties during the re-allocation of functions as a response to fault occurrence.





CHAPTER 6:

Fault Management Within IMA

6. Fault Management within IMA

This chapter will discuss how certain aspects of fault management are implemented within the IMA created for the purposes of this project. It will describe how the system uses the configuration/reconfiguration functions available to perform fault management actions. The main focus here is on fault tolerance and how this is managed to maintain service in the presence of faults.

6.1. Baseline System Initial Configuration

As discussed earlier in the thesis, the overall goal of the IMA is to maintain the airgap of a maglev test rig. A simple realisation of this was presented in Chapter 6 to test the ability of the IMA to maintain an effective control loop. In this Chapter, it is necessary to extend this system to include static redundancy, and flexibility within the hardware to allow for the possibility of system reconfiguration.

Figure 6-1 shows the control loop required to maintain airgap control and is taken from the description of the Maglev rig and subsequent controller design that can be found in Appendices A and C. The dashed boxes in Figure 6-1 represent the real-time applications that will be defined for implementation.

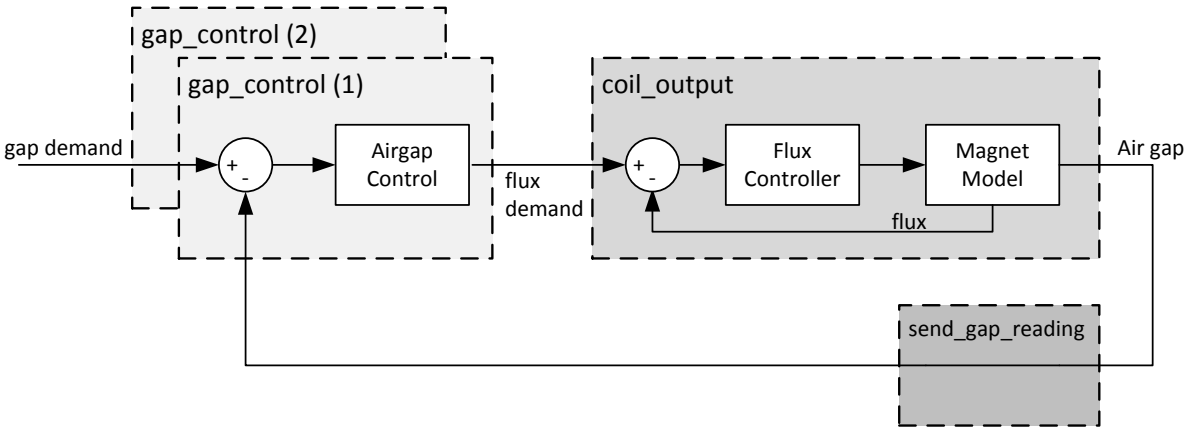


Figure 6-1 Control Loop Realisation for Airgap control

It can be seen from Figure 6-1 that some functions are duplicated to provide a level of static redundancy. These functions and the redundant channels are best highlighted in the function network shown in Figure 6-2.

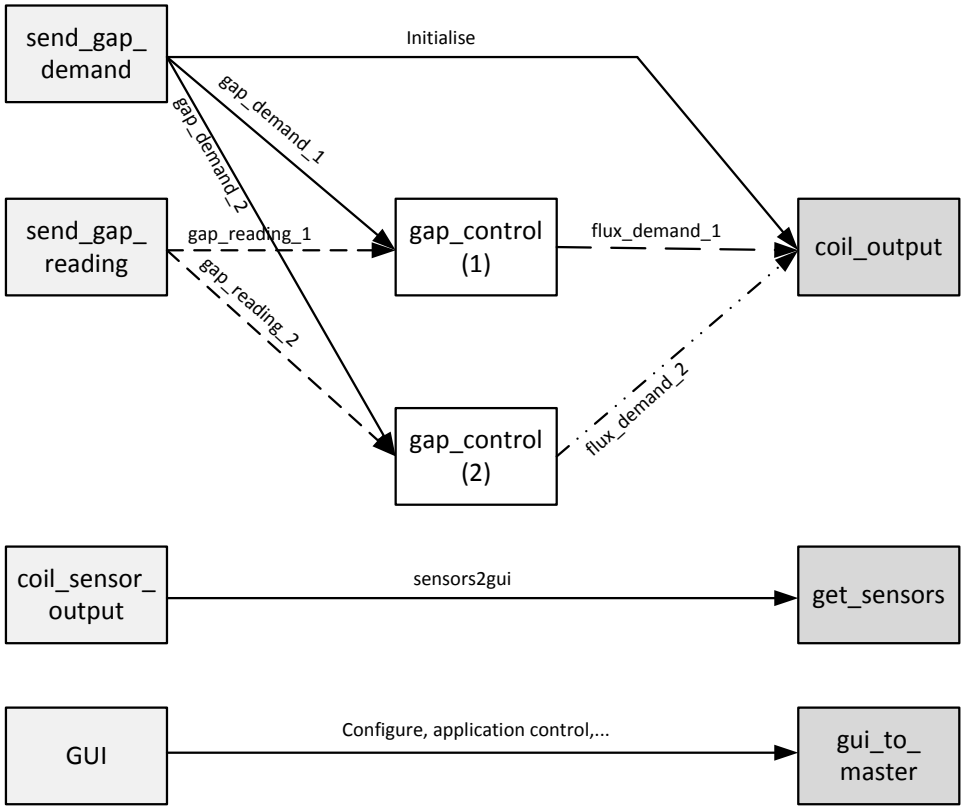


Figure 6-2 Network of Air Gap Control Functions

The following describes the functions in detail. The 'Inputs' and 'Outputs' refer specifically to network communications requirements and not to physical connections to the hardware module.

Application 1: send_gap_demand

- **Description** – Collects the airgap demand from the user and provides it as a reference for the gap control loop.
- **Inputs** – None. The reference is input from the user via the GUI.
- **Outputs**
    - *gap_demand* - sent to each instance of gap_control and is indexed accordingly.

  - *initialise* - sent to the mathematical model of the coil and allows a manual reset of the internal variables within the model.

- **Assignment** – *GUI*

Application 2: send_gap_reading

- **Description** – Collects sensor data from I/O module and transmits the latest airgap reading to the gap controller. This application differs from coil_sensor_output as it is part of the time critical control loop, whereas coil_sensor_output is for user reference.
- **Inputs** – None. Readings taken from sensors onboard I/O module, which in this case is a real-time simulation of a single magnet.
- **Outputs** – *gap_reading*. This is sent to each instance of gap_control, and is indexed to reference each specific data packet.
- **Assignment** – *IOMod*
-

Application 3: gap_control(1&2)

- **Description** – Performs the gap control function defined in Chapter 4. The rate this controller can be implemented is defined by the time frame of the network communication structure.
- **Inputs** – *gap_demand*, *gap reading*
- **Outputs** – *flux_demand*
- **Assignment** – *!GUI*.

Application 4: coil_output

- **Description** – Performs the flux control loop, and generates the drive signal to the coil. In this realisation, the coil model is housed within this application, such that the flux feedback becomes a return of an internal variable as opposed to a sensor reading.
- **Inputs** – *flux_demand*

- **Outputs** – none. This represents the end of the control functions for this time frame.
- **Assignment** – *IOMod* - This has to be placed on the I/O module to interact with the coil.

Application 5: coil_sensor_output

- **Description** – Provides a communication channel to feed sensor data back to the user. By specifying this function separately to the control loop functions allows the systems to designate this as 'low priority'.
- **Inputs** – none. The information generated here is obtained from sensor readings (or in this case, the mathematical model)
- **Outputs** – *sensors_2_gui*. Contains the flux, current, air gap and drive voltage applied.
- **Assignment** – *IOMod* - This has to be placed on the I/O module to interact with the coil.

Application 6: get_sensors

- **Description** – receives sensor data from the I/O module and provides information to the user via a GUI.
- **Inputs** – sensors_2_gui
- **Outputs** – none. The information is displayed to a user.
- **Assignment** – *GUI*. This has to be placed on the GUI module in order to interact with the user.

Application 7: GUI

- **Description** – provides the mechanism for the user to interact with the systems management functions housed on the 'Master' module. It is this application that allows the injection of simulated faults.
- **Inputs** – none.
- **Outputs**

  - *configure* – commands the system to go from an initial condition to implementing the main function
  - *application_control* – allows simulated fault signals to be injected to an applications
  - *module_control* – allows simulated fault signals to be injected to a module.
- **Assignment** – *GUI*. This has to be placed on the GUI module in order to interact with the user.

Application 8: gui_2_master

- **Description** – Interprets the information provided by the 'GUI' application and passes information or commands to the server master functions..
- **Inputs**
  - *configure*
  - *application_control*
  - *module_control*
- **Outputs** – none.
- **Assignment** – *Master*. Has to be situated with the Master module.

There are four modules available for these resources to be assigned. These are:

- **Server Master** – generic module but also houses the server master functions
- **GUI** – a module that has user interface input/output abilities
- **IOMod_3** – an input/output module
- **Module_4** – a generic processing unit, with no specific hardware attachments

Using the methods explained in Chapter 5, and tested in Chapter 6, the system configuration shown in Figure 6-3 is realised using the above hardware and functional architecture and assuming no faults.

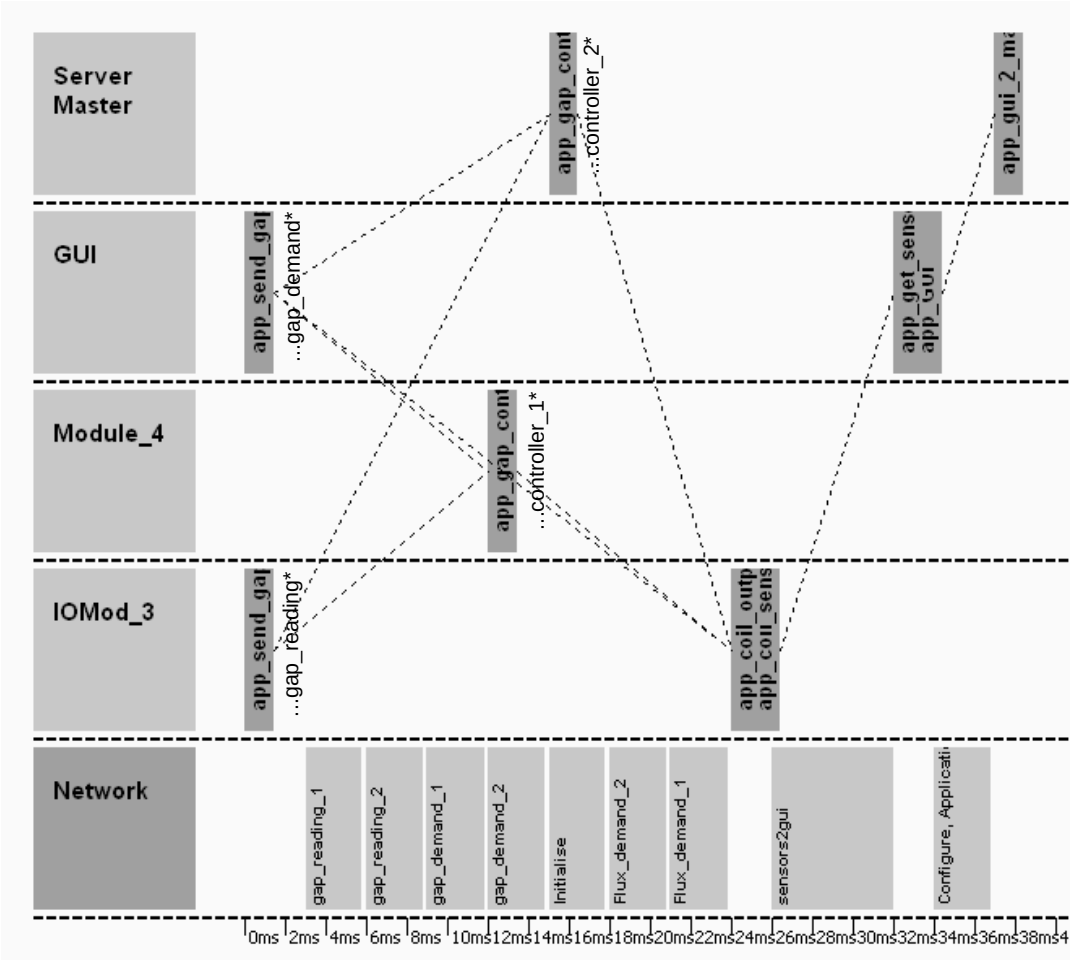


Figure 6-3 Assignment of functions to resources For duplex gap controller

*Due to the automatic generation of figures, some text is lost and replaced here for clarity.

This assignment of functions is the starting point for the following tests where failures can be simulated in parts of the system and the response observed.

6.2. System Response to a Single Fault

This section describes how the system responds to the introduction of simulated faults. It will be shown how the fault is initially tolerated before reconfiguration restores higher levels of redundancy.

The following sections detail two different faults, and describe how the system responds to their introduction. After each experiment, the system is restored to pre-fault condition to allow each fault to be assessed in isolation.

6.2.1. Simulation of an Application Failure

In this failure simulation, a fault will be injected into the application "app_gap_controller_1". This will be injected at a point where the system is in normal operation in that the IMA is fully configured and the airgap is following a defined set point. This application has been chosen as it represents a time critical function and is a relatively straight-forward application to duplicate.

6.2.1.1. Fault Injection

The further following assumptions are made about this fault:

- There will be no output from this application (fail quiet)
- There will be no propagation of failure from this application to the wider operating system (fault containment)
- It will still be possible to terminate this application.
- Faulty information being propagated down-stream is protected either by information from the Systems Manager (after the fault is reported), or by voting / model-based input monitoring (at the instant of failure).

The application directly affected by the introduction of this fault is shown in Figure 6-4.

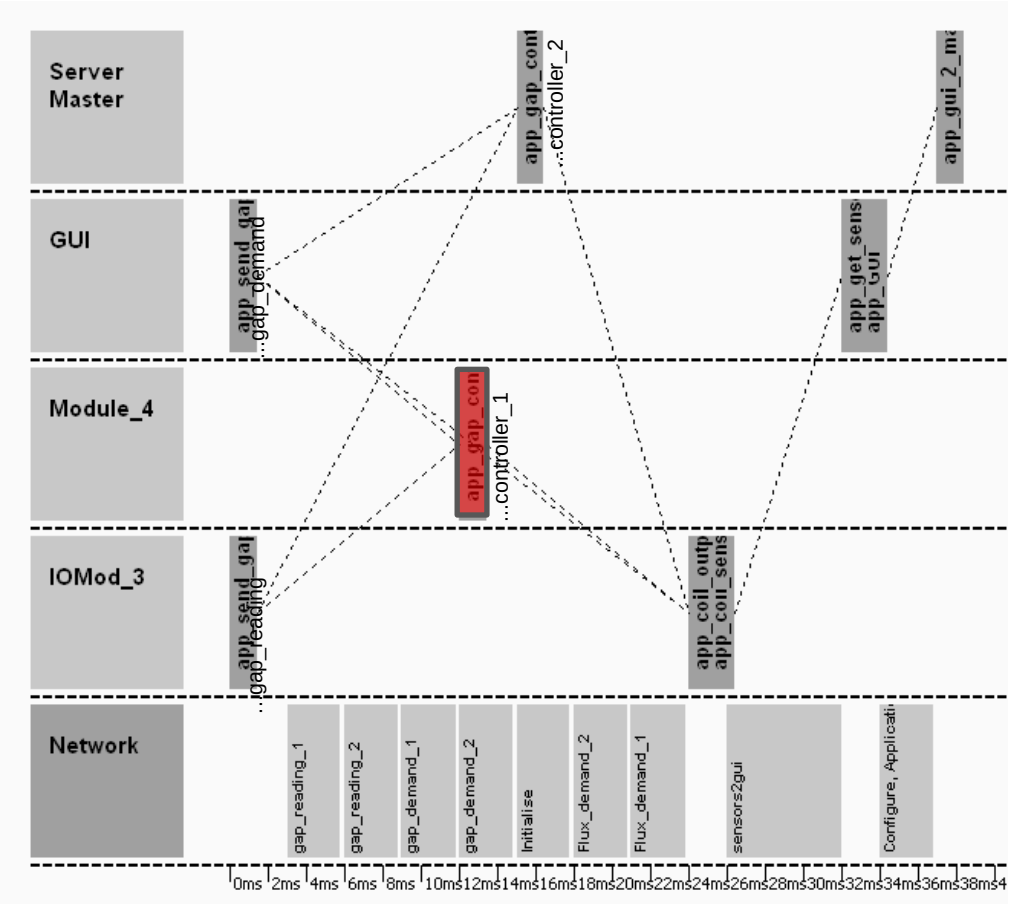


Figure 6-4 Application failure introduced to 'gap_controller_1'

The introduction of this fault creates a change to the functional network of the system as an instantaneous result of this failure. It can be seen that the application downstream of the failure still receives information from the second gap control application, but the system is now operating at a minimal level of redundancy and as such cannot survive a failure in the second gap_control application. However, by using traditional methods for designing redundancy into the functional network, the system tolerates the initial occurrence of the fault. This is highlighted in Figure 6-5, but importantly it provides time for reconfiguration activities to occur without interim loss of service.

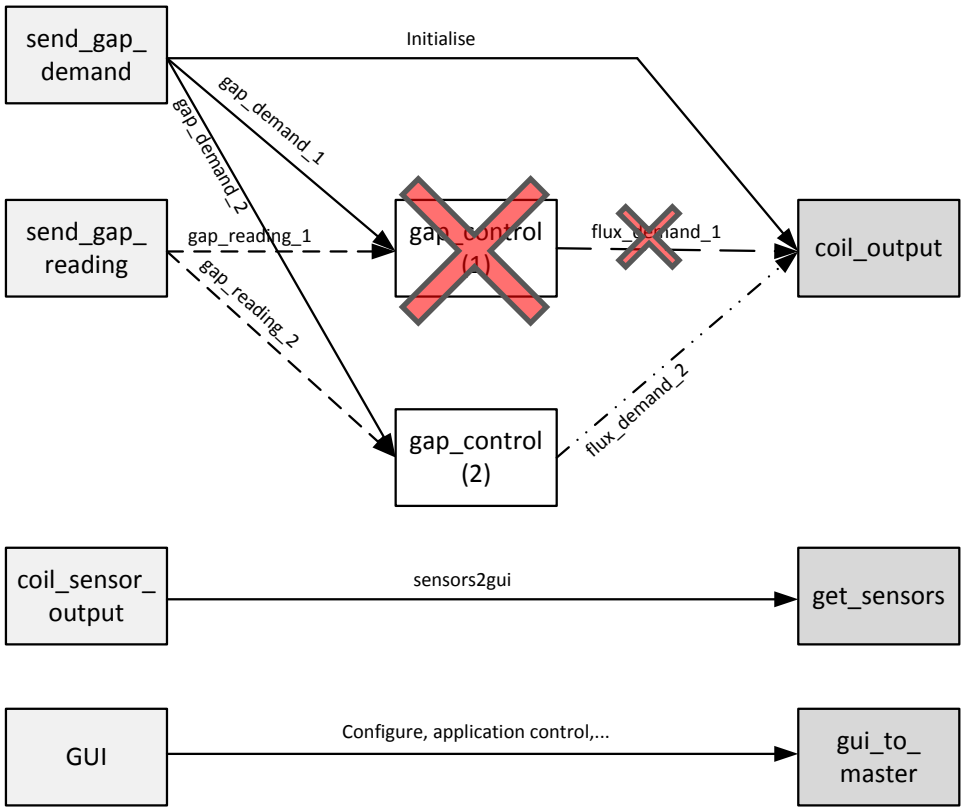


Figure 6-5 Effect of failure on the functional network

6.2.1.2. Reconfiguration to Restore Higher Levels of Redundancy

There are a number of available options to managing an application failure, and some of which depends on the actual failure mode that occurred. The course of action that is adopted here is that the application is forced to re-site on a different processing module. This method is chosen as it demonstrates a clear, tangible effort to restore functional capability as opposed to subtle methods such as terminating and restarting the application.

Forcing an application to be assigned to a different module is realised by adjusting the assignment criteria part of the module specification, then re-executing the configuration algorithm developed. This forces the configuration algorithm into a different assignment of functions, inclusive of the new requirement based on failure.

Assignment Criteria for app_gap_controller_1	
Before failure:	**After failure :**
!GUI	!GUI
!app_gap_controller_2	!app_gap_controller_2
	!Module_4

Table 6-1 Change in Assignment Criteria as a result of Fault Injection

The new allocation of functions is shown in Figure 6-6.

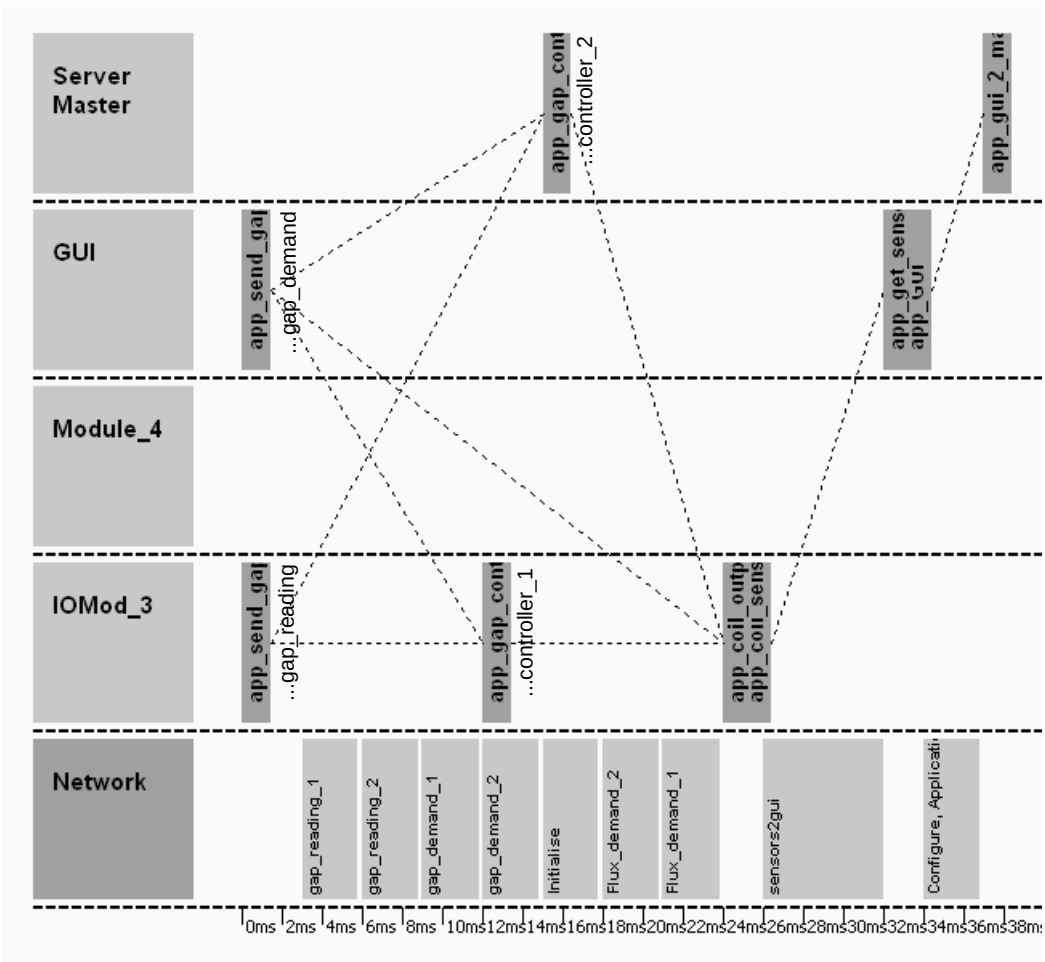


Figure 6-6 Allocation of functions as a result of reconfiguration

The overall result is that the gap control function is housed on 'IOMod3'. Although it appears that only this application has moved, a complete reconfiguration had to occur to ensure that the communication schedule was reworked to accommodate different application addresses.

6.2.1.3. Evaluation of the Reconfiguration Process

Figure 6-6 shows that some level of redundancy has been restored. At first impression, it appears that full redundancy has returned, but only if this assessment is limited to the functional network presented in Figure 6-5. The second gap controller is now located alongside other time critical applications and introduces a bigger potential impact should a failure of module 'IOMod3' occur. However, higher levels of redundancy have been restored as the duplex gap control function is available once again.

The effect of the above series of events on the overall service provision from the system is shown in Figure 6-7. By extracting configuration logs from the Server Master module it was possible to obtain a time reference to the configuration activities and observe the effect of these on the airgap. The airgap demand is recorded on the GUI module, and the airgap reading on IOMod3. During this test the system was commanded to track a square wave air gap demand.

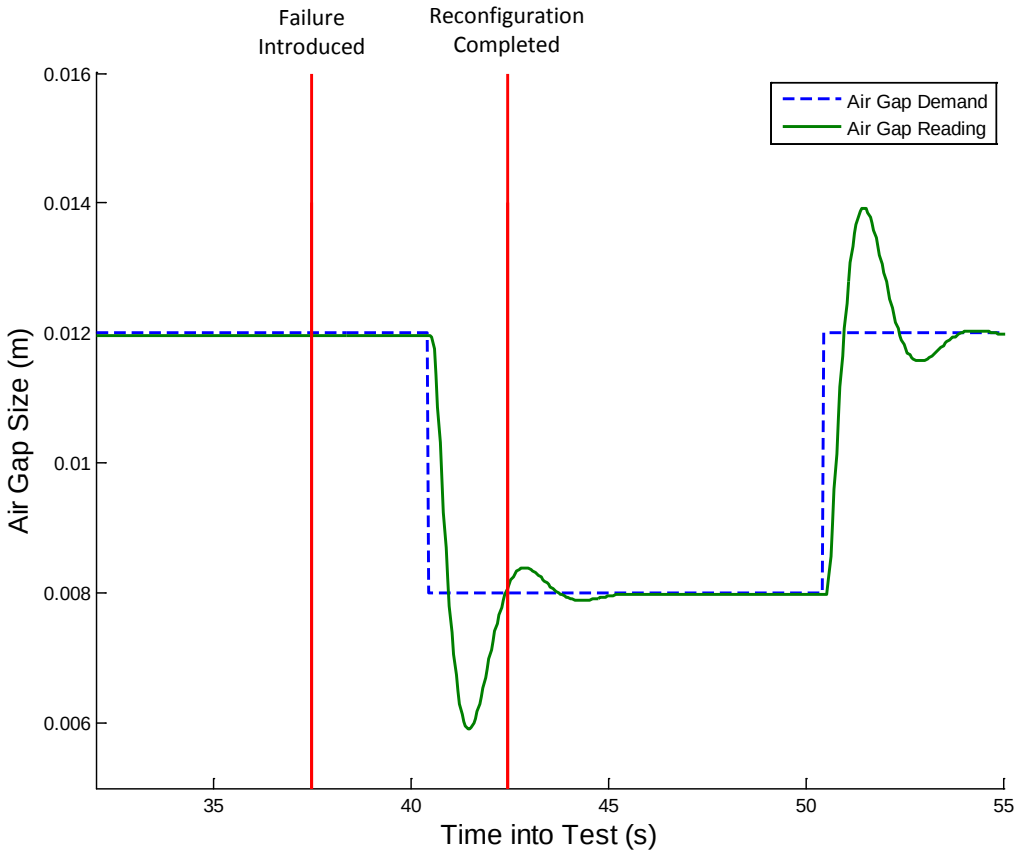


Figure 6-7 Airgap response during reconfiguration activity

It can be seen that the service provision is unaffected by; the introduction of the failure, the period over which the failure is tolerated, and the introduction of a new configuration. It takes five seconds from the failure introduction to implement the new configuration shown in Figure 6-6.

6.2.2. Simulation of a Processing Module Failure

In this failure simulation, a fault will be injected into the module “Module4”. As before, the failure will be simulated at a point where the system is in normal operation in that the IMA is fully configured and the airgap is following a defined set point. The system has been restored to initial conditions such that this failure can be observed in isolation to an application failure.

Module4 has been chosen as it is the only module on the network with a non-specific role. A failure introduced to the GUI, Server Master, or IO Module would result in more complex hardware and software redundancy to accommodate and tolerate faults.

6.2.2.1. Fault Injection

The further following assumptions are made about this fault:

- There will be no output from the entire module (fail quiet)
- There will be no propagation of failure from this module to the wider network (fault containment)
- It will still be possible to terminate and ignore this module.
- Applications affected downstream of any applications that fail, can select the remaining unaffected signal either by information from the Systems Manager, or by monitoring the sanity of the inputs.

The module and applications directly affected by this failure are highlighted in Figure 6-8.

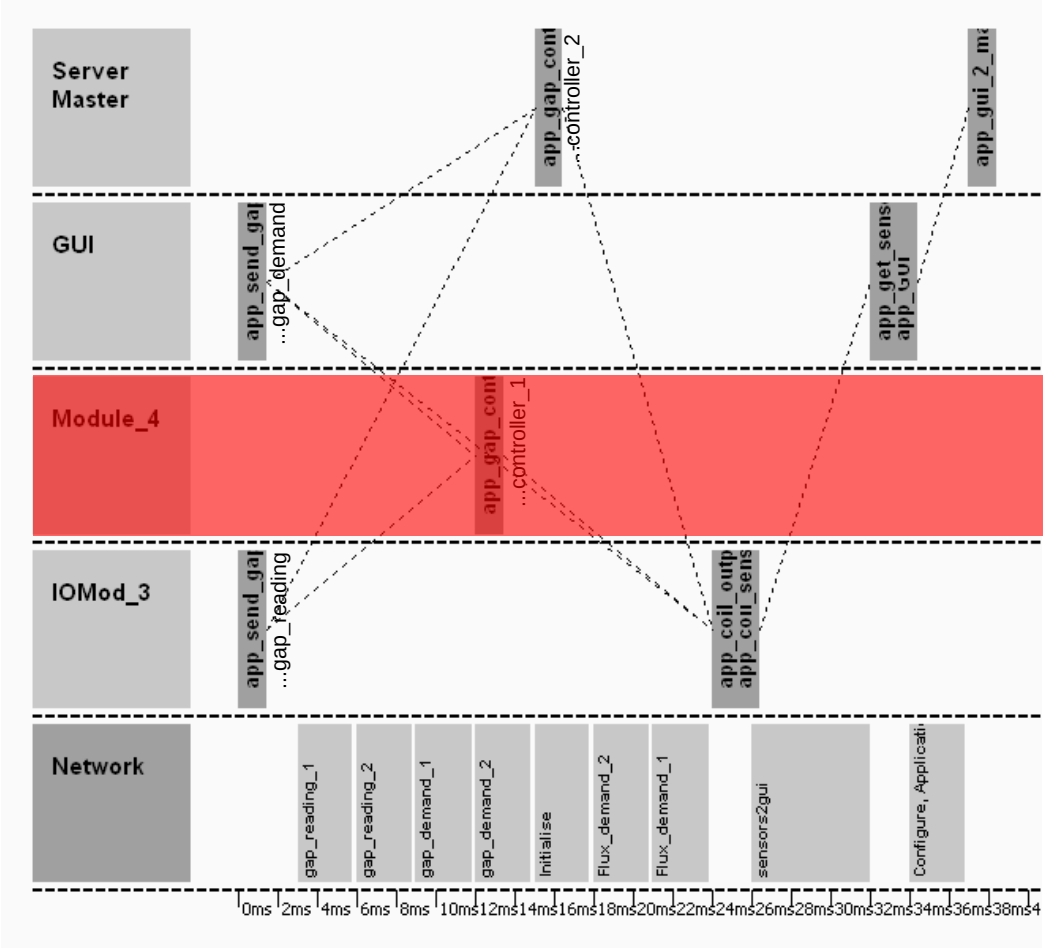


Figure 6-8 Module failure introduced to module 4

As before, the effect on the functional network is the same as the previous failure scenario as only the gap control application is affected. This is shown in Figure 6-9. Again, the fault is tolerated at the instance of failure as a redundant processing channel exists.

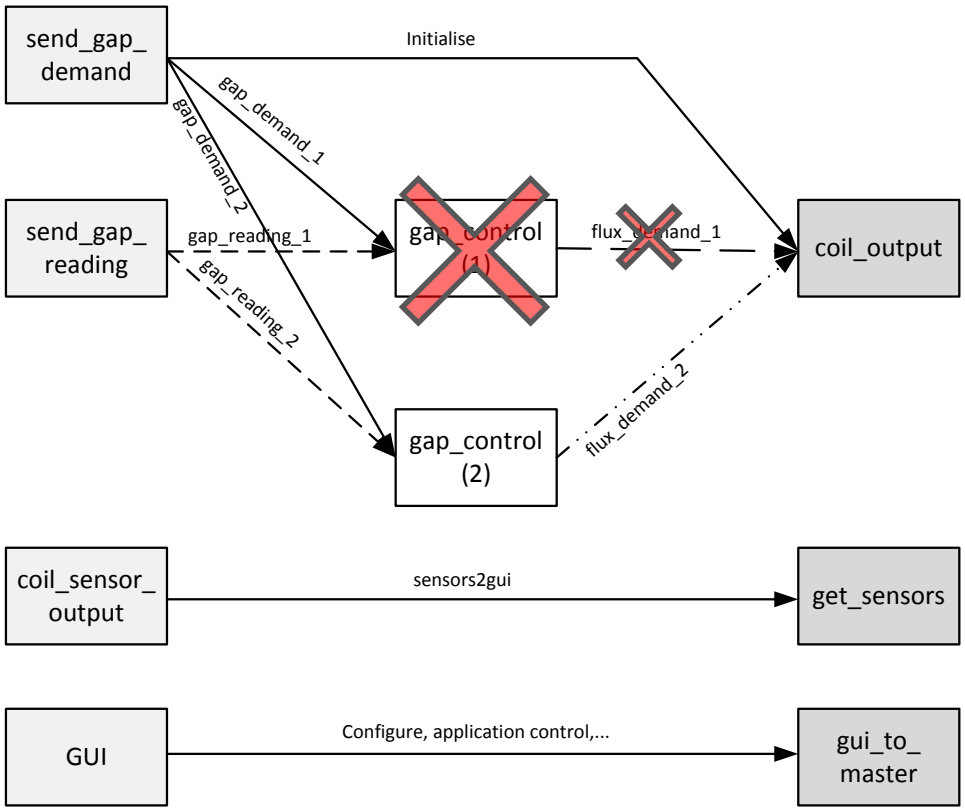


Figure 6-9 Effect of Module4 failure on the functional network

6.2.2.2. Reconfiguration to Restore Higher Levels of Redundancy

In the event of a module failure, the course of action designed is to attempt to reconfigure the system, but ignoring this module as an available resource. It is possible to attempt to reboot the module as a 'running repair', but in this instance it is assumed that it has failed and shut down.

The systems manager therefore initiates a reconfiguration request, but removes this module as an option for function assignment (Table 6-2). It will therefore only find a new configuration if available resource exists on other modules that do not contradict the assignment criteria defined in the system design. As a result of this scenario, the allocation of function generated by the reconfiguration algorithm is shown in Figure 6-10.

Available Modules on the Network	
Before failure:	**After failure :**
Server_Master	Server_Master
GUI	GUI
Module_4	IOMod_3
IOMod_3	

Table 6-2 Change in available modules as a result of failure

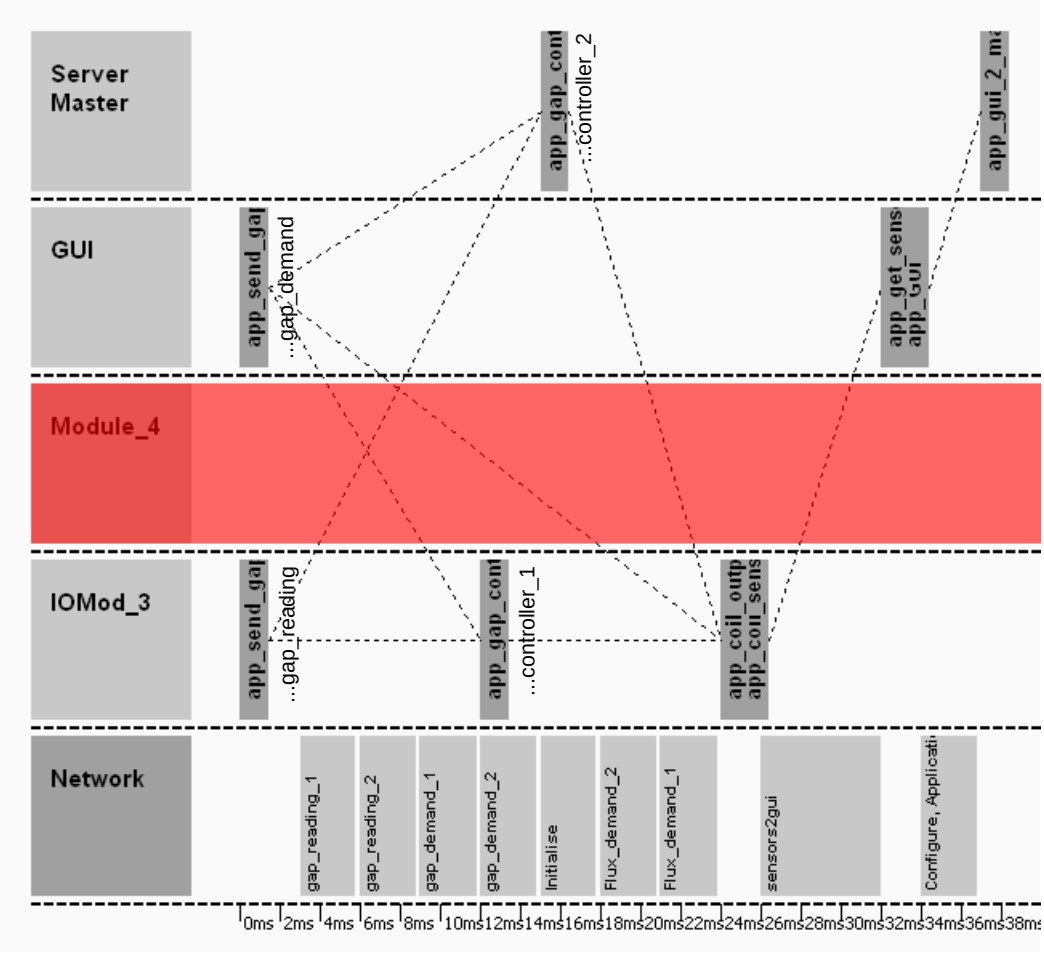


Figure 6-10 Allocation of functions as a result of reconfiguration

The result is that the affected function is reallocated to IOMod_3, restoring some levels of redundancy. This is by chance the same restoration method that occurred previously, but the movement occurred via different reasoning.

6.2.2.3. Evaluation of Reconfiguration Process

As before, the real time data for the air gap analysis and the configuration information can be extracted from event logs situated in each of the processing modules. Figure 6-11 shows the effect of the introduction of the failure and the reconfiguration on the airgap.

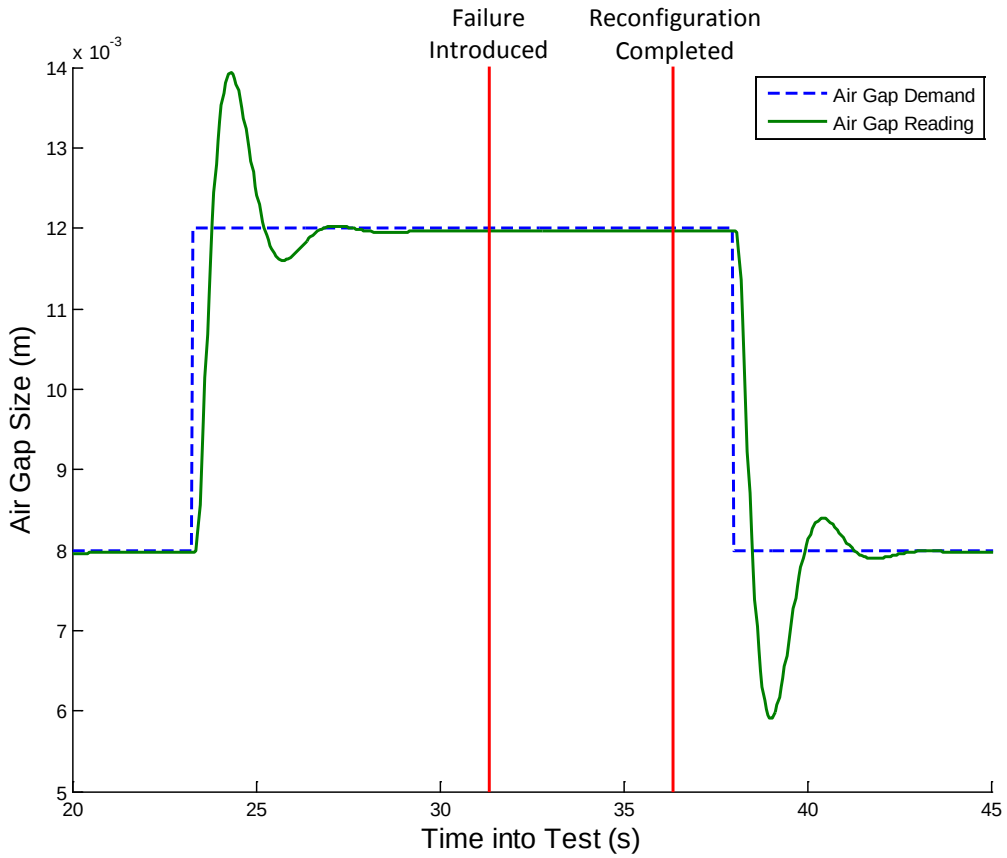


Figure 6-11 Airgap response during reconfiguration activity

As with the application failure, it can be seen the airgap response is unaffected by the systems management activities occurring. Again, the time taken to from the introduction of failure to a completed reconfiguration is 5 seconds.

6.3. System Response to a Series of Faults

This section shows how the IMA structure degrades gracefully in the presence of occurring faults in order to make best use of the available resources. The performance of the system during and after the introduction of each fault is monitored. Unlike the experiments in Section 7.2, the system remains in its resultant condition to allow a sequential failure to be

introduced. This will demonstrate how the IMA will gracefully degrade from full redundancy to lower levels of redundancy whilst maintaining service.

6.3.1. Failure 1 of 3: Application Failure of gap_control_2

The first failure in this series is an application failure of one of the gap control applications. The assumptions about this failure mode are the same as those described in Section 6.2.1.1. The application failed is shown in Figure 6-4.

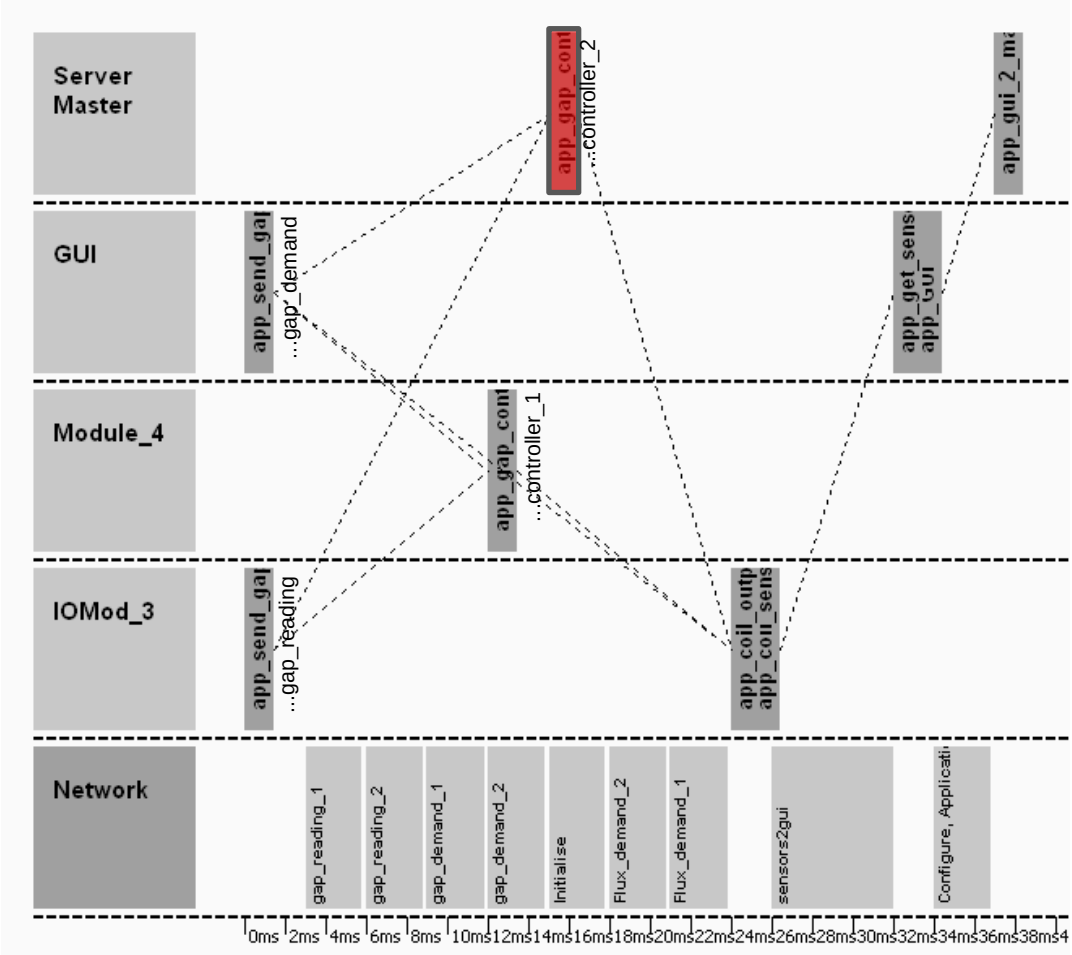


Figure 6-12 Application failure introduced to 'gap_controller_2'

Following recognition of this failure, the systems manager reconfigures in a manner similar to that described in section 6.2.1.2. This is done by removing the Server Master module as an option to place the application gap_controller_2 by amending the assignment criteria accordingly (Table 6-3). The resultant configuration is shown in Figure 6-13.

Assignment Criteria for app_gap_controller_2	
Before failure:	**After failure :**
!GUI	!GUI
!app_gap_controller_1	!app_gap_controller_1
	!Server_Master

Table 6-3 Change in assignment criteria as a result of failure

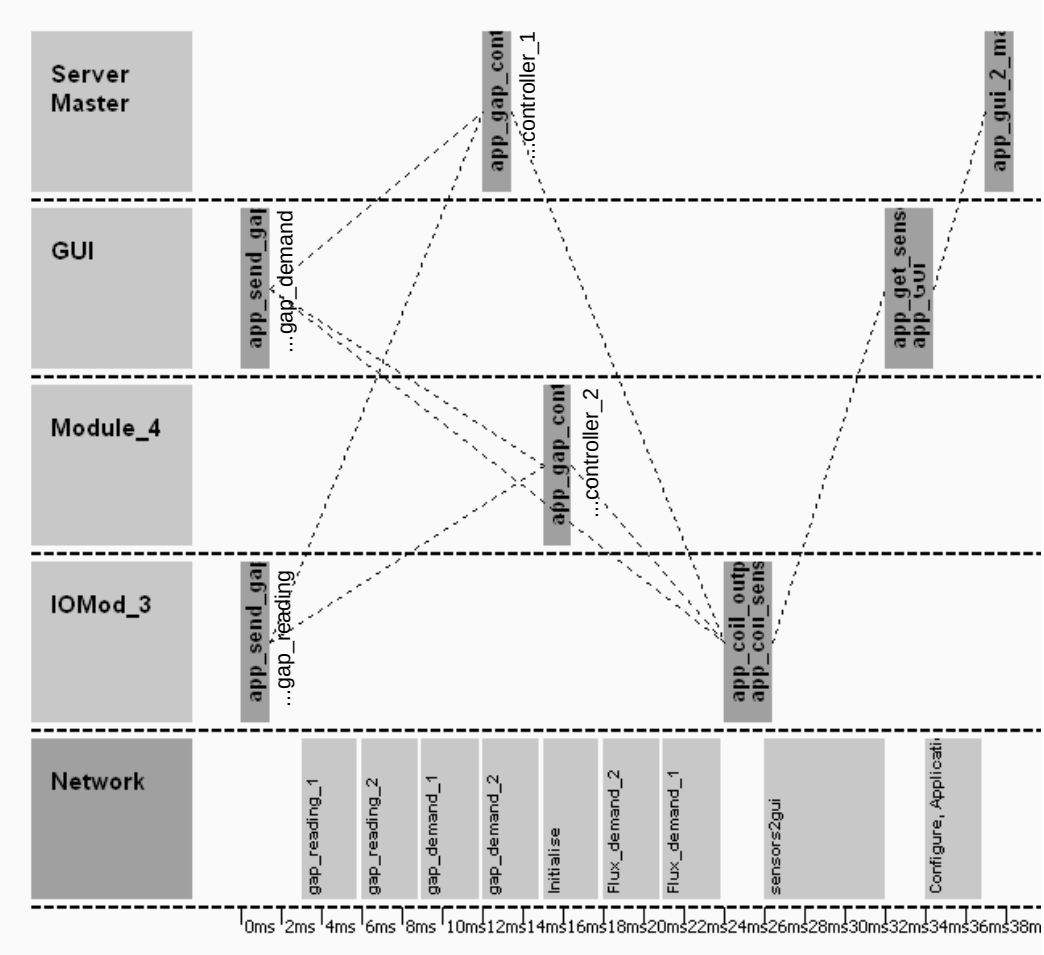


Figure 6-13 Reconfiguration following the failure of the gap_control_2 application

The solution to the new configuration problem is to swap the locations of the two gap control applications. This is a valid response to the software failure as:

- Assuming no hardware failure, it should be possible to run other similar applications on the same module

- A single point failure in software design could be projected by 'dissimilar programming'

Because dissimilar processing methods were not adopted in this case, the resolution here is almost as if gap_controller_2 was simply stopped and re-started.

6.3.2. Failure 2 of 3: Module Failure of Module_4

The second failure introduced into this series is the failure of Module_4.

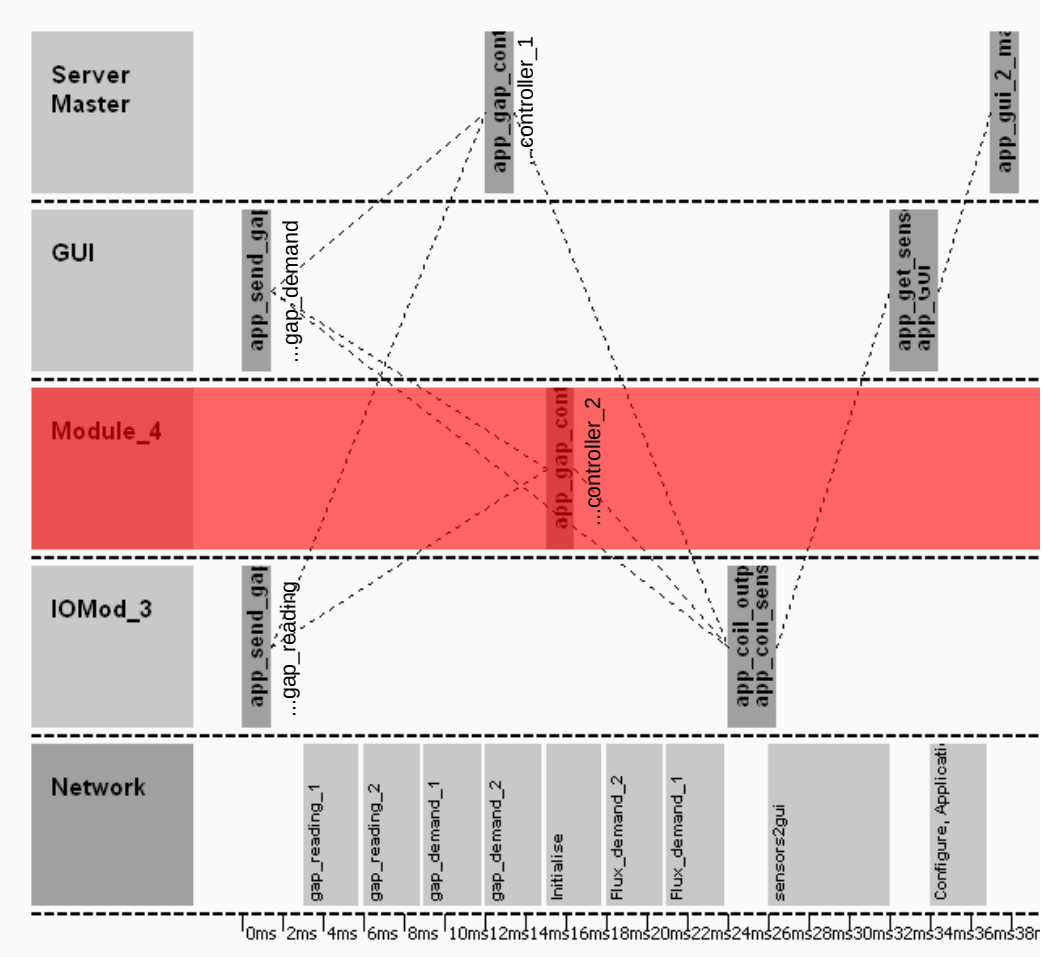


Figure 6-14 Failure of module_4

Following the introduction of this failure, the system must find a new configuration that will avoid placing functions on the failed module, and avoid replacing gap_controller_2 in the place it originally failed. As in Section 6.2.2, the modules available to assign resources are updated within the Systems Manager before reconfiguration is executed (Table 6-4). The resulting configuration is shown in Figure 6-15.

Available Modules on the Network	
Before failure:	**After failure :**
Server_Master	Server_Master
GUI	GUI
Module_4	IOMod_3
IOMod_3	

Table 6-4 Change in available modules as a result of failure

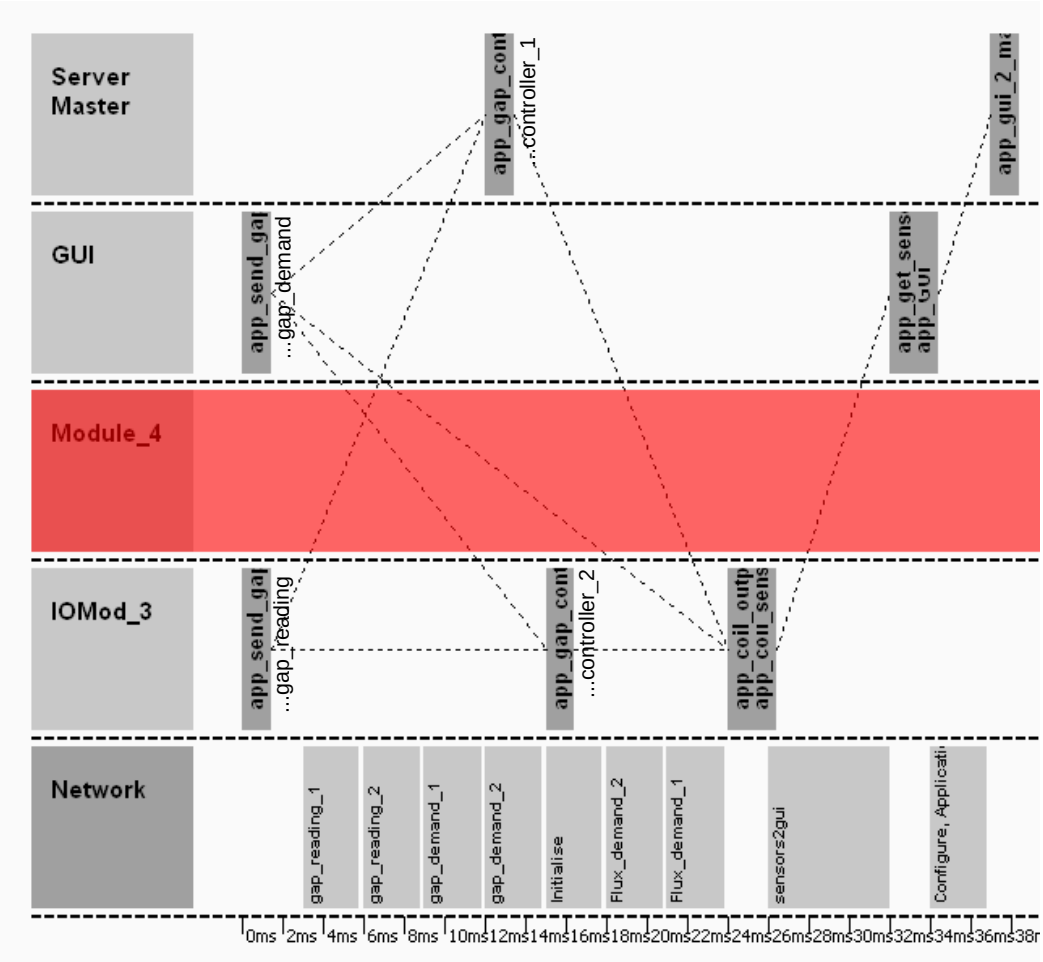


Figure 6-15 Configuration as a result of failing module_4

6.3.3. Fault 3 of 3: Application Failure of gap_controller_1

The final failure introduced is to fail gap_controller_1.

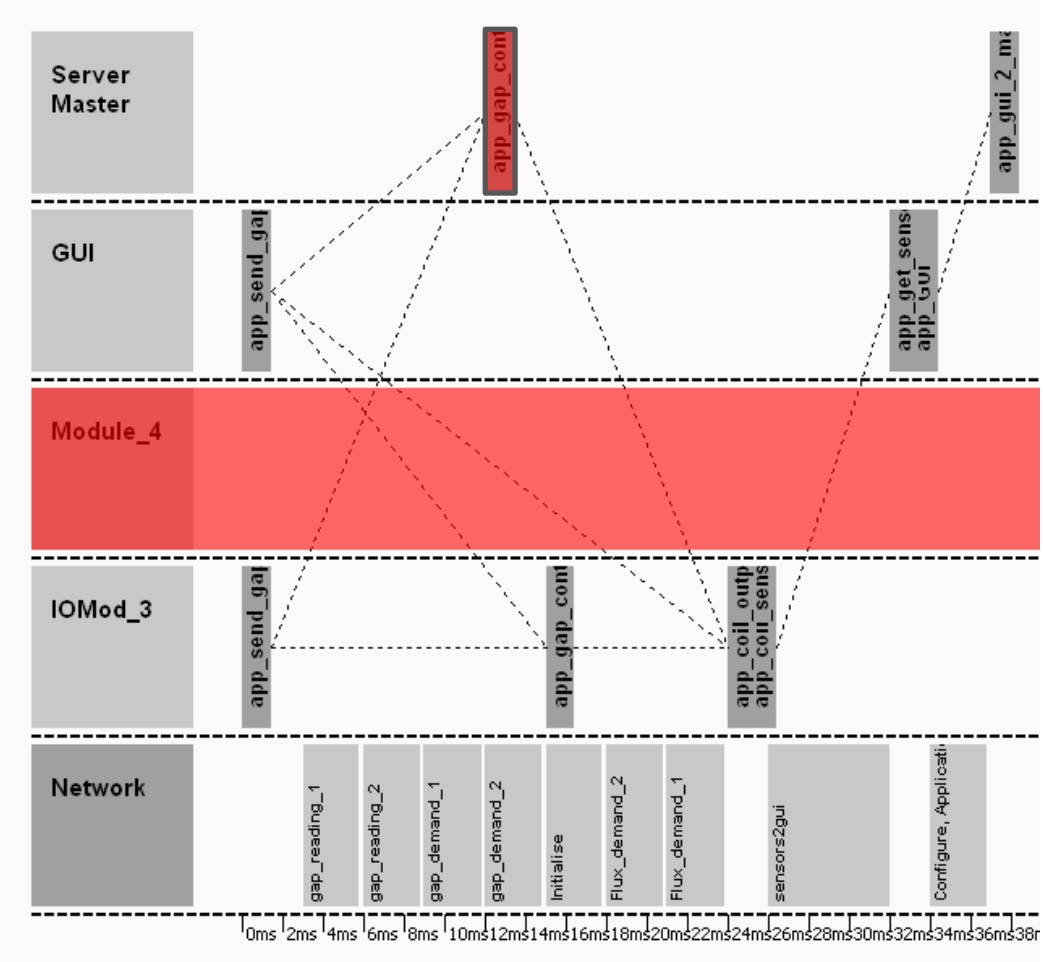


Figure 6-16 Failure of gap_controller_1

Once again, the assignment criteria is adjusted to note this failure, as shown in Table 6-5:

Assignment Criteria for app_gap_controller_1	
Before failure:	**After failure :**
!GUI	!GUI
!app_gap_controller_2	!app_gap_controller_2
	!Server_Master

Table 6-5 Change in assignment criteria as a result of failure

This time, when the reconfiguration algorithm is executed, an error is returned. It is now unable to find a new solution to the configuration problem that accommodates all the

initial assignment criteria, and the new requirements imposed by failures occurring. The Server master therefore leaves the system in this reduced redundant mode.

Despite three failures occurring, the overall effect on the function network is shown in Figure 6-17

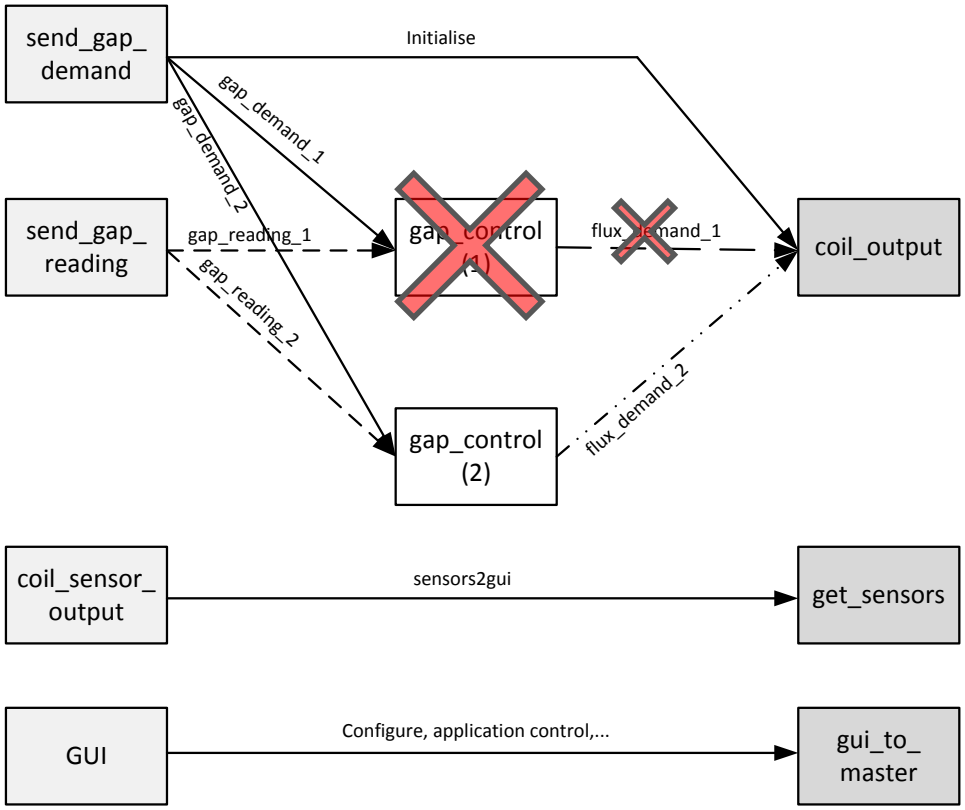


Figure 6-17 Effect on functional network of three failures

However, at this point the system can tolerate no more failures in the gap_control applications.

6.3.4. Evaluation of Series of Failures

The effect of the series of failures can be seen on the time response of the simulated airgap as shown in Figure 6-18.

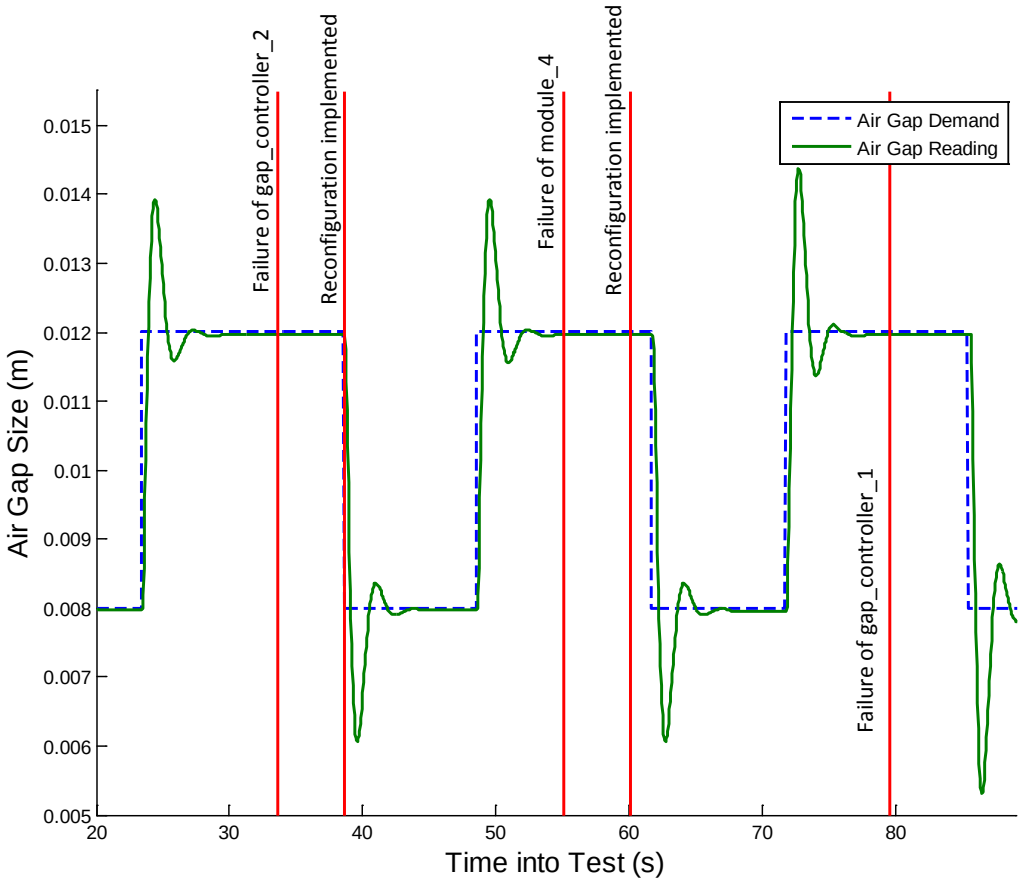


Figure 6-18 Time response of airgap during series failures

Once again, it is seen that the systems management activities are occurring with minimal influence on the time response of the airgap as deterministic performance of the control loop is maintained. Close observation reveals a slight degradation in response of the airgap following the second reconfiguration by the increased amount of overshoot. This occurs as there is a subtle difference in the position in the time frame at which activities happen, but does not cause significant reduction of the designed controller performance.

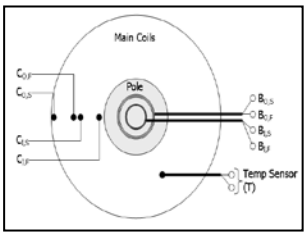
6.4. Summary

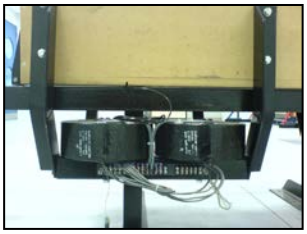
It has been demonstrated that following the introduction of a failure to the IMA, the system can reconfigure to re-obtain higher levels of redundancy. Alongside this, it has been shown that with careful management, the real time performance of a system can be maintained during these activities.

From failure identification to the implementation of a new configuration took five seconds. During this transition stage the system tolerates the fault by continuing to operate with the so called 'static redundancy' (i.e. multiplexed processing channels) designed into the system. The importance of this design feature is observed in the time response graphs in that no disruption to service occurs. If these redundant channels do not exist then there would a race between catastrophic failure and ability to initiate a new configuration. By accepting these lower levels of redundancy for a small amount of prescribed time, a sensible configuration can be realised.

During this investigation, an interesting point arose regarding the type of controller implemented in this solution. Here, a phase advance (PA) controller is used for control. This type of controller is not highly dependent on the states of the internal variables. By this, if the system is in a steady state condition and an internal variable reset occurs, it is unlikely that any perturbation of system response would be witnessed. If the controller used was an integral based controller the situation would be different. The value of the integral is stored within internal variables meaning that reset of these would result in an instant change in controller output. In the above examples, the controllers are stopped and restarted upon reconfiguration. If integral action is part of the controller then it is possible that sharing internal variables would be required upon reconfiguration to prevent a disruption to the provision of service. Otherwise transient oscillatory behaviour may be observed as a result of the implementation of a new configuration.

Within this demonstration, three failures with specific failures modes have been selected that allow graceful degradation of the system to occur. The limitation of hardware prevented full redundancy being designed into all parts of the system (such as dual inputs/sensing/actuation) which would have allowed different series failure tests to occur and a broader demonstration of redundancy. However, the demonstration summarised here has shown that with correct initial systems design and the use of reconfiguration to exploit flexibility in available resources, higher levels of redundancy can be re-established following occurrences of failure.





CHAPTER 7:

Conclusion

7. Conclusion

This chapter reviews the methods and results presented in the thesis and assess them against the original objectives, as set out in Chapter 1, and highlights how the objectives have been achieved and the proposed contributions have been demonstrated. Finally, recommendations are made for the future work in the continuation in this line of study.

7.1. Assessment of Objectives

In Chapter 1, section 1.4, a series of objectives were identified. The intention is that the satisfaction of these objectives would enable the demonstration of the research contributions highlighted in section 1.3. Here the objectives have been re-stated with reference to the evidence that supports their conclusion.

Objective 1: *Investigate relevant literature to determine current state-of-the-art in areas such as; IMA implementation, Fault management strategies within IMA, and methods for assigning avionic functionality to an IMA hardware installation.*

Chapter 2 summarised the key findings from a literature search regarding topics appropriate to the research. It was found that many IMA installations were seeking to adopt a multi-static approach, whereby a series of validated combinations of software assignment to IMA hardware configuration are stored as system blueprints. These configurations could then be reverted should the operational need occur. This method introduces a number of significant operational savings (such as commonality of parts, limited ability to reconfigure to optimise redundancy levels) but could not match the expected savings or extended operating periods of a dynamically reconfigurable arrangement.

It was identified that there is a lack of evidence of dynamically reconfigurable IMA systems in operation. The following reasons have been identified as a cause:

- A lack of certification route to implementation due to the possibility of configurations being implemented at run-time that have to been specifically validated.
- Dynamic configuration is considered a 'marginal gain' over the operational benefits available for a multi-static solution
- The Increase in autonomy assigned to the IMA system is risky in that a configuration arrangement may be assigned that does not satisfy fundamental operational needs.

Furthermore, it was highlighted that there is little research in how dynamically reconfigurable systems could be used to manage the occurrence of faults in new and novel ways. It is this understanding that would allow IMA installations to dramatically extend operational periods between scheduled maintenance by tolerating faults via a graceful degradation of system availability. However, conclusive research into how these mechanisms could be implemented is lacking.

Objective 2: *Develop a method of automatic configuration/reconfiguration of required avionic applications to distributed process resources*

Chapter 4 discusses the method developed to automatically generate a system configuration where a function set of software applications is logically assigned to an IMA installation. This method considers hardware compatibility, hardware availability and temporal constraints when assigning functions. This iterative process ensures that software elements are configured such that static redundancy levels are maintained to a high standard, and ensures events run concurrently in order to maintain real-time execution constraints.

Objective 3: *Design and develop a hardware test bed (i.e. a representative IMA system with appropriate 'middleware' and real-time communication strategy) to enable implementation and test of the automatic configuration/reconfiguration method.*

Chapter 3 of this thesis identified the necessary requirements of a representative IMA system that would be constructed for testing new methods of IMA systems management. This chapter considered not only the functional requirements necessary to accommodate an IMA style arrangement, but also extended these requirements to ensure that appropriate observation and data capture could be undertaken in order to validate results.

The requirements were broken down into the IMA sub-functions in order to define how each element should behave for the purposes of future testing. Definitions extended to hardware requirements (such as processing arrangement and communications needs), and software requirements (such as configuration management, communication protocols and fault management).

The development of the test installation based on the requirement set is described in Chapter 4. Here, it is detailed how the required functional IMA components are represented in the laboratory environment by the use of COTS components. A description of the general IMA software implementation is also described, including the use of basic systems management techniques to manage communication and synchronisation of the modules.

Objective 4: *Validate that the IMA implementation is capable of facilitating distributed real-time control operations with a range of functional topologies*

The results for assessing this objective are contained in Chapter 5. In this case, a series of conceptual networks of software functions were defined. For each functional network, the algorithm was tasked with automatically assigning the software applications to a defined IMA arrangement. A graphical output generated from these offline tests showed that the configuration algorithm successfully found an arrangement for these systems. It was shown that for these configurations, redundant processing channels were preserved and applications were placed temporally such that concurrency of events was maintained.

Further tests contained in Chapter 5 show how the configuration method could accommodate multiple networks of functions, which often have different levels of criticality associated with them. The test conducted in Chapter 5.2.2 shows how a real-time distributed control algorithm was implemented to manage the airgap of the maglev rig. The functional set also included applications required for user input, data reporting and systems management activities.

Objective 5: *Verify the ability of the configuration method to appropriately assign avionic applications to available resources using a range of functional topologies and available modules.*

Although the results discussed above showed that a configuration could be obtained to satisfy operational requirements, it was important to test the efficacy of the implementation to operate repeatedly and consistently, particularly with regards to the communication methodology. As described in Chapter 5, a test was defined where a 'token' was passed through a long series of applications and systems wide timing data was captured and assessed. This allowed a clear analysis of the effectiveness of the communication mechanism, inclusive of timing details and the level of repeatability.

Section 5.2.2 shows the effectiveness of distributed, real-time operations by the demonstration of an air-gap controller. Here, parts of the 'gap control' algorithm have been distributed across the IMA system, such that the whole system must perform as expected in order to maintain stability of an otherwise unstable system. It was shown that the controller performed as expected and real-time, distributed control occurred in a repeatable manner.

Objective 6: *Verify the ability of the IMA system to reconfigure a distributed real-time control functional set (following a fault injection) to an allocation that tolerates component failure.*

Chapter 6 undertook a series of tests where failures were injected into an IMA installation during operation. By collating management data from across the system, it was possible to trace and observe how the failure occurrence was managed. It was shown that service provision (i.e. real-time control of the maglev airgap) was maintained at all stages of the fault management process, including the implementation of a new configuration.

The second part of this assessment, also detailed in Chapter 6, showed that a series of failures could be introduced to the system and service provision can be maintained to an extent. This section showed how the system gracefully degraded from high levels of redundancy to minimal levels. The use of reconfiguration extended the possible operating period before maintenance would be required or the current mission had to be aborted. The results showed that during the fault management processes, service provision was maintained at all times.

One finding from this task was that during reconfiguration, a subtle change in the temporal arrangement of activities occurred. This had the effect of changing the closed loop controller performance by a small amount. Although the system operated well within sensible boundaries, it highlighted the importance of ensuring that real-time activities are respected not only in terms of processing, but in the system wide implementation.

7.2. Assessment of Contributions

In Chapter 1 of this thesis, a number of research contributions were stated. This section intends to identify how each of these contributions have been addressed.

Contribution 1: *A solution for the dynamic reconfiguration problem for assigning required systems functions (namely a distributed, real-time control function with redundant processing channels) to available computing resources whilst protecting the functional concurrency and time critical needs of the control actions.*

When conducting the literature search for the purpose of this research, it was found that a key area of knowledge that lacked completeness was a method by which an IMA configuration could be generated at run-time. It was evident from the literature that the configuration of applications is more than just a 'spatial' problem in that the order of events occurring system wide must be considered for maintaining consistency with time-critical tasks.

The reconfiguration method presented in this thesis, as discussed in Chapter 4, was developed to investigate how systems configurations could be generated at run-time. In doing so, considerations were not only made regarding matching an avionic function to an appropriate resource, but maintaining awareness of reliability conditions and time-critical execution requirements. The method developed was tested in Chapter 5 using a series of functional networks and the results showed how the systems arranged both 'spatially' and 'temporally'. This progresses previous work where such configurations are generated off-line only.

Contribution 2: *A systems management strategy that utilises the dynamic reconfiguration properties of an IMA to restore high levels of redundancy in the presence of failures.*

The main potential benefits of dynamic reconfiguration are the ability to extend mission capability in the presence of failures occurring, and to extend maintenance free operating

periods by continuing to provide service at sufficient levels of reliability by 'gracefully degrading'. Graceful degradation is the process whereby the system begins with high levels of redundancy then constantly reconfigures to make best use of remaining hardware as failures occur. These processes are only viable if the re-configuration of the system is managed automatically and is not a constant concern to operational staff.

This thesis aims to extend work undertaken to define Fault Management strategies within IMA, specifically how reconfiguration can be used to maintain service in the presence of faults. The method presented shows how health monitoring can be used to provide up-to-date information about system availability. This information can then be used to inform an automated systems manager to assess if reconfiguration would be beneficial. If so, a new configuration can be calculate and initiated using remaining available resource.

The solution to using reconfiguration is presented in Chapter 5, and required a consideration of the wider systems requirements to solve. Fundamentally, the system is tasked to provide a service in a time-critical manner that cannot be interrupted whilst a new configuration is being calculated. The method proposed relied on good systems design such that at the instant of failure, service is continued by traditional redundant channels. The purpose of dynamic reconfiguration is not to respond instantly as a fault handling method, but to operate at a higher level in order make the system safer for when the next failure occurs.

In order to prove the efficacy of the above contributions, the methods were tested on the representative IMA installation that was described in Chapters 3 and 4. The testing, detailed in Chapters 5 and 6, was designed to show the operational benefits and difficulties of dynamic reconfiguration in a simplified capacity.

The key test was detailed in Chapter 6. Here, the functional requirement of the system was to provide high reliability control calculations for a Maglev vehicle. As such, a series of IMA functions were designed with the control law calculations for the air-gap duplicated for redundancy. The system was tasked with generating not only the initial systems

configuration, but was to reconfigure the arrangement of functions should the health of systems component change. A series of simulated failures was injected into the system and the resultant configurations were recorded and observed. It was shown that not only was service continually provided, but also that this system would degrade gracefully by reconfiguring to make use of available redundant processing capacity. This was done by gathering and analysing real-time data collected from across the system components.

7.3. Future Work and Recommendations

Although this work demonstrates a method by which automatic configuration within an IMA could be used, there remains many areas that require further research and clarification. The following subsections highlight and discuss some of these.

7.3.1. Identified Areas of Good Practice for Reconfigurable IMA

Some benefits of reconfiguration were highlighted in the discussion to Contribution 2 of this thesis. During investigation, some areas were noted that would be positive aspects to remain in consideration during any subsequent design and implementation of a reconfigurable system.

The design process adopted here relied in an assumption that the initial design of the applications and modules allowed a configuration to be found. This initial configuration was designed to have a degree of static redundancy. This fundamental decision creates a number of key safety features. At the point of failure, the system would retain service provision should all partitions be robust such that the failure does not propagate in unexpected ways. Secondly, this creates a time window for a better configuration to be found. This window could be seconds, minutes or hours (dependant on the resultant levels of reliability following the failure) meaning that time can be taken to derive a new sensible configuration without attempting to achieve and implement this within a single communication time frame window. Finally, if a new system configuration is not possible due to lack of redundant processing resource then the system is still in a 'safe' state. Again, using an assessment of the remaining reliability levels, an informed decision can be made about the continuation of the mission. A recommendation here is that methods of automatically generating fault effects analysis are applied here and included in the decision making process. This would also enable systems designers to decide on the required degree of redundancy to provide similar safety levels to current standards. For example, could a triplex system now replace a quadraplex system due to the possibility of reconfiguration?

7.3.2. Certification

A large amount of research is already underway in the field of certifying IMA. A promising solution to simplify this method is the use of 'modular certification techniques' where the system is not directly certified as a whole, but parts of the system can be considered independently. This means a re-use of a part of a system does not require recertification of the system whole, saving time and reducing the cost of multiple systems configuration definitions.

This method would require extension for use with dynamic systems. Here, the configuration would have to be highly automated to allow on-line 'approval' of the proposed configuration generated due to a change in operational circumstance. Current thinking in the way aircraft systems are built and certified are not appropriate for this type of solution. Nevertheless, some method is required to confirm that the configuration proposed by the automated system is safe to implement on the platform, and provide an increase in operational benefits.

7.3.3. Quantification of Potential Savings

One of the proofs outstanding from this research is the quantification of the potential gains of a dynamically reconfigurable IMA over a multi-static solution. It has been shown that dynamic reconfiguration is possible and redundant processing channels can be restored by use of spare system capacity. However, a thorough study is required in order to balance the risk of undertaking this activity with the benefits gained. At this stage it is hard to measure how 'risky' reconfiguration is. A thorough analysis into the potential hazards is required, along with suggested strategies for mitigating them. This must be weighed against the cost benefit of operating an IMA with this capability, such as the potential weight saving by carrying a reduced set of avionic hardware. Understanding this metric will help inform system design in terms of choosing the right number (of the correct type) of processing components to facilitate the functional needs.

7.3.4. Controller Lag Approximations

It was shown that by reconfiguring the communication network, the exact time at which data is sent within an overall time frame can never be known. Although it will not 'jitter' around the time frame when operating in a set configuration, there is the potential to subtly adjust the response characteristics by subtle changes in the communication criteria.

As long as good control law design is undertaken and the sample rate of the overall time frame is suitably fast, it is unlikely that the variations in control performance would be noticed. The possible variations in data communication should be considered at the point of control design such that design limits are adhered to in all possible circumstance.

7.3.5. Configuration Optimisation

Chapter 4 discussed briefly that although the configuration algorithm developed generated a useable functional assignment, there is still room to optimise placement to maximise system resources. Improvement here would directly lead to weight savings within the avionics package as fewer processing modules would be required to meet the avionics needs.

The method of generating resource assignment could readily be changed to an optional method which also uses up-to-date system availability information and assignment constraints to allocate a network of avionic functions. However, they should all retain the fundamental requirements of the allocation problem in that functions require placing not only on a resource, but in an allotted period within the time frame of time in order to ensure concurrency of the execution stream.

v. References

ARINC, 2006a. 651-1 Design Guidance for Integrated Modular Avionics.

ARINC, 2006b. 653P0-1 Avionics Application Software Standard Interface, Part 0.

Avizienis, A., Laprie, J.-C. & Randell, B., 2000. Fundamental Concepts of Reliability. In *Third Information Survivability Workshop*.

Bisson, K. & Troshynski, T., 2003. Switched Ethernet testing for avionics applications. *AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference. Proceedings*, pp.546–550.

Burns, A. & Wellings, A.J., 2001. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*, Addison Wesley.

Charara, H. & Fraboul, C., 2005. Modelling and simulation of an avionics full duplex switched Ethernet. *Telecommunications, 2005. Advanced Industrial Conference on Telecommunications/service Assurance with Partial and Intermittent Resources Conference/e-learning on Telecommunications Workshop. aict/sapir/elete 2005. Proceedings*, pp.207–212.

Collinson, R.P.G., 2011. *Introduction to avionics systems*, Springer.

Conmy, P. & McDermid, J., 2001. High level failure analysis for Integrated Modular Avionics. In *Proceedings of the Sixth Australian workshop on Safety critical systems and software - Volume 3*. SCS '01. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., pp. 13–21. Available at: http://dl.acm.org/citation.cfm?id=563780.563783.

Conmy, P.M., 2006. Safety Analysis of Computer Resource Management Software. *University of York Department of Computer Science-Publications-YCST*, 7.

Coutinho, R.M.A., 2008. Aspects on Architecture for Independent Distributed Avionics (AIDA). *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.A.1–1–1.A.1–9.

Dajiang, S., Jinxia, A. & Jihong, Z., 2011. A new approach to improve safety of reconfiguration in Integrated Modular Avionics. In *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*. pp. 1C4–1–1C4–12.

Dedieu, C. & Loffler, E., 2000. *Modular Avionics Upgrade: The Cost Effective Solution to Adapt Existing Fighters to the Operational Requirements of Today's Battlefield*, DTIC Document.

Deng, Z. & Liu, J.W.S., 1997. Scheduling real-time applications in an open environment. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*. pp. 308–319.

Edwards, R.A., 1997. Exploiting the Potential of Integrated Modular Avionics. In *Proc. of ERA Avionics Conference and Exhibition*.

Ellis, S.M., 1997. Dynamic software reconfiguration for fault-tolerant real-time avionic systems. *Microprocessors and Microsystems*, 21(1), pp.29–39. Available at: http://www.sciencedirect.com/science/article/pii/S0141933197000173.

Elmqvist, J. et al., 2008. Demonstration of a formal method for incremental qualification of IMA systems. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.5.D.3–1–5.D.3–8.

Elston, J., Argrow, B. & Frew, E., 2005. A distributed avionics package for small uavs. *AIAA Infotech@ Aerospace*, pp.733–742.

Evans, A., 2003. Model Driven Development - Fact or Fiction?

Field, D. et al., 1997. Software techniques for IMA-system management, system security and blueprints. In *Proc. of ERA Avionics Conference and Exhibition*.

Ford, B. et al., 2009. Adaptive architectures for future highly dependable, real time systems. In *Conference on Systems Engineering Research CSER2009*. Loughborough, UK: Research School of Systems Engineering, Loughborough University. Available at: https://dspace.lboro.ac.uk/2134/11751.

Gaska, T., Watkin, C. & Chen, Y., 2015. Integrated Modular Avionics ??? Past, present, and future. *Aerospace and Electronic Systems Magazine, IEEE*, 30(9), pp.12–23.

Goodall, R., 1985. The Theory of Electromagnetic Levitation. *Physics in Technology*, 16, pp.207–213.

Goodall, R.M., 2004. Dynamics and control requirements for ems maglev suspensions. In *Proceedings of the 18th International Conference on Magnetically Levitated Systems and Linear Drives*. Shangai: {\copyright} Roger M. Goodall.

Goodall, R.M., 2000. On the robustness of flux feedback control for electro-magnetic Maglev controllerso Title. In *Proceedings of the 16th International Conference on Magnetically-Levitated Systems and Linear Drives*. Rio de Janeiro.

Grigg, A. et al., 1999. A Method for Design and Analysis of Next Generation Aircraft Computer Systems. In *Proc. of INCOSE Conference on Systems Engineering*. Citeseer.

Grigg, A., 2002. *Reservation-Based Timing Analysis (A Partitioned Timing Analysis Model for Distributed Real-Time Systems)*. York.

Hampp, J., 2007. Industry Progress Towards an IMA Software Solution.

Hladik, P.-E. et al., 2008. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1), pp.132–149. Available at: http://www.sciencedirect.com/science/article/pii/S0164121207000672.

Hollow, P., McDermid, J. & Nicholson, M., 2000. Approaches to certification of reconfigurable IMA systems. In *Proceedings 10th International Symposium of the International Coucil on Systems Engineering*.

Johnson, D.M. & Omiecinski, T.A., 1998. The feasibility and benefits of dynamic reconfiguration in integrated modular avionics. *Aeronautical Journal*, 102(1012), pp.99–105. Available at: http://cat.inist.fr/?aModele=afficheN&cpsidt=2183299 [Accessed February 11, 2013].

Jolliffe, G., 2005. Producing a safety case for IMA blueprints. *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, 2, p.14 pp. Vol. 2.

Kim, J.-E. et al., 2014. Integrated Modular Avionics (IMA) Partition Scheduling with Conflict-Free I/O for Multicore Avionics Systems. *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pp.321–331.

Kopetz, H., 2011. *Real-time systems : design principles for distributed embedded applications*, New York: Springer.

Kopetz, H. et al., 2005. The time-triggered Ethernet (TTE) design. *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pp.22–33.

Lee, Y.H. et al., 2000. Scheduling tool and algorithm for integrated modular avionics systems. In *Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th*. pp. 1C2/1–1C2/8 vol.1.

Lee, Y.-H. et al., 2000. Resource scheduling in dependable integrated modular avionics. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*. pp. 14–23.

Linling, S., Wenjin, Z. & Kelly, T., 2011. Do safety cases have a role in aircraft certification? *Procedia Engineering*, 17, pp.358–368. Available at: http://www.sciencedirect.com/science/article/pii/S1877705811027202.

Lisagor, O., Sun, L. & Kelly, T., 2010. The illusion of method: challenges of model-based safety assessment. In *28th International System Safety Conference (ISSC)[submitted to]*. Citeseer.

Little, R., 1991. Advanced avionics for military needs. *Computing & Control Engineering Journal*, 2(1), pp.29–34.

Liu, B., 2016. Approach for integrated modular avionics reconfiguration modelling and reliability analysis based on AADL. *IET Software*, 10(1), pp.18–25(7). Available at: http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2014.0179.

Lopez, J. et al., 2008. Modular avionics for seamless reconfigurable UAS missions. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.A.3–1–1.A.3–10.

Mapleston, M., 2006a. Fault Management in Advanced Architectures.

Mapleston, M., 2006b. Fault Management in Integrated Modular Systems.

Mazuk, D., 2008. IMA Resource Allocation process. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.B.2–1–1.B.2–6.

Michail, K., 2009. *Optimised configuration of sensing elements for control and fault tolerance applied to an electro-magnetic suspension system*. Loughborough University. Available at: https://dspace.lboro.ac.uk/2134/5806.

Miller, S.P. et al., 2009. Implementing logical synchrony in integrated modular avionics. In *Digital Avionics Systems Conference, 2009. DASC '09. IEEE/AIAA 28th*. pp. 1.A.3–1–1.A.3–12.

MOD, 2005. *ASAAC Standards Part 1 Proposed Standards for Architecture*,

Moir, I. & Seabridge, A., 2008. *Aircraft Systems: Mechanical, Electrical and Avionics Subsystems Integration*, Wiley.

Moir, I., Seabridge, A. & Jukes, M., 2006. *Military avionics systems*, John Wiley & Sons.

Moir, I., Seabridge, A.G. & Jukes, M., 2003. *Civil avionics systems*, American Institute of Aeronautics and Astronautics Bury St. Edmunds,, UK.

Montano, G. & McDermid, J., 2008. Human involvement in dynamic reconfiguration of Integrated Modular Avionics. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*. pp. 4.A.2–1–4.A.2–13.

Moore, G.E., 2006. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff. *Solid-State Circuits Newsletter, IEEE*, 11(5), pp.33–35.

Morel, M., 2014. Model-based safety approach for early validation of integrated and modular avionics architectures. In *Model-Based Safety and Assessment*. Springer, pp. 57–69.

Morgan, M.J., 1991. Integrated modular avionics for next-generation commercial airplanes. In *Aerospace and Electronics Conference, 1991. NAECON 1991., Proceedings of the IEEE 1991 National*. pp. 43–49 vol.1.

O. Lisagor. J A McDermid, D.J.P., 2006. Towards a Practicable Process for Automated Safety Analysis. In *ISSC*.

Parkinson, P. & Kinnan, L., 2003. Safety-critical software development for integrated modular avionics. *Embedded System Engineering*, 11(7), pp.40–41.

Parr, G.R. & Edwards, R., 1999. Integrated modular avionics. *Air & Space Europe*, 1(2), pp.72–75. Available at: http://www.sciencedirect.com/science/article/pii/S1290095899800185.

Porcarelli, S. et al., 2003. An approach to manage reconfiguration in fault-tolerant distributed systems. In *Proceedings of the ICSE Workshop on Software Architecture for Dependable Systems*. pp. 71–76.

Prisaznuk, P.J., 2008. ARINC 653 role in Integrated Modular Avionics (IMA). In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*. pp. 1.E.5–1–1.E.5–10.

Prisaznuk, P.J., 1992. Integrated modular avionics. *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, pp.39–45 vol.1.

Randell, B. & Xu, J.I.E., The Evolution of the Recovery Block Concept. , pp.1–24.

RTCA, 2005. *DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*,

Salomon, U. & Reichel, R., 2011. Automatic safety computation for IMA systems. In *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*. pp. 7C3–1–7C3–9.

Salzwedel, H., Fischer, N. & Baumann, T., 2008. Aircraft level optimization of avionics architectures. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.B.4–1–1.B.4–7.

Schavey, T. & Duba, S., 2008. Streamlining IMA integration through model-driven methodologies. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.B.3–1–1.B.3–5.

Scottsdael, L.Y. & Glendale, J.T., 2002. Flight Control Module Merged into the Integrated Modular Avionics.

Si, C., Wang, S. & Liu, B., 2015. A model driven multi-constraint safety analysis method for integrated modular avionics systems on time domain. *Prognostics and System Health Management Conference (PHM), 2015*, pp.1–6.

Stephenson, Z., Nicholson, M. & McDermid, J., 2006. Flexibility and Mangeability of IMS Projects. In *ISSC*.

Strunk, E.A., Knight, J.C. & Aiello, M.A., 2004. Distributed reconfigurable avionics architectures. In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*. pp. 10.B.4–101–10 Vol.2.

Sutterfield, B., Hoschette, J.A. & Anton, P., 2008. Future integrated modular avionics for jet fighter mission computers. *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pp.1.A.4–1–1.A.4–11.

Tsai, W.T. et al., 2006. RTSOA: Real-Time Service-Oriented Architecture. *Service-Oriented System Engineering, 2006. SOSE '06. Second IEEE International Workshop*, pp.49–56.

Varshney, P.K., 1997. Multisensor data fusion. *Electronics & Communication Engineering Journal*, 9(6), pp.245–253.

Wake, A.S. et al., 1997. Modular avionics operating system — software concept. *Microprocessors and Microsystems*, 21(1), pp.63–68. Available at: http://www.sciencedirect.com/science/article/pii/S0141933197000215.

Wartel, F. et al., 2013. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, pp.241–248.

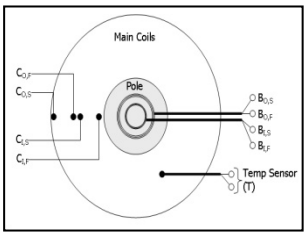
Watkins, C.B., 2006a. Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources. *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, pp.1–12.

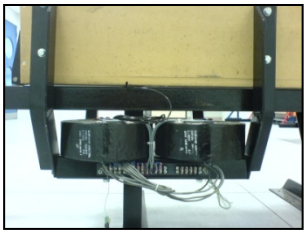
Watkins, C.B., 2006b. Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources. *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, pp.1–12.

Watkins, C.B. & Walter, R., 2007. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*. pp. 2.A.1–1–2.A.1–10.

Yann-Hang, L. et al., 2000. Resource scheduling in dependable integrated modular avionics. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*. pp. 14–23.

Zhang, J.-G., Pervez, A. & Sharma, A.B., 2003. Avionics data buses: an overview. *Aerospace and Electronic Systems Magazine, IEEE*, 18(2), pp.18–22.





APPENDIX A:

Maglev Controller Development

A. Maglev Controller Development

A.1. Mathematical Model of Maglev Rig

This section describes the process that was undertaken in order to produce a block diagram representation of the dynamics of the Maglev system. Electromagnetism, especially when involved in electromagnetic levitation, is a complicated process to model due to the many inherent non-linear relationships. In order to simplify the model, it is preferable to assume linear relationships around a normal operating point. A sensible nominal operating point is chosen, and a model is built up by considering simplified linear relationships of the important variables for small fluctuations around this point. This is very similar to the techniques in building up simplified equations for modelling complex aircraft characteristics for development of flight control systems.

The draw back to this modelling technique is that the further away from the normal operating point the system goes, the less accurate the model becomes, and the less reliable the derived control system is. The solution to the problem lies in producing a suitably robust control structure that will cope with these variations in the fundamental characteristics of the system. This can be verified by deriving a controller based on a central operating point, and then testing this controller against a range of expected operating conditions. At this stage it is important to consider the requirements to go from a state of rest, to a state of controller levitation where the airgap at the beginning of this state will be around 25mm, and will need to move to a steady gap of 10mm.

The four main variables of interest within a magnetic levitating system are:

- F - Force in Newtons
- B – Flux density in Teslas
- G – air gap in meters
- I – coil current in amps

We can assume that a 'normal', steady operating point exists. In an aircraft this could be defined as "steady, level flight", but unfortunately, no such description exists here. In this condition, let us assume that each of the above variables have a value of F_0, B_0, G_0 and I_0.

Therefore at any time of operation, it can be assumed that an instantaneous value of F, B, G and I can be stated as:

$$F = F_0 + f$$
$$B = B_0 + b$$
$$G = G_0 + g$$
$$I = I_0 + i$$

Where f, b, g and i all represent *small* variations around the operating point.

By now considering the relationships between the variables around the operating point, a model can be deduced.

It can be shown that for a constant gap:

$$B \propto I$$

Also, for a constant current,

$$B \propto \frac{1}{G}$$

The relationships of the small variations can be approximated by assuming that they are directly proportional and that the gradients can be approximated by using the normal operating point values. By doing this and combining the two relationships:

$$b = \frac{B_0}{I_0} i - \frac{B_0}{G_0} g \qquad \textit{(equation 1)}$$

It can also be shown that:

$$F \propto B^2$$

Therefore, the gradient at the operating point is can be shown to be:

$$2\frac{F_0}{B_0}$$

And so:

$$f = \frac{2F_0}{B_0} b \qquad \textit{(equation 2)}$$

If F_0 is the Force required to be exerted on the load to maintain 'steady level flight', then F_0 must be equal to the weight of the suspended load. Therefore, and vertical acceleration around the normal operating point can be expressed as:

$$\ddot{z} = \frac{f}{M} \qquad \textit{(equation 3)}$$

The small variation in air gap is given by the difference between the position of the vehicle and the distance from the normal operating point to the track:

$$g = z_t - z \qquad \textit{(equation 4)}$$

The variable z_t represents the position of the track. If it is desired to model the track as a disturbance, in order to model the natural non-uniform nature of the track, z_t can be modelled as a random fluctuation around zero. The characteristics of the distribution will be dependant on track quality. For the purposes of control development, this value is set to zero.

The next stage in this derivation is to define the electrical dynamics of the system. The voltage applied across the coils will be equal to the sum of the demand of the resistive and inductive components. Coils have the tendency to resist any voltage applied to them, due to electromagnetic inductance of the close wires. Rather than simply including leakage inductance, the principle of electromagnetic induction must be applied, as will be shown later:

$$v = 2Ri + 2L\frac{di}{dt} + 2NA\frac{db}{dt}$$

Where:

v is the small change of coil voltage from normal operating point

R is the coil resistance

N is the number of turns in the coil

A is the pole face area of the coil

Referring back to equation 1:

$$b = \frac{B_0}{I_0}i - \frac{B_0}{G_0}g$$

Let:

$$K_i = \frac{B_0}{I_0},\ K_g = \frac{B_0}{G_0}$$

So:

$$b = K_i i - K_g g$$

Differentiating both sides with respect to time:

$$\dot{b} = K_i \dot{i} - K_g \dot{g}$$

Substituting into the expression for v:

$$v = 2Ri + 2L\dot{i} + 2NA\left(K_i \dot{i} - K_g \dot{g}\right)$$

Re-arranging to make 'i' the subject:

$$2Ri + 2L\dot{i} + 2NAK_i \dot{i} = v + 2NAK_g \dot{g} \qquad \textit{(equation 5)}$$

This equation can be manipulated further for assistance in deriving the resultant block diagram:

$$e_1 = v + 2NAK_g \dot{g}$$

So:

$$2Ri + 2L\dot{i} + 2NAK_i \dot{i} = e_1$$

Apply Laplace transform:

$$2Ri + s(2L + 2NAK_i)i = e_1$$
$$i\left(2R + s(2L + 2NAK_i)\right) = e_1$$
$$i = \frac{1}{2R + 2(L + NAK_i)s} e_1$$

Letting $K_L = L + NAK_i$

$$i = \frac{1}{2R + 2K_L s} e_1$$

The equations derived above that are used to create the block diagram are:

1. $b = K_i i - K_g g$
2. $f = K_b b$
3. $\ddot{z} = \frac{f}{M}$
4. $g = z_t - z$

5. $2Ri + 2L\dot{i} + 2NAK_i\dot{i} = v + 2NAK_g\dot{g}$

Where:

$$K_i = \frac{B_0}{I_0},\ K_g = \frac{B_0}{G_0} \text{ and } K_b = \frac{2F_0}{B_0}$$

A.1.1. Block Diagram Representation

The above formulae can be represented in block diagram notation as shown in Figure A-1:

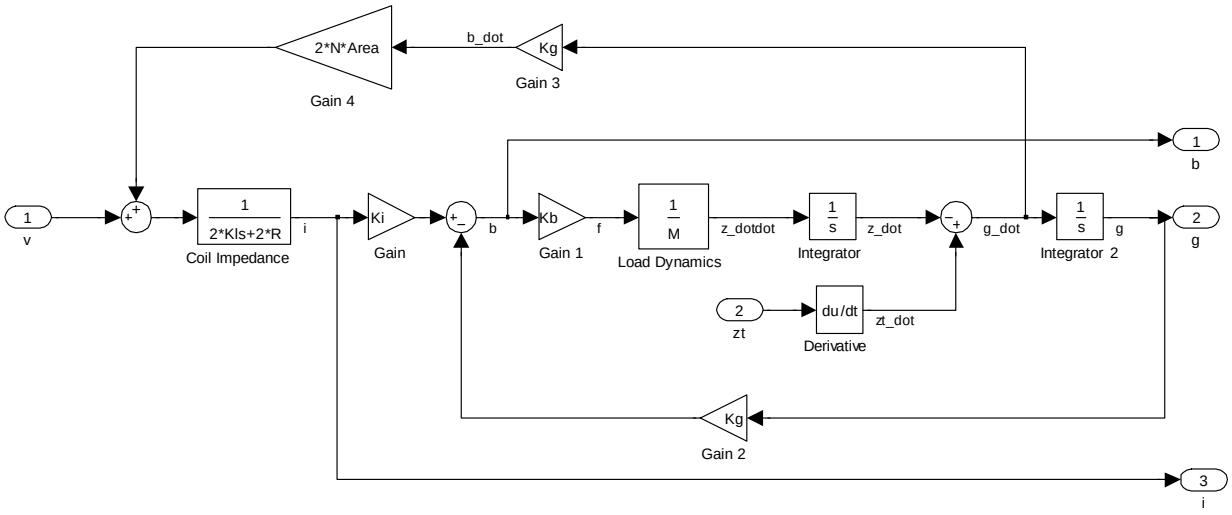


Figure A-1 Block Diagram of Magnet System

It can be seen that by including the principle of electromagnetic induction in equation 5, the changes in the air gap have an effect on the voltage applied to the coil. This relationship would have been neglected if only electrical induction was considered.

A.1.2. Transfer function derivation from block diagram

It is useful to express the above block diagram as a transfer function for simulation purposes later in the project. The transfer function was derived by a series of block diagram simplification steps as described below. The starting point is the block diagram of the magnet system as shown in Figure A-1. This can be simplified to the model shown in Figure A-2:

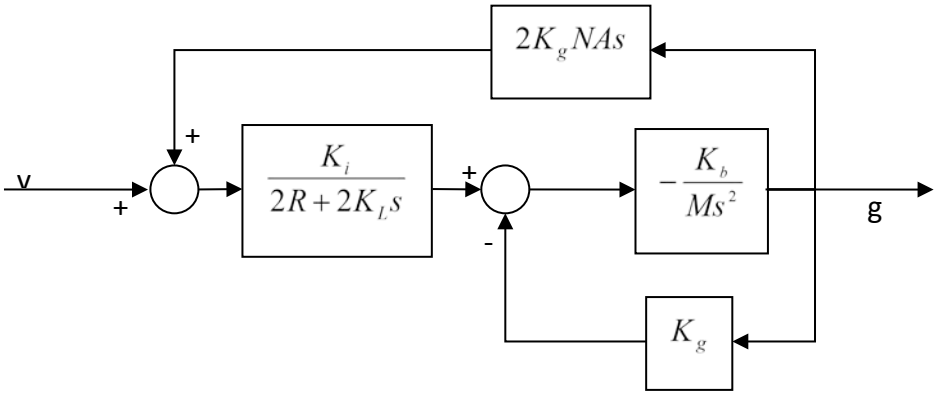


Figure A-2 Simplified Block diagram

Redrawing this model to make 'b' the obvious output:

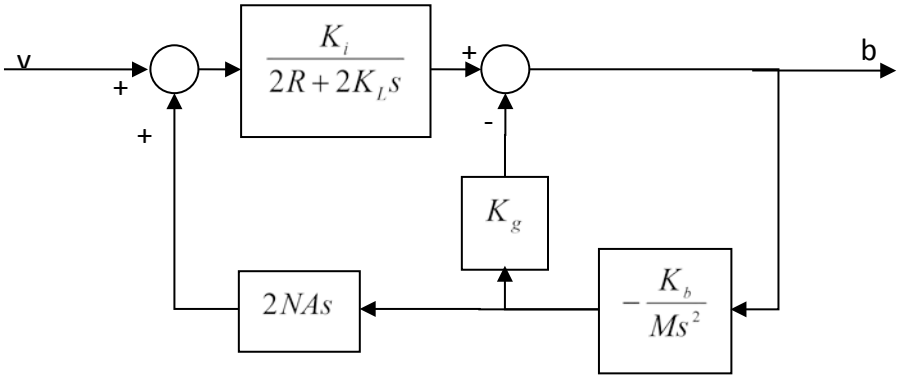


Figure A-3 Transfer function derivation 1

Moving the term $-\frac{K_b}{Ms^2}$ past the junction:

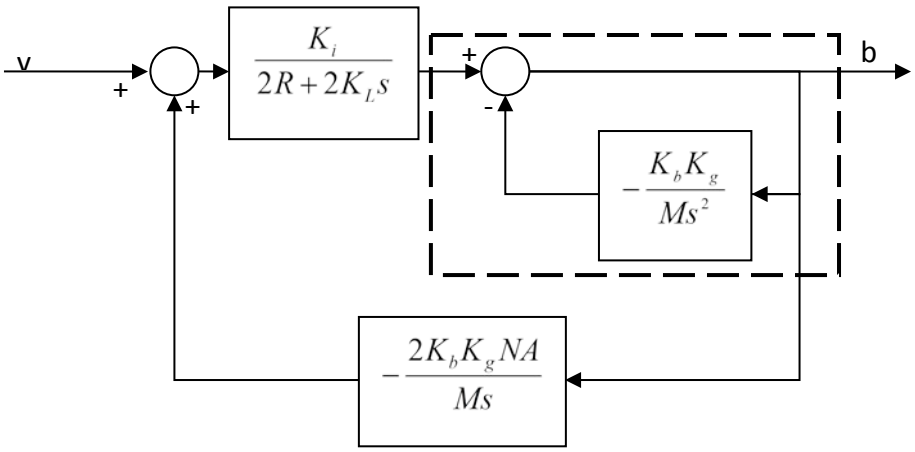


Figure A-4 Transfer function derivation 2

Closing the loop highlighted by the dashed box using a standard formula for a feedback system:

$$F(s) = \frac{G(s)}{1+GH(s)}$$
$$= \frac{1}{1-\frac{K_b K_g}{Ms^2}}$$
$$= \frac{Ms^2}{Ms^2 - K_b K_g}$$

Which yields the following block diagram:

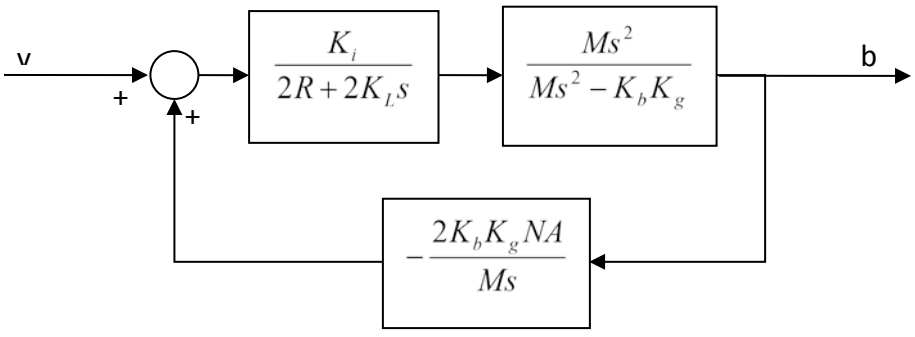


Figure A-5 Transfer function derivation 3

Which simplifies further to:

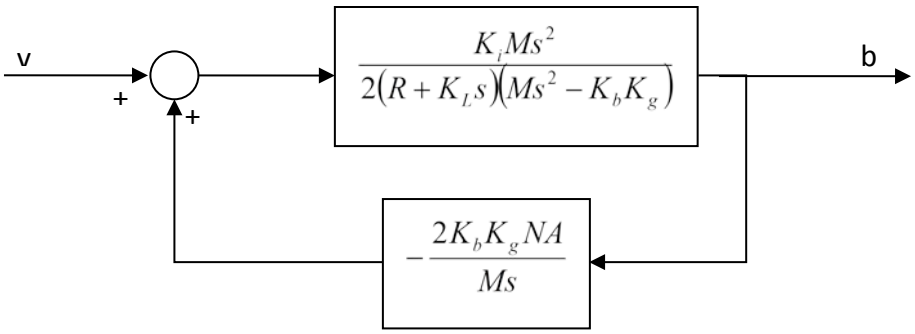


Figure A-6 Transfer function derivation 4

So the transfer function of b(s) over v(s) can be derived, again using a standard formula for a feedback system:

$$\frac{b(s)}{v(s)} = \frac{G}{1-GH}$$

$$= \frac{\dfrac{K_i Ms^2}{2(R+K_L s)(Ms^2-K_b K_g)}}{1+\dfrac{2K_i K_b K_g NAMs^2}{2Ms(R+K_L s)(Ms^2-K_b K_g)}}$$

$$= \frac{K_i Ms^2}{2(R+K_L s)(Ms^2-K_b K_g)+2K_i K_b K_g NAs}$$

Solving for the denominator:

$$2(R+K_L s)(Ms^2-K_b K_g)+2K_i K_b K_g NAs$$
$$= 2(K_L Ms^3+RMs^2-K_L K_b K_g s-RK_b K_g+K_i K_b K_g NAs)$$
$$= 2(K_L Ms^3+RMs^2+(K_i K_b K_g NA-K_L K_b K_g)s-RK_b K_g)$$
$$= 2K_L M\left(s^3+\frac{R}{K_L}s^2+\frac{K_b K_g}{M}\left(\frac{NAK_i}{K_L}-1\right)s-\frac{RK_b K_g}{K_L M}\right)$$

Thus, the transfer function becomes:

$$\frac{b(s)}{v(s)} = \frac{K_i Ms^2}{2K_L M\left(s^3+\dfrac{R}{K_L}s^2+\dfrac{K_b K_g}{M}\left(\dfrac{NAK_i}{K_L}-1\right)s-\dfrac{RK_b K_g}{K_L M}\right)}$$

$$= \frac{\dfrac{K_i}{2K_L}s^2}{s^3+\dfrac{R}{K_L}s^2+\dfrac{K_b K_g}{M}\left(\dfrac{NAK_i}{K_L}-1\right)s-\dfrac{RK_b K_g}{K_L M}}$$

A.1.3. Derivation of F_0, B_0, G_0 and I_0

These values have previously been defined as part of the design work for the initial phases of the test rig. They have been included here for completeness. As stated previously, they represent the nominal operating point that was discussed at the start of this section.

A.1.3.1. F_0

The Rig is specified to lift 200kg on 4 magnets (one in each corner), therefore 1 magnet is to have the capacity to lift 50kg.

As such:

F_0 = 500N

A.1.3.2. G_0

G_0, is simply the specified airgap between the rail and the magnets. In this design:

G_0 = 10mm

A.1.3.3. B_0

B_0 is the standard value of magnetic flux carried by the poleface. The value chosen was a nominal one selected based on experience. From this, design characteristics of the magnet could be derived. The chosen value is:

B_0 = 0.5T

From this choice, the required area of the poleface needed to cope with this value can be derived. This is a similar process as to calculating the required thickness of a wire based on a specified amount of resistance. The total area of the poleface calculated will take into account both polefaces as the 'resistive effect' to the electromagnetic flow will double because of the two air gaps.

Therefore, from standard electromagnetic physics:

$$A = \frac{2\mu_0 F_0}{B_0^{\,2}}$$

Which yields:

$$A_{total} = \frac{2 \times 4\pi \times 10^{-7} \times 500}{0.5^2}$$
$$\approx 0.005 m^2$$
$$= 50 cm^2$$

And thus the required pole diameter of a single magnet can be found:

$$A = \frac{\pi D^2}{4}$$
$$\therefore D = \sqrt{\frac{0.0025 \times 4}{\pi}}$$
$$= 56 mm$$

A.1.3.4. I_0

I_0 is the nominal operating current of the system. The derivation starts by deriving the number of ampere turns needed to induce the required amount of flux through the electromagnetic circuit. This is rather like calculating the amount of voltage required to draw current through an electrical circuit.

A complete calculation would take into account the flow of flux throughout the entire electromagnetic circuit. In practice, this would be similar to taking into account the resistance of an entire electromagnetic circuit, i.e. taking into account the resistance of copper tracks that join up resisters. The focus is therefore on the resistive nature of the two air gaps contained in the electromagnetic loop.

Again, from standard electromagnetic physics:

$$B = \frac{NI\mu_0}{2G}$$

Note the factor of 2 that appears in the denominator to take into account the two airgaps within the loop.

Therefore:

$$NI = \frac{2G_0B_0}{\mu_0}$$
$$= \frac{2\times 0.01\times 0.5}{4\pi\times 10^{-7}}$$
$$\approx 8000AT$$

To allow for flux leakage:

$$NI = 10000AT$$

As there are two coils to each magnet, each coil must provide 5000AT.

Each coil has been built with 456 turns so therefore:

I_0 = 11A

The coil resistance is 1 Ohm.

A.1.4. Leakage Inductance

The leakage inductance, L, used is a nominal value based on 5% of the mutual inductance.

The mutual inductance is give by:

$$NA\frac{B_0}{I_0}$$

Therefore, the value of 'L' is given by:

$$L = 0.05\times NA\frac{B_0}{I_0}$$
$$= 0.05\times 456\times 0.0025\times \frac{0.5}{11}$$
$$= 0.00259H$$

This is the value of inductance for a single coil, not for the full magnet.

A.1.5. Summary

Therefore, the following has been derived:

Description	Unit	Value
Operating force of each magnet	F_0	500N
Operating Airgap	G_0	0.01m
Operating flux density	B_0	0.5T
Operating current	I_0	11A
Poleface Area (single magnet)	A	0.0025m2
Number of turns (single coil)	N	456
Coil Resistance (single coil)	R	0.5 Ohm
Coil inductance (single coil)	L	2.59 mH
B_0/I_0	K_i	0.0455 T/A
B_0/G_0	K_g	50 T/m
$2F_0/B_0$	K_f	2000 N/T
$L+NAK_i$	K_L	0.0545 H

Table A-1 Summary of derived Maglev system variables

These values can now be used within the Matlab Simulink model for the design of an appropriate controller

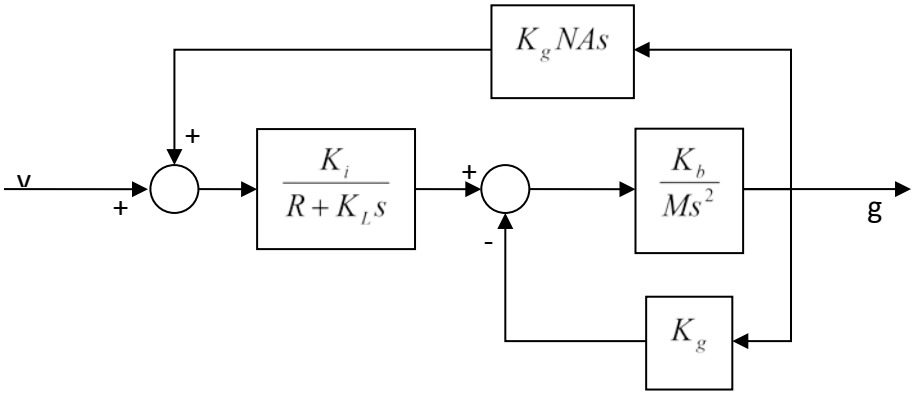
A.2. Controller Design and Development

The structure of the controller is taken from a known solution as stated in REFERENCE. It consists of a flux loop nested by an outer gap loop. In addition to this, it is envisaged that upon final implementation, both loops will be implemented digitally. The flux loop will be executed locally to the sensors and actuators, but the gap loop will implemented over a network. Initial tests show that the approximate frequency that the flux loop can run will be 1000Hz, and the gap loop 100Hz.

The following describes how the appropriate digital control functions are derived, based on the research done in REFERENCE.

A.2.1. Inner Flux Loop Controller Design

Consider Figure A-2, that shows a simplified block diagram of the magnet system:



The feedback loop at the top of the picture is a manipulation of the formula to avoid the use of a differentiator when the system is implemented in simulink. The true form of this block diagram is shown inFigure A-7, correctly showing the induced voltage due to the coil as a function of flux:

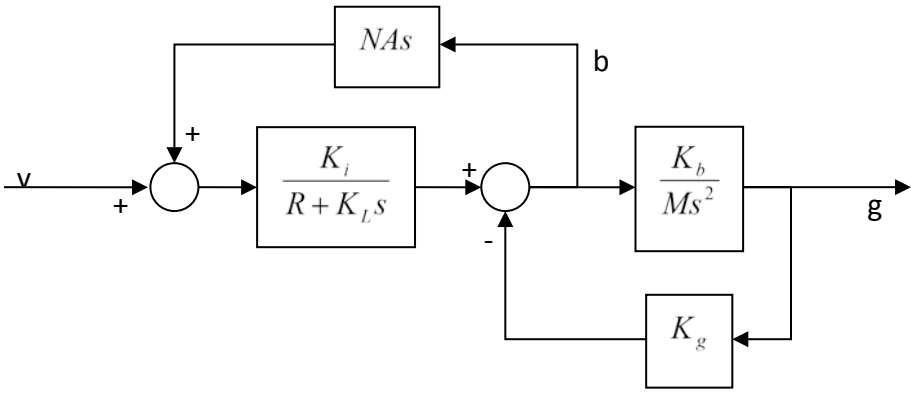


Figure A-7 Modified block diagram of magnet system

The aim is to stabilise the inner flux loop by applying a controller that will vary the input voltage based on the error between the actual and demanded flux level. Applying a generic controller to the above system is represented by the following diagram:

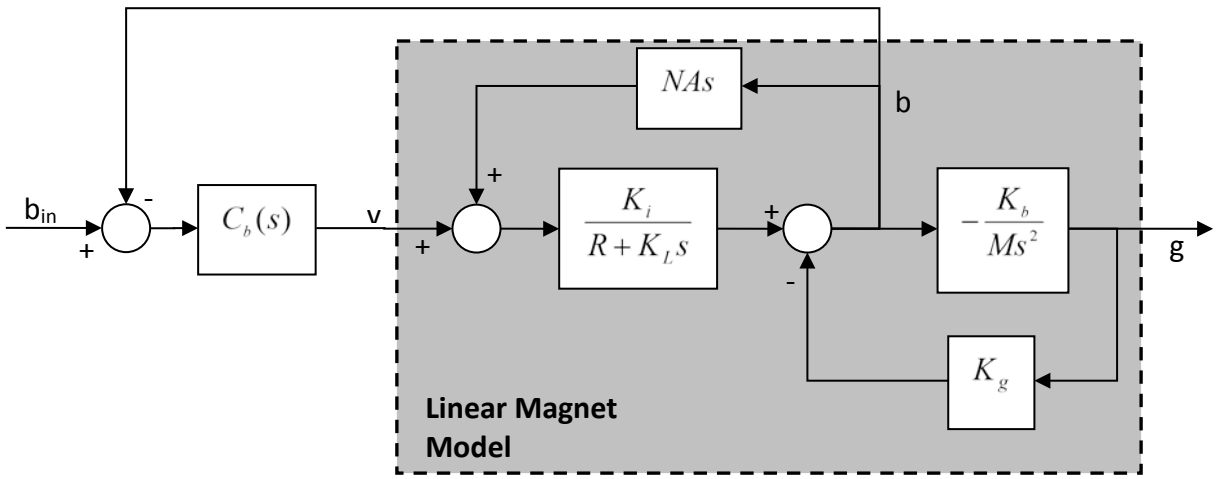


Figure A-8 Flux control loop

This control loop will have to have appropriate gain and phase margins of GM>6db and PM>40degres.

A.2.1.1. Analogue Controller Design

A proportional plus integral (PI) controller should provide good control characteristics for the inner loop, the design of which is shown in Figure A-9. This is the simulink model used to test the PI controller.

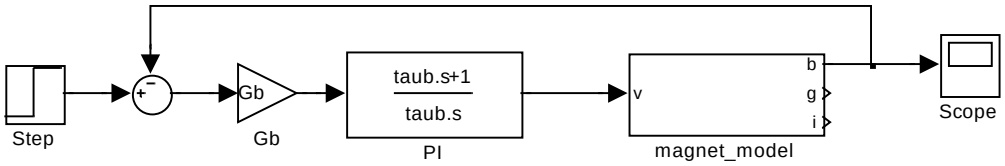


Figure A-9 Simulink model of flux loop

The uncompensated Nichols plot of v(s)/b(s) is shown in Figure A-10:

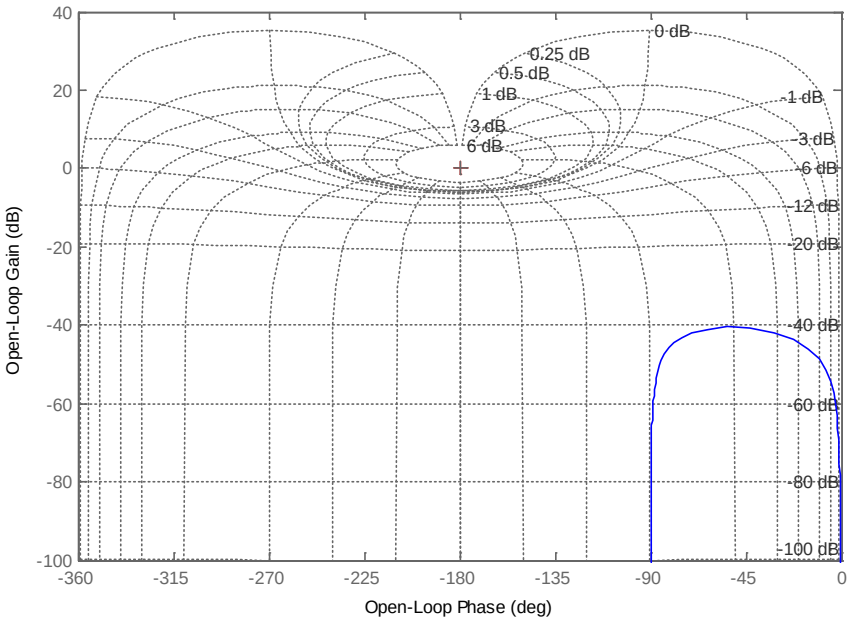


Figure A-10 Nichols Plot of uncompensated flux loop

It is necessary to derive values of gain and integral action required to achieve appropriate stability margins

Gain (Gb)

It can be shown that the open loop transfer function between voltage and flux density at high frequencies can be approximated to:

$$\frac{b(s)}{v(s)} \approx \frac{1}{sNA}$$

Therefore, to give a bandwidth of f_b, the flux loop gain needs to be:

$$G_b = 2\pi f_b NA \text{ in units of V/T}$$

Assuming a bandwidth of 50Hz:

$$\begin{aligned} G_b &= 2\pi f_b NA \\ &= 2\pi \times 50 \times 456 \times 50 \times 10^{-4} \\ &= 716 V/T \\ \therefore G_b &\approx 700 V/T \end{aligned}$$

Applying this gain yields the Nichols chart shown in Figure A-11.

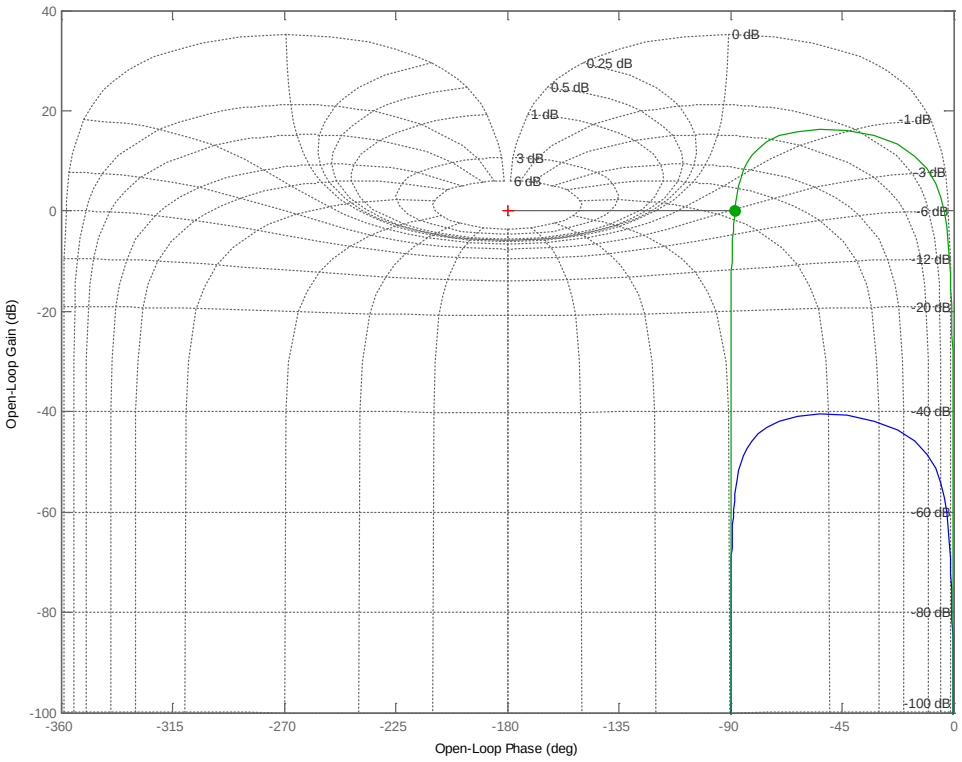


Figure A-11 Nichols plot of flux loop with proportional control

The measure stability attributes are: PM = 91.8 degrees and w = 292 rad/s.

Integral action

As can be seen in Figure A-11, there is a large phase margin (~90 degrees) that can be exploited to improve the control response. This allows the introduction of Integral action to the controller to ensure a zero steady state error in the flux loop.

From the Nichols chart above, we can assume that for an optimal response of a PM of 70 degrees, we require a 20 degree phase shift at 0dB crossover point.

The break frequency of the integral action should be approximately 20Hz. This value will be used to see how effective it is. To derive τ_b :

$$f_b = 20Hz$$

$$\tau_b = \frac{1}{2\pi f_b}$$

$$\therefore \tau_b = \frac{1}{40\pi} = 0.008$$

Applying this to the control loop, the following Nichols chart is obtained.

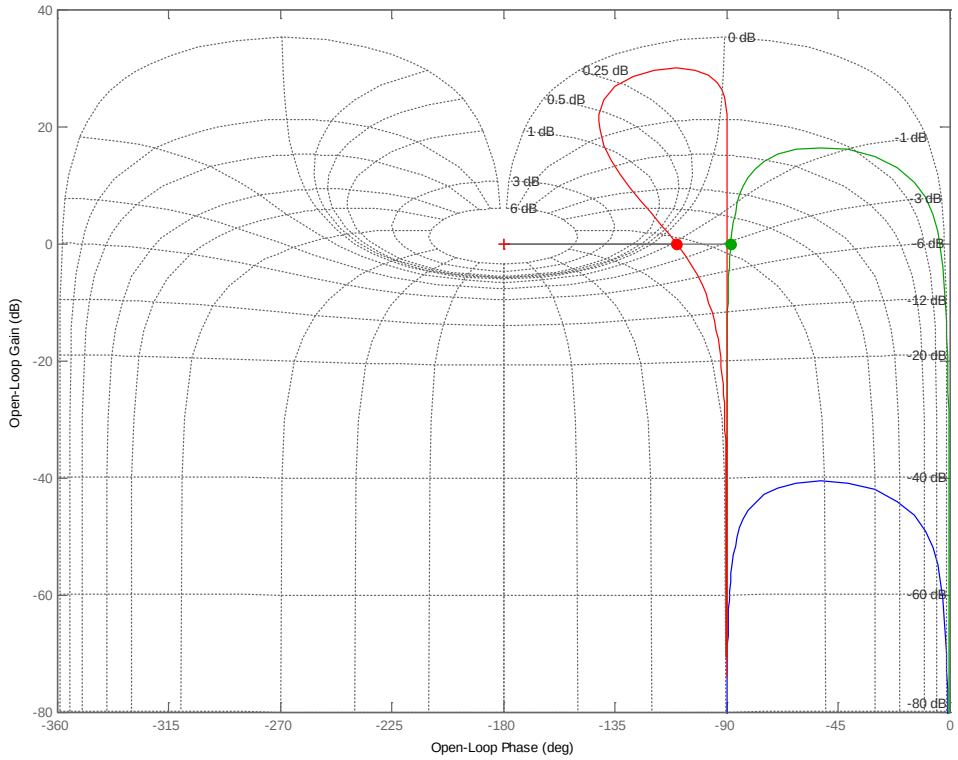


Figure A-12 Nichols Plot of PI compensated flux loop

From this Nichols plot, it is possible to establish that the phase margin of the system is 69.9 degrees and the closed loop bandwidth is 416 rads/s, which is a closed loop bandwidth of 66Hz.

A.2.1.2. Digital Control Design

The controller described above will be implemented on a digital controller with the capacity to sample at 1000Hz. This is below the recommended bandwidth for design by emulation, thus design will take place using w-plane design. For this purpose, a zero-order hold is introduced to the model to represent this digital controller.

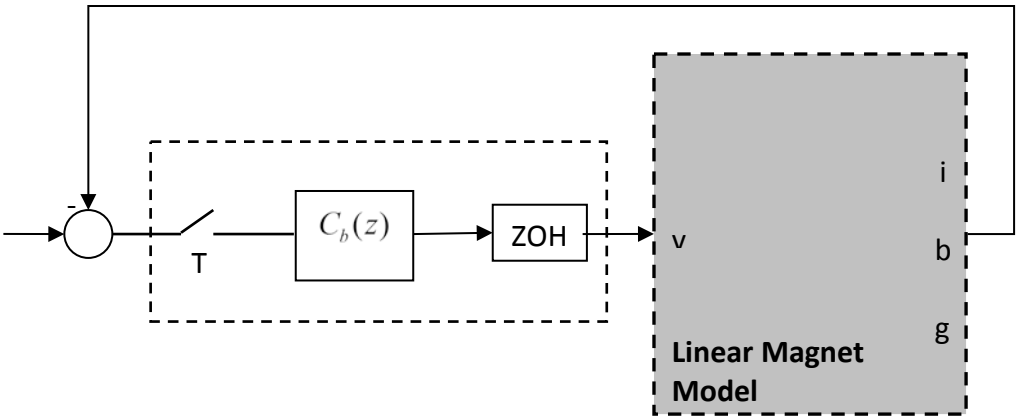


Figure A-13 Digitized representation of flux controller

The simulink model used to represent the uncompensated system, inclusive of the controller sampling zero-order hold is shown in Figure A-14.

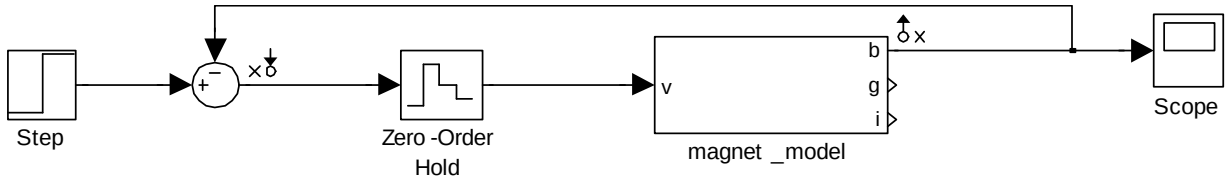


Figure A-14 Simulink model of sampling in loop

The Nichols plot of this system is shown in Figure A-15:

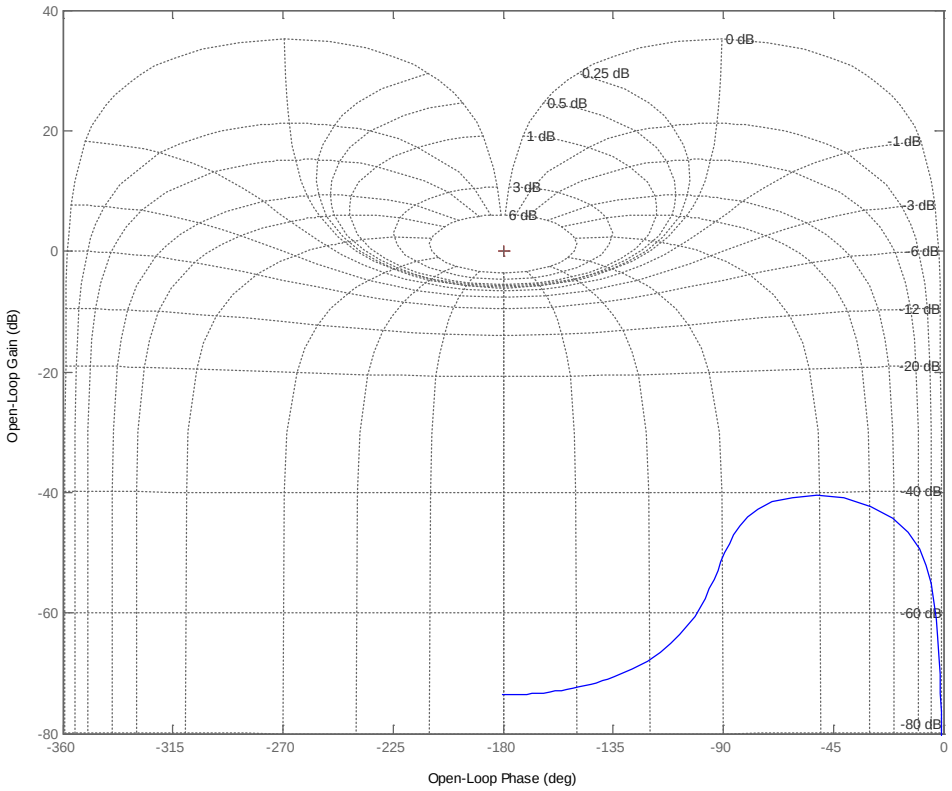


Figure A-15 Nichols Plot of uncompensated system with sampling

As with the analogue controller, it is necessary to derive appropriate values of gain and integral action.

Gain (Gb)

It can be shown that the open loop transfer function between voltage and flux density at high frequencies can be approximated to:

$$\frac{b(s)}{v(s)} \approx \frac{1}{sNA}$$

Therefore, to give a bandwidth of f_b, the flux loop gain needs to be:

$$G_b = 2\pi f_b NA \text{ in units of V/T}$$

Assuming a bandwidth of 50Hz:

$$\begin{aligned} G_b &= 2\pi f_b NA \\ &= 2\pi \times 50 \times 456 \times 50 \times 10^{-4} \\ &= 716 V/T \\ \therefore G_b &\approx 700 V/T \end{aligned}$$

Applying this gain yields the Nichols chart shown in Figure A-16.

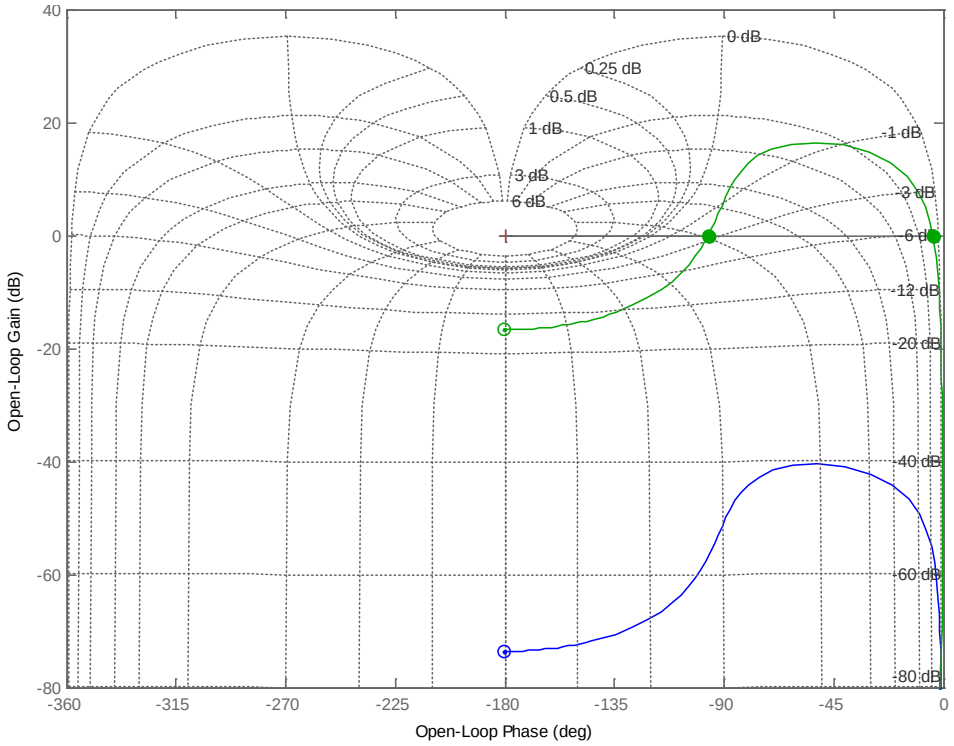


Figure A-16 Nichols Plot of flux loop with digital proportional control

The measured stability margins are: PM = 83.4 degrees at w = 293 rad/s.

Integral action

As can be seen, there is still a large phase margin (~85 degrees) that can be exploited. As such, integral action can be added to the controller to ensure a zero steady state error in the flux loop.

From the Nichols chart above, we can assume that for an optimal response of a PM of 70 degrees, we require a 15 degree phase shift at 0dB crossover point.

Goodall [insert reference] states that the break frequency of the integral action should be approximately 20Hz. This value will be used to see how effective it is. To derive τ_b:

$$f_b = 20Hz$$

$$\tau_b = \frac{1}{2\pi f_b}$$

$$\therefore \tau_b = \frac{1}{40\pi} = 0.008$$

Applying these values to the controller the Nichols chart in Figure A-17 is obtained.

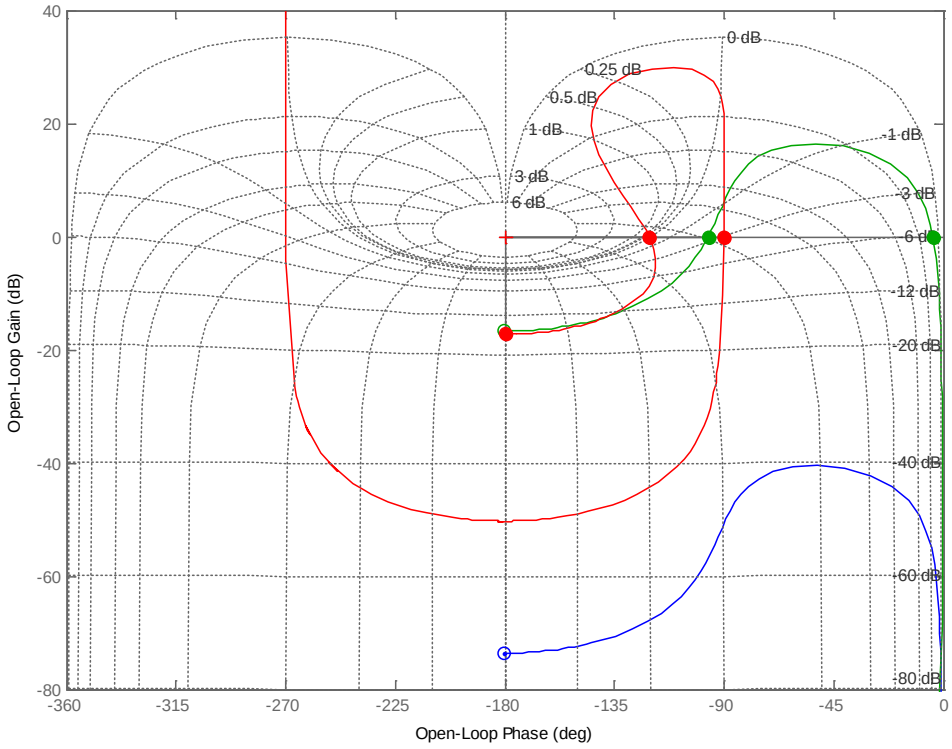


Figure A-17 Nichols plot of flux loop with digital controller

From this Nichols plot, it is possible to establish that the phase margin of the system is 59.3 degrees and gain margin of 17.3dB. The closed loop bandwidth is 452 rads/s, which is a closed loop bandwidth of 71.9Hz.

A.2.1.3. Digitization of Controller (z)

The controller that has been designed is in continuous time. The implementation will be on a computer and therefore will need digitising. This can be done by applying the bilinear transform to the continuous time controller. The controller derived from above is:

$$C(w) = G_b\left(\frac{\tau_b w + 1}{\tau_b w}\right)$$

Where:

$G_b = 700V/T$

$\tau_b = 0.008s$

For Bilinear transform:

$$w \Rightarrow \frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}$$

So:

$$C(z) = G_b\left(\frac{\tau_b\left(\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}\right)+1}{\tau_b\left(\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}\right)}\right)$$

$$= G_b\left(\frac{2\tau_b\left(1-z^{-1}\right)+T\left(1+z^{-1}\right)}{2\tau_b\left(1-z^{-1}\right)}\right)$$

$$= G_b\left(\frac{2\tau_b - 2\tau_b z^{-1} + T + Tz^{-1}}{2\tau_b - 2\tau_b z^{-1}}\right)$$

$$C(z) = \frac{G_b\left(T+2\tau_b\right)+G_b\left(T-2\tau_b\right)z^{-1}}{2\tau_b - 2\tau_b z^{-1}}$$

This can be expressed as:

$$C(z) = \frac{ab0 + ab1z^{-1}}{bb0 + bb1z^{-1}}$$

Where:

$$ab0 = G_b\left(T+2\tau_b\right)$$
$$ab1 = G_b\left(T-2\tau_b\right)$$
$$bb0 = 2\tau_b$$
$$bb1 = -2\tau_b$$

The discrete controller is applied to the simulink diagram in the following way:

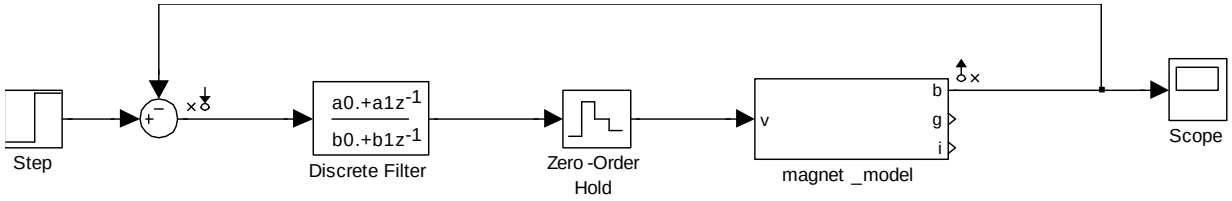


Figure A-18 Flux loop with digital controller

The Nichols plot in Figure A-19 shows the discrete time controller system implemented at 1000Hz.

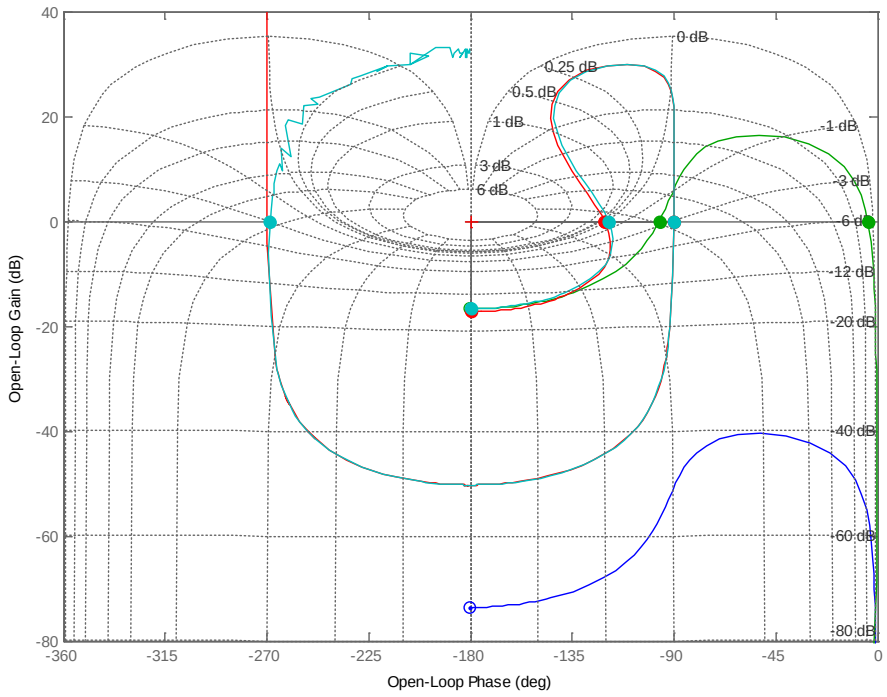


Figure A-19 Nichols plot of flux loop with 1kHz digital controller

The phase margin is 61.1 degrees and the gain margin is 16.7 degrees. The closed loop bandwidth of the system is 474 rad/s which is equivalent to 75.4Hz.

A.2.1.4. Digitisation of Controller (delta)

The controller design has been performed in continuous time and requires digitising in order for this to be implemented on a computer. This can be done by applying the bilinear transform using the delta operator and then deriving the real time control equations.

$$C(w) = G_b\left(\frac{\tau_b w + 1}{\tau_b w}\right)$$

Where:

$G_b = 700V/T$

$\tau_b = 0.008s$

For the bilinear transform:

$$w \Rightarrow \frac{2\delta}{T(2+\delta)}$$

Therefore:

$$\begin{aligned}
C(w) &= G_b\left(\frac{\tau_b w + 1}{\tau_b w}\right) \\
&= G_b\left(\frac{\tau_b\left(\frac{2\delta}{T(2+\delta)}\right)+1}{\tau_b\left(\frac{2\delta}{T(2+\delta)}\right)}\right) \\
&= G_b\left(\frac{2\tau_b\delta + T(2+\delta)}{2\tau_b\delta}\right) \\
&= G_b\left(\frac{(2\tau_b + T)\delta + 2T}{2\tau_b\delta}\right) \\
&= \frac{G_b(2\tau_b + T)\delta + 2G_bT}{2\tau_b\delta} \\
&= \frac{G_b(2\tau_b + T) + 2G_bT\delta^{-1}}{2\tau_b} \\
&= \frac{\frac{G_b}{2\tau_b}(2\tau_b + T) + \frac{G_bT}{\tau_b}\delta^{-1}}{1} \\
&= p + q\delta^{-1}
\end{aligned}$$

Where:

$$p = \frac{G_b(2\tau_b + T)}{2\tau_b}$$

$$q = \frac{G_b T}{\tau_b}$$

The real time equations are derived as follows. Let:

$$C(w) = \frac{y(\delta)}{v(\delta)} \times \frac{v(\delta)}{u(\delta)} = p + q\delta^{-1}$$

Where 'y' is the output, u is the input and v is an internal variable. The above function is expanded below:

$$\frac{y(\delta)}{v(\delta)} = p + q\delta^{-1}$$

$$\frac{v(\delta)}{u(\delta)} = 1$$

And also:

$$v = u$$

(Equation 1)

Then:

$$y(\delta) = pv + q\delta^{-1}v$$

Let:

$$w = \delta^{-1}v$$

$$\therefore w = w + v$$

(Equation 2)

So:

$$y = pv + qw$$

(Equation 3)

Equations 1, 2 and 3 are implemented in that order to perform the control actions.

A.2.2. Outer Gap-loop Control Design

A.2.2.1. Analogue Controller Design

When a controller is applied to the outer loop, the control structure generated for a single magnet is as shown in Figure A-20.

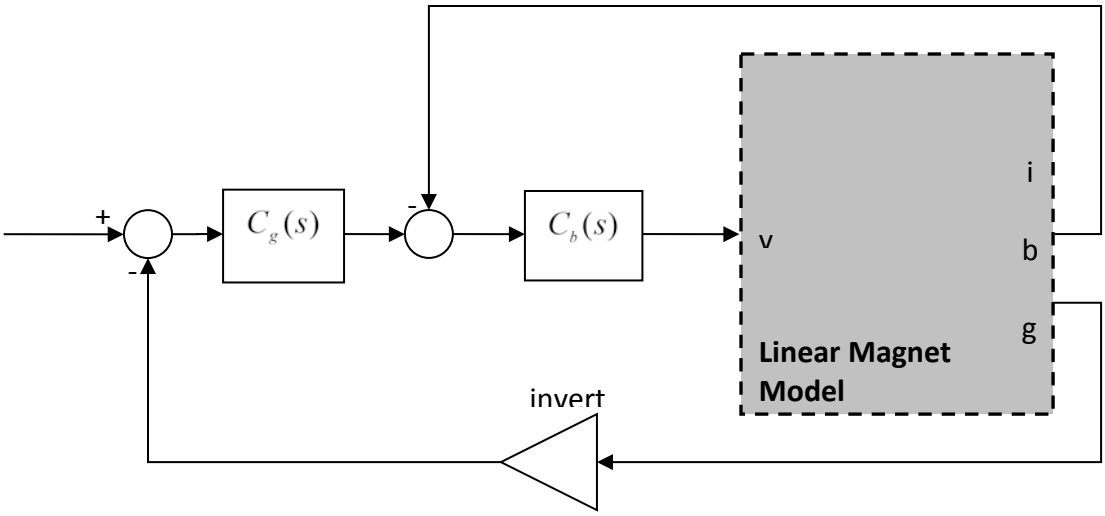


Figure A-20 Control Structure for Maglev system

The flux loop controller was derived in the previous section (A.2.1) and was found to be:

$$C_b = 700\frac{(0.008+1)}{0.008}$$

The inversion on the gap feedback loop is required to correct the polarity and give a positive value of gap.

The Nichols plot for the uncompensated outer loop is shown in Figure A-21.

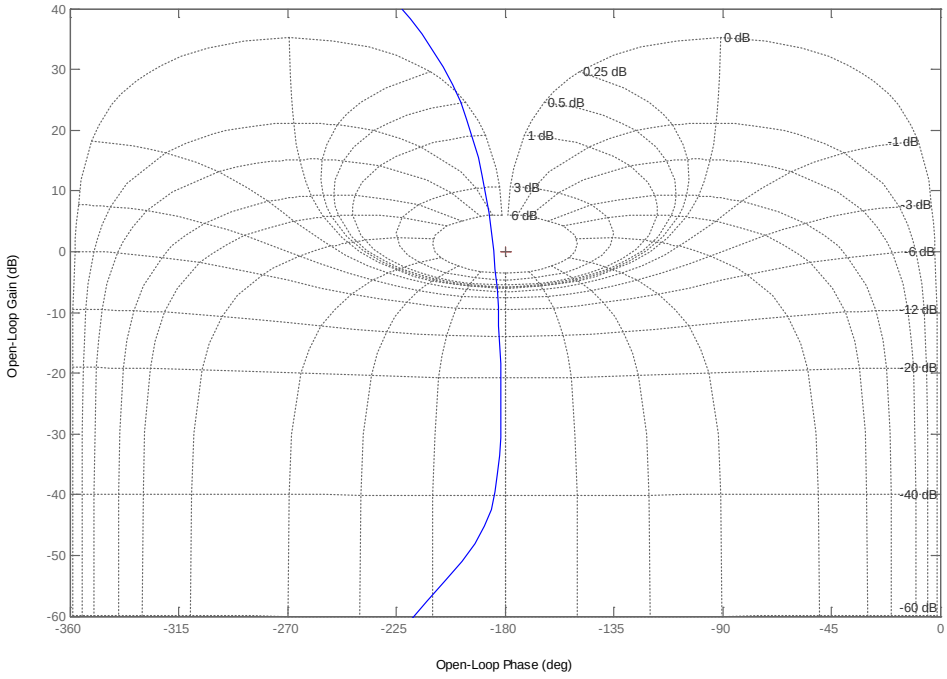
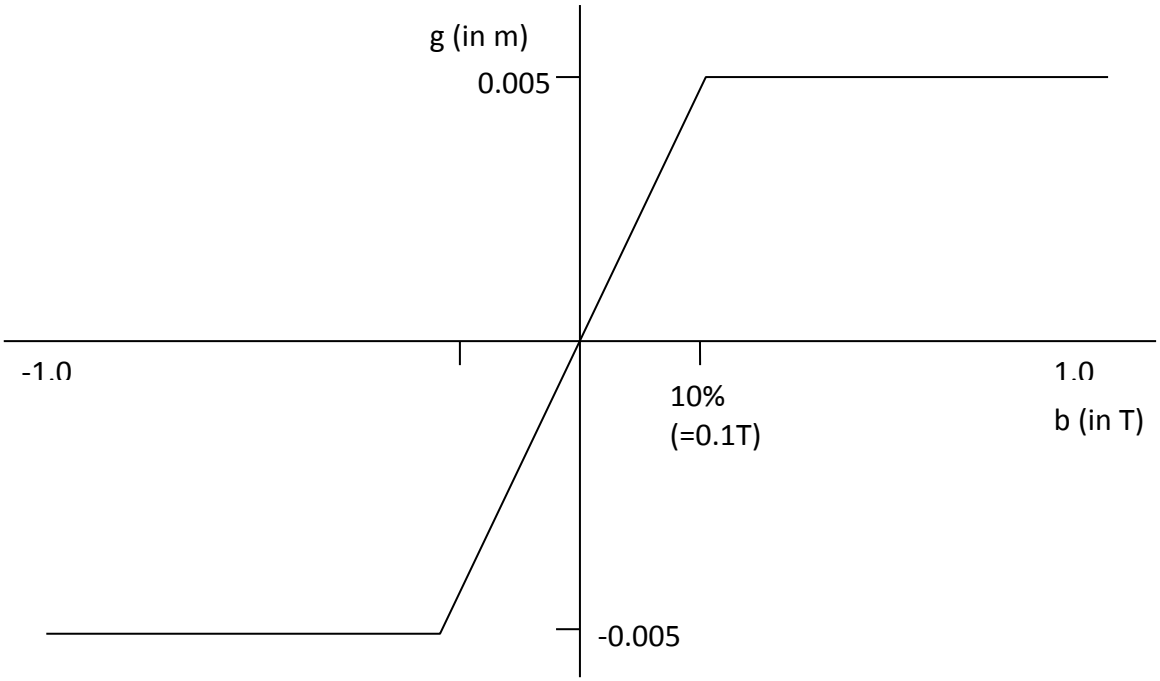


Figure A-21 Nichols Plot of uncompensated Gap Loop

It can be seen that there is a need for phase advance to stabilise the system.

A suitable value of gain, can be found by considering a proportional gain of 10%. Assuming the maximum change in gap allowable is 0.005m (5mm), and the maximum change in flux density is 1T:



The gradient shown will equal the gain of the controller.

$$G_g = \frac{0.1}{0.005} = 20T/m$$

The Nichols chart obtained by applying this controller is shown in Figure A-22:

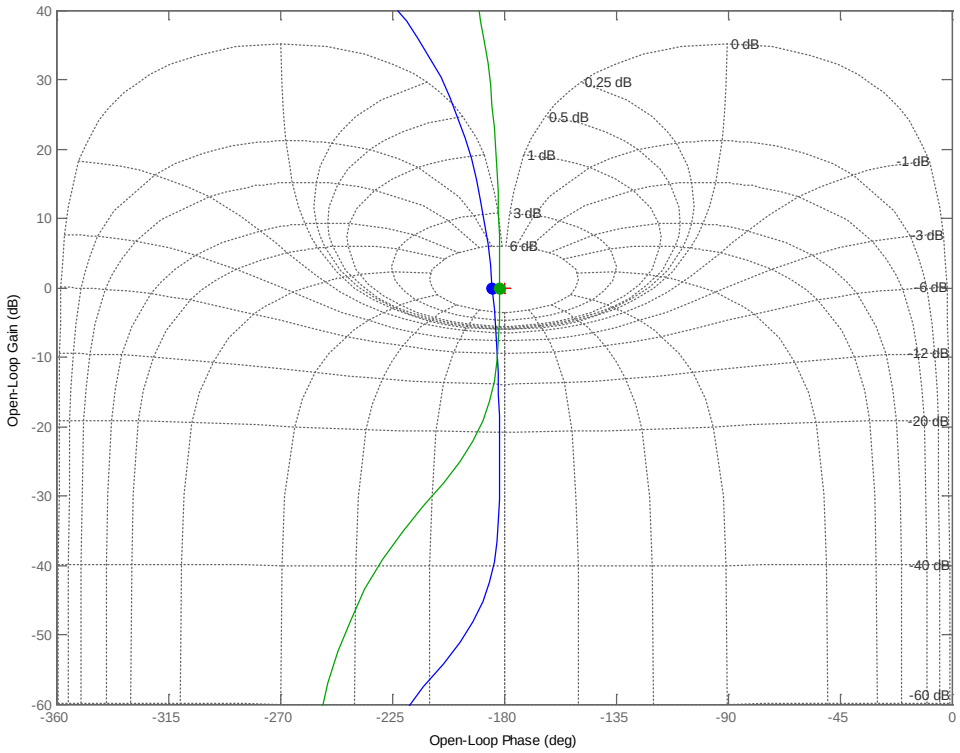


Figure A-22 Nichols Plot of proportional control on gap loop

A Phase advance controller has the following structure:

$$C_g(s) = G_g \frac{(k\tau s + 1)}{(\tau s + 1)}$$

Where k is the phase advance ratio, G_g is the gain and τ is the time constant.

The value of k is a nominal value and is chosen here to be 5.

The Phase advance implemented by this controller can be expressed as:

$$\begin{aligned}\angle C_g(s) &= \angle C_g(jw) \\ &= \angle G_g + \angle(k\tau\omega j + 1) - \angle(\tau\omega j + 1) \\ &= \tan^{-1}(G_g) + \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right) \\ &= \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right)\end{aligned}$$

At the 0dB point, we require a phase advance of at least 40 degrees. With 'k' already chosen, and $\omega = 28.5 rad/s$ at 0dB,

$$-40 = \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right)$$

$$\tan(-40) = \frac{1}{k\tau\omega} - \frac{1}{\tau\omega}$$

$$= \frac{(1-k)}{k\tau\omega}$$

$$\tau = \frac{(1-k)}{k\omega\tan(-40)}$$

$$= \frac{(1-5)}{5\times 28.5\times(-0.839)}$$

$$= \frac{-4}{-119.6}$$

$$\therefore \tau = 0.0334$$

The Nichols chart in Figure A-23 shows the response of the system with controller with the above parameters.

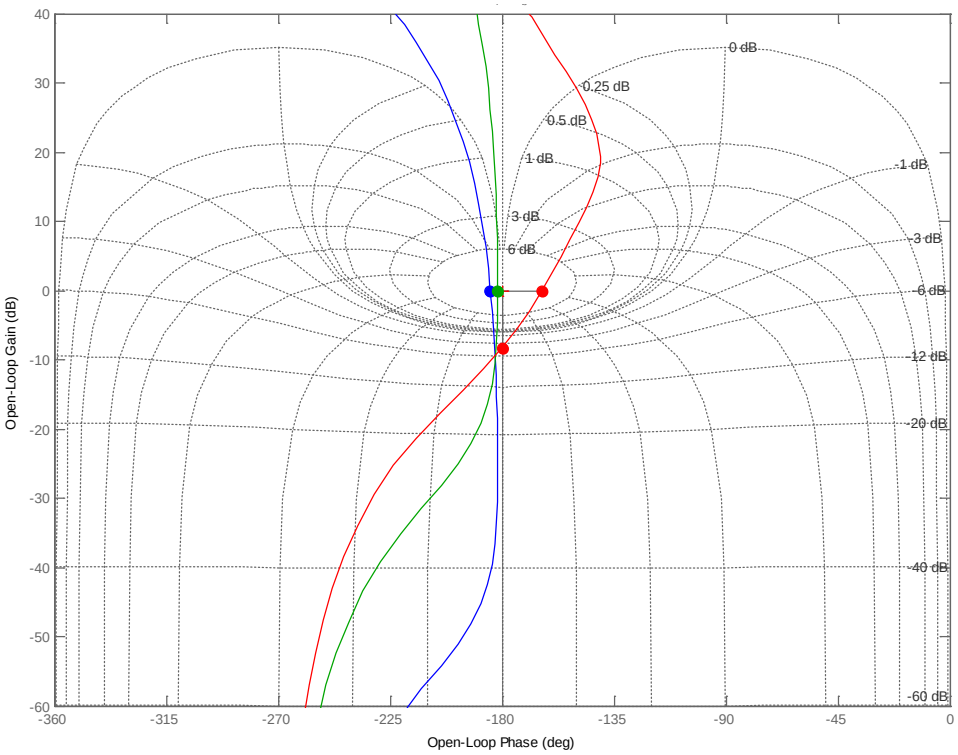


Figure A-23 Nichols plot of PA Gap controller applied to system

This gives a phase margin of 16.1 degrees, and a gain margin of 8.31dB. It can be seen on the Nichols chart above that a larger stability margin can be achieved by adjusting the active frequency of the controller. As such, the value of τ is adjusted to 0.01. This gives the following Nichols plot:

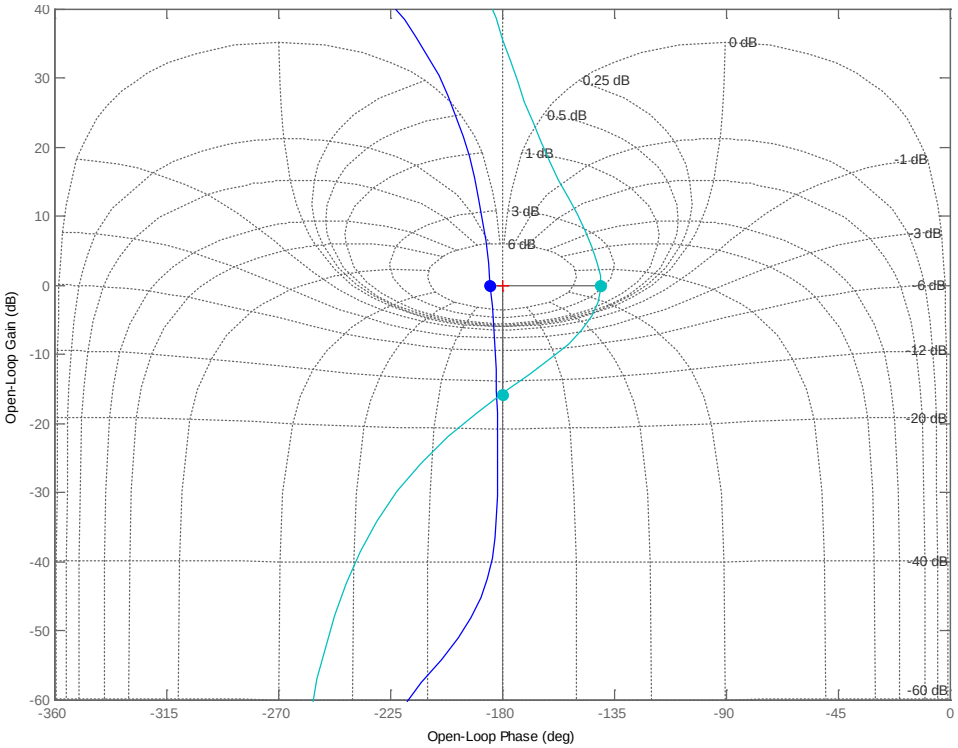


Figure A-24 Nichols plot of adjust PA Gap controller applied to system

This controller yields a phase margin of 39.6 degrees, and a gain margin of 15.8dB. The bandwidth of the system was found to be 12.5Hz.

As a final test, a gap step input of 2mm is put into the loop. The following is a copy of the output of the system.

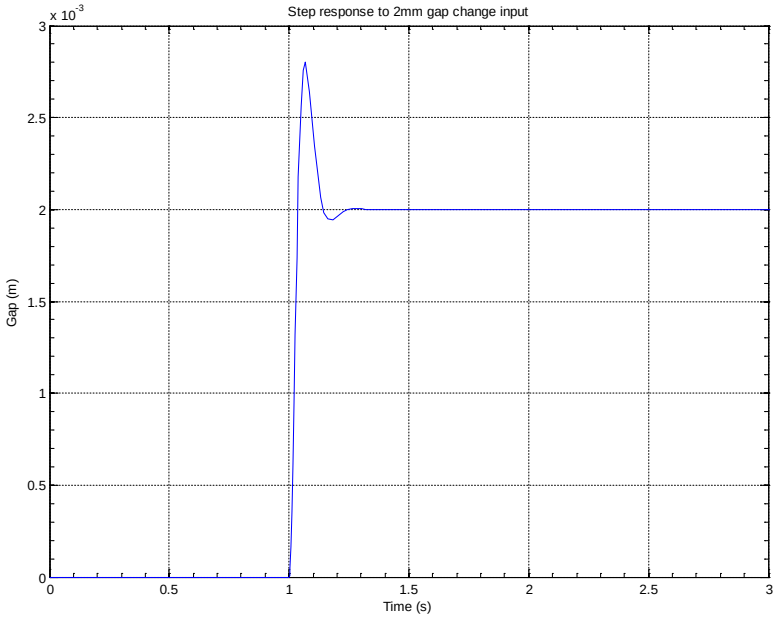


Figure A-25 Step response of system

As can be seen, the system settles nicely with zero steady state error.

A.2.2.2. Digital Controller Design

Initial test show that the outer gap loop will be implemented at around 100Hz. As with the flux loop, this is lower than the recommended sampling frequency given the bandwidth of the system. As such, a w-plane design will be used to derive a controller.

The control structure inclusive of the sampling delay is shown in Figure A-26:

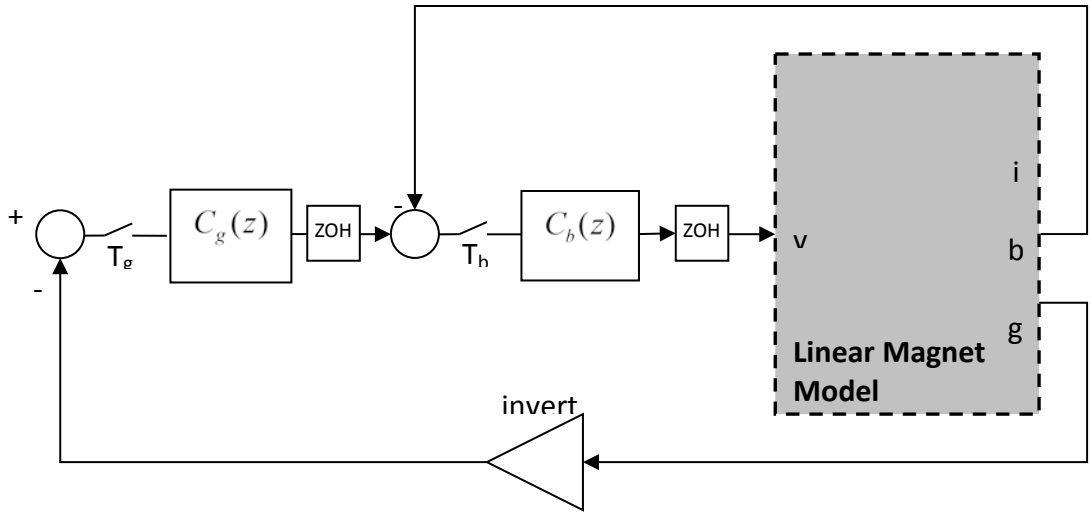


Figure A-26 digital control scheme

T_i is the inner (flux) loop sampling time and T_g is the outer (gap) loop sampling time. The Nichols plot for the uncompensated outer loop is shown in Figure A-27.

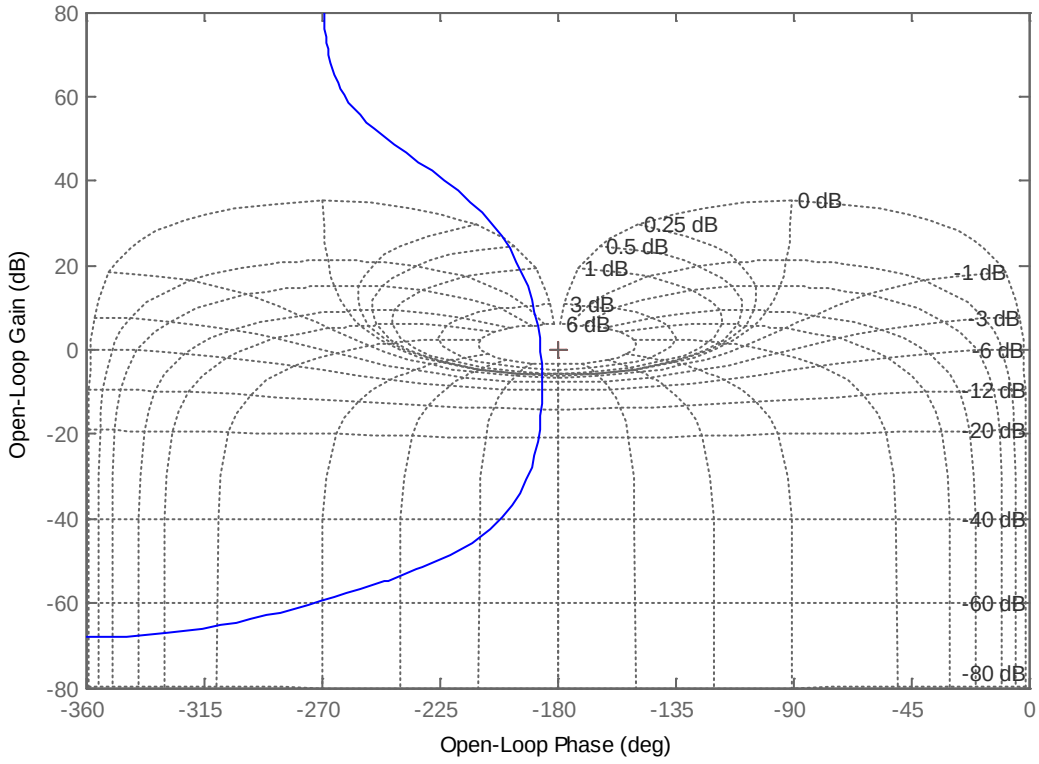


Figure A-27 Nichols plot of Gap loop

Again, it can be seen that there is a need for phase advance to stabilise the system.

Using the same value of gain as previously:

$$G_g = 20T/m$$

The Nichols chart obtained by applying this controller is shown in Figure A-28:

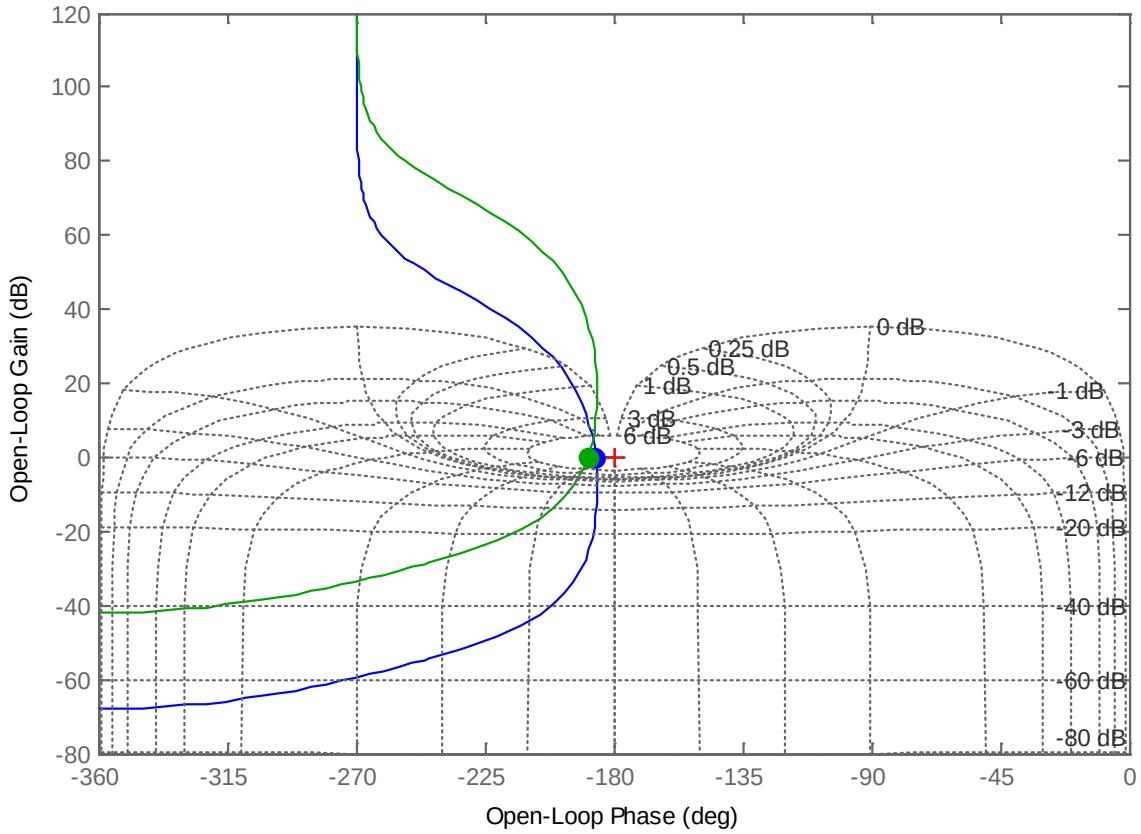


Figure A-28 Nichols chart of Gap loop with proportional gain controller

A Phase advance controller has the following structure:

$$C_g(s) = G_g \frac{(k\tau s+1)}{(\tau s+1)}$$

Where k is the phase advance ratio, G_g is the gain and τ is the time constant.

The value of k is a nominal value and is chosen here to be 6.

The Phase advance implemented by this controller can be expressed as:

$$\begin{aligned}\angle C_g(s) &= \angle C_g(jw) \\ &= \angle G_g + \angle(k\tau\omega j+1) - \angle(\tau\omega j+1) \\ &= \tan^{-1}(G_g) + \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right) \\ &= \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right)\end{aligned}$$

At the 0dB point, we require a phase advance of at least 40 degrees. With 'k' already selected, and $\omega = 28.5 rad/s$ at 0dB,

$$-40 = \tan^{-1}\left(\frac{1}{k\tau\omega}\right) - \tan^{-1}\left(\frac{1}{\tau\omega}\right)$$

$$\tan(-40) = \frac{1}{k\tau\omega} - \frac{1}{\tau\omega}$$

$$= \frac{(1-k)}{k\tau\omega}$$

$$\tau = \frac{(1-k)}{k\omega\tan(-40)}$$

$$= \frac{(1-6)}{6\times 28.5\times(-0.839)}$$

$$= \frac{-5}{-143.5}$$

$$\therefore \tau = 0.0348$$

The Nichols chart in Figure A-29 shows the response of the system with controller with the above parameters.

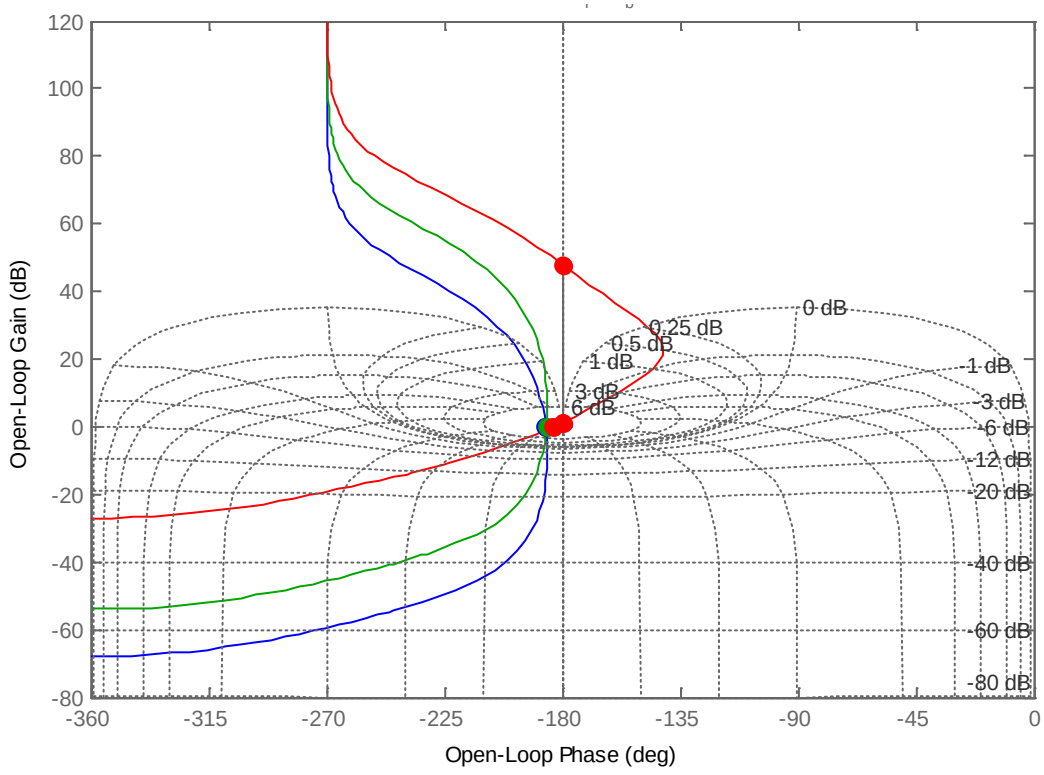


Figure A-29 Nichols Plot of PA Gap controller

This gives a slight negative phase and gain margin. It can be seen on the Nichols chart above that a larger stability margin can be achieved by adjusting the active frequency of the controller, and reducing the gain. As such, the value of τ is adjusted to 0.01, and the value of gain adjusted to 10. This gives the following Nichols plot:

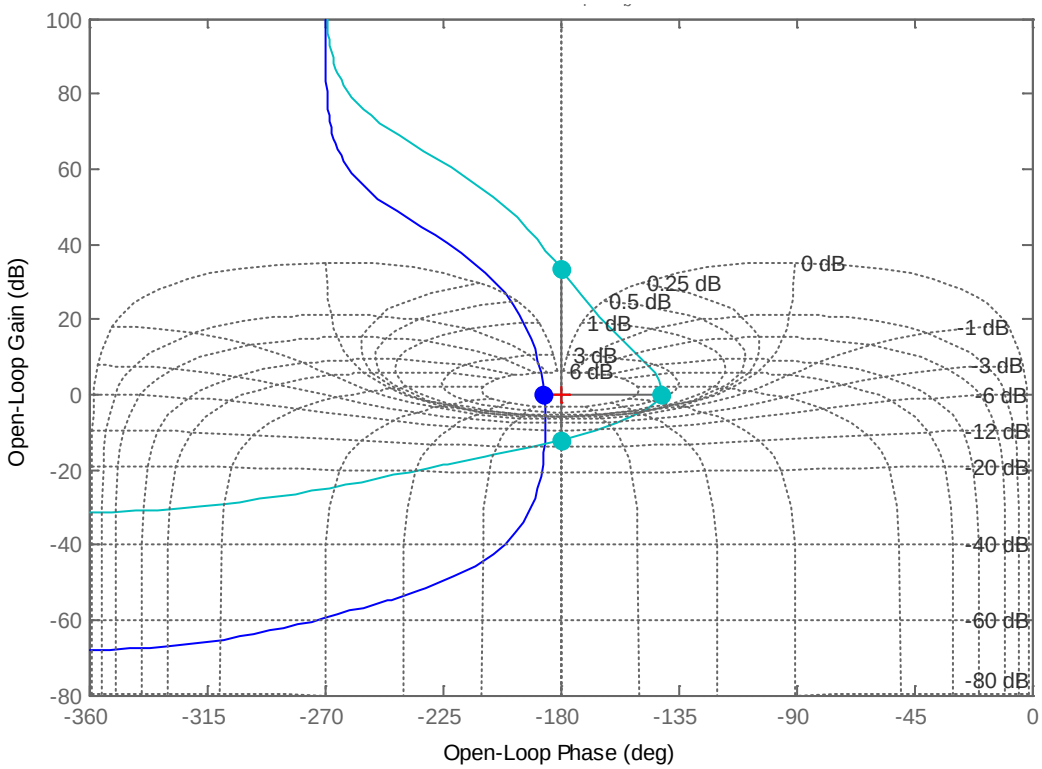


Figure A-30 Nichols plot of adjusted PA gap controller

This controller yields a phase margin of 38.2 degrees, and a gain margin of 12dB.

A.2.3. Digitisation of Controller (z)

The controller generated above is a continuous time controller and requires conversion to a discrete time controller for implementation on a real time computer.

Once again, this can be done by applying the binomial transform to the continuous controller. So:

$$C_g(w) = G_g \frac{(k\tau w + 1)}{(\tau w + 1)}$$

Where:

$$G_g = 10$$
$$k = 5$$
$$\tau = 0.01$$

Applying the binomial transform of:

$$w \Rightarrow \frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}$$

So:

$$C_g(z) = G_g \frac{\left(k\tau\left(\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}\right)+1\right)}{\left(\tau\frac{2\left(1-z^{-1}\right)}{T\left(1+z^{-1}\right)}+1\right)}$$
$$= G_g \frac{\left(2k\tau\left(1-z^{-1}\right)+T\left(1+z^{-1}\right)\right)}{\left(2\tau\left(1-z^{-1}\right)+T\left(1+z^{-1}\right)\right)}$$
$$= G_g \frac{\left(2k\tau-2k\tau z^{-1}+T+Tz^{-1}\right)}{\left(2\tau-2\tau z^{-1}+T+Tz^{-1}\right)}$$
$$C_g(z) = \frac{G_g\left(T+2k\tau\right)+G_g\left(T-2k\tau\right)z^{-1}}{\left(T+2\tau\right)+\left(T-2\tau\right)z^{-1}}$$

This can also be expressed as:

$$C_g(z) = \frac{ag0+ag1z^{-1}}{bg0+bg1z^{-1}}$$

Where:

$$ag0 = G_g\left(T+2k\tau\right)$$
$$ag1 = G_g\left(T-2k\tau\right)$$
$$bg0 = T+2\tau$$
$$bg1 = T-2\tau$$

A.2.4. Digitisation of Controller (delta)

The controller design has been performed in continuous time and requires digitising in order for this to be implemented on a computer. This can be done by applying the bilinear transform using the delta operator and then deriving the real time control equations.

$$C_g(w) = G_g \frac{\left(k\tau w+1\right)}{\left(\tau w+1\right)}$$

Where:

$$G_g = 10$$
$$k = 5$$
$$\tau_b = 0.01s$$

For the bilinear transform:

$$w \Rightarrow \frac{2\delta}{T(2+\delta)}$$

Therefore:

$$\begin{aligned}
C_g(w) &= G_g \frac{(k\tau w+1)}{(\tau w+1)} \\
&= G_g \frac{\left(\frac{2k\tau\delta}{T(2+\delta)}+1\right)}{\left(\frac{2\tau\delta}{T(2+\delta)}+1\right)} \\
&= G_g \frac{(2k\tau\delta+T(2+\delta))}{(2\tau\delta+T(2+\delta))} \\
&= G_g \frac{(2k\tau\delta+2T+T\delta)}{(2\tau\delta+2T+T\delta)} \\
&= G_g \frac{(2k\tau+T)\delta+2T}{(2\tau+T)\delta+2T} \\
&= G_g \frac{\frac{(2k\tau+T)}{(2\tau+T)}+\frac{2T}{(2\tau+T)}\delta^{-1}}{1+\frac{2T}{(2\tau+T)}\delta^{-1}} \\
&= \frac{\frac{G_g(2k\tau+T)}{(2\tau+T)}+\frac{G_g 2T}{(2\tau+T)}\delta^{-1}}{1+\frac{2T}{(2\tau+T)}\delta^{-1}} \\
&= \frac{p+qd_1\delta^{-1}}{1+d_1\delta^{-1}}
\end{aligned}$$

Where:

$$p = \frac{G_g(2k\tau + T)}{(2\tau + T)}$$

$$q = G_g$$

$$d_1 = \frac{2T}{(2\tau + T)}$$

The real time equations are derived as follows. Let:

$$C(w) = \frac{y(\delta)}{v(\delta)} \times \frac{v(\delta)}{u(\delta)} = \frac{p + qd_1\delta^{-1}}{1 + d_1\delta^{-1}}$$

Where 'y' is the output, u is the input and v is an internal variable. The above function is expanded below:

$$\frac{y(\delta)}{v(\delta)} = p + qd_1\delta^{-1}$$

$$\frac{v(\delta)}{u(\delta)} = \frac{1}{1 + d_1\delta^{-1}}$$

So:

$$\frac{v(\delta)}{u(\delta)} = \frac{1}{1 + d_1\delta^{-1}}$$

$$v(1 + d_1\delta^{-1}) = u$$

$$v = u - d_1\delta^{-1}v$$

Let:

$$w = d_1\delta^{-1}v$$

$$\therefore w = w + d_1v$$

(Equation 1)

So:

$$v = u - w$$

(Equation 2)

Remember that:

$$\frac{y(\delta)}{v(\delta)} = p + qd_1\delta^{-1}$$

So:

$$y = pv + qd_1\delta^{-1}v$$

$$y = pv + qw$$

(Equation 3)

Equations 1, 2 and 3 are implemented in that order to perform the control actions.

A.2.5. Step Response Test

As a final test, a gap step input of 2mm is put into the loop. The following is a copy of the output of the system.

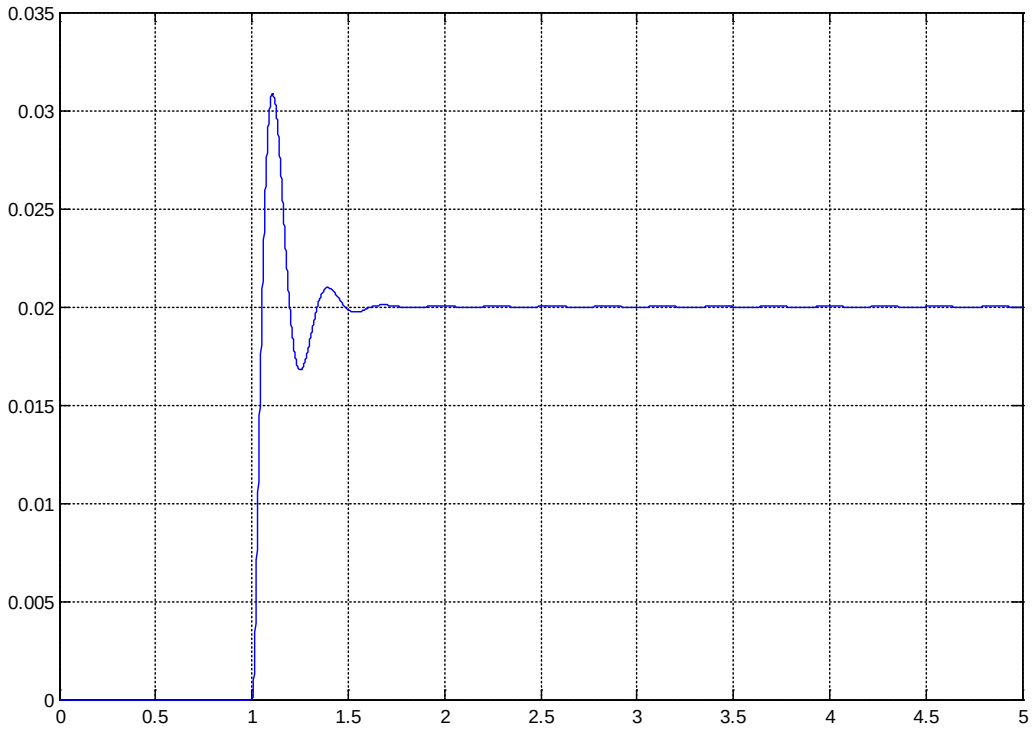
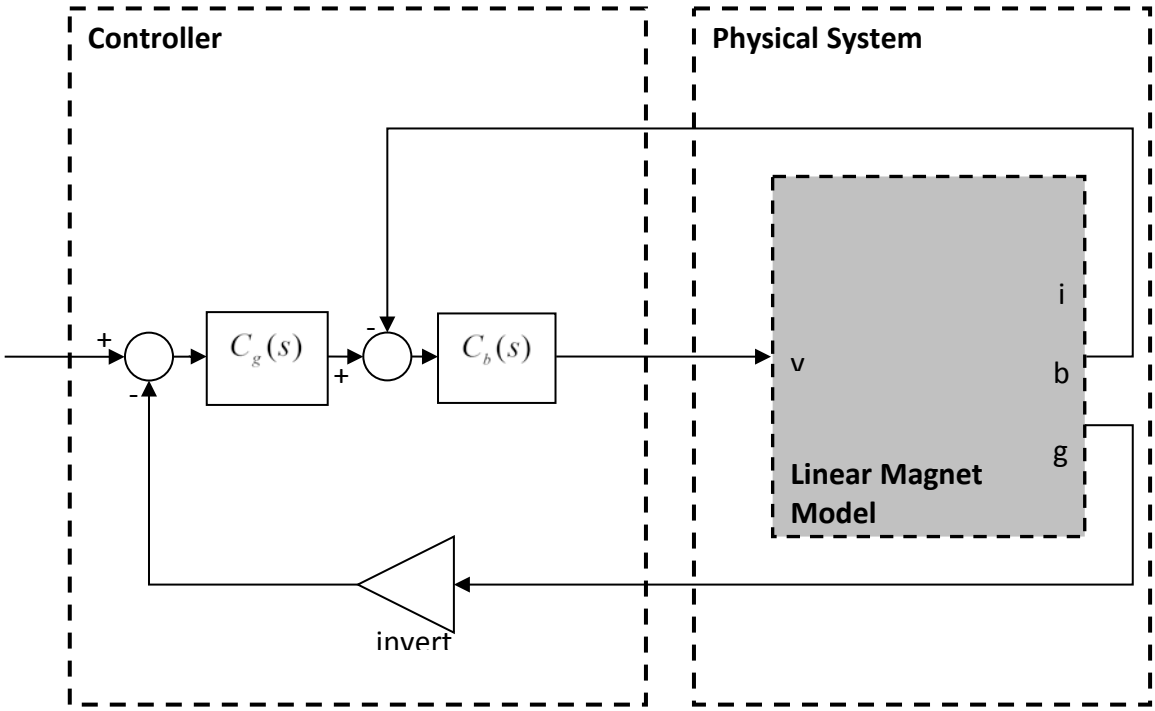


Figure A-31 Step response of digital gap controlled system

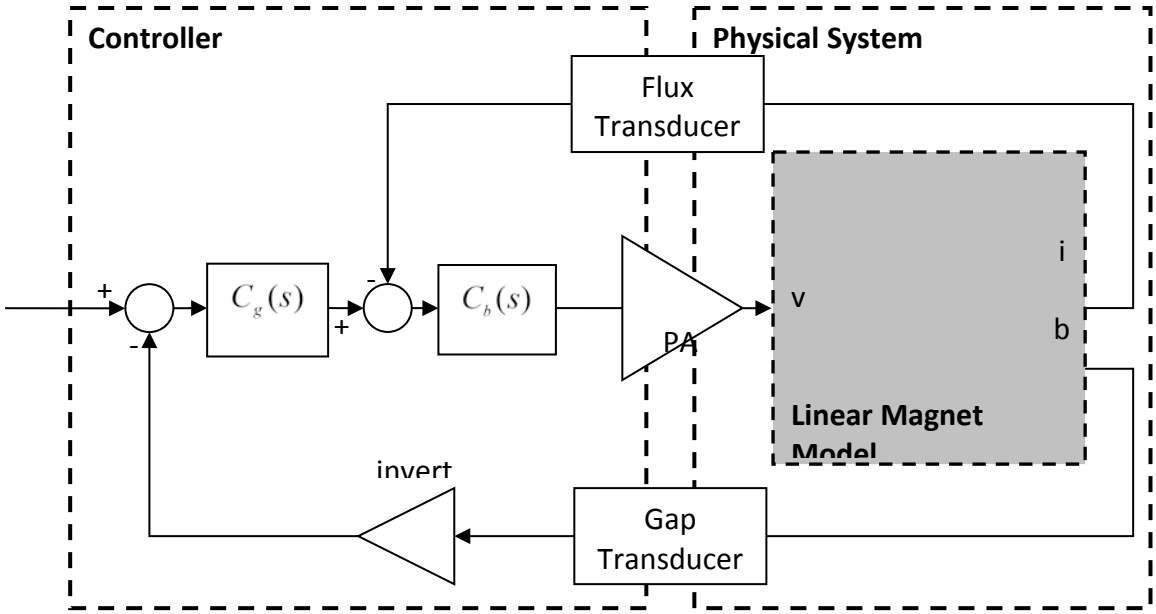
As can be seen, the system overshoots, but does settle with a small oscillatory error. This is a result of implementing the gap controller with a slow time period.

A.3. Hardware Implementation

It is possible to represent the system as a controller and physical system. This is more appropriate to the physical aspect of the final product. The system boundaries are shown in the diagram below.



In order to facilitate the interaction between the controller and the physical boundary, various transducers have been designed. These sit on the above diagram as thus:



From the physics of the system, it is possible to define the maximum and minimum signal values expected during the operation of the system. It is normal in analogue systems to assume that control signals will have the range $\pm 10V$. The other signal values are shown in Figure A-32:

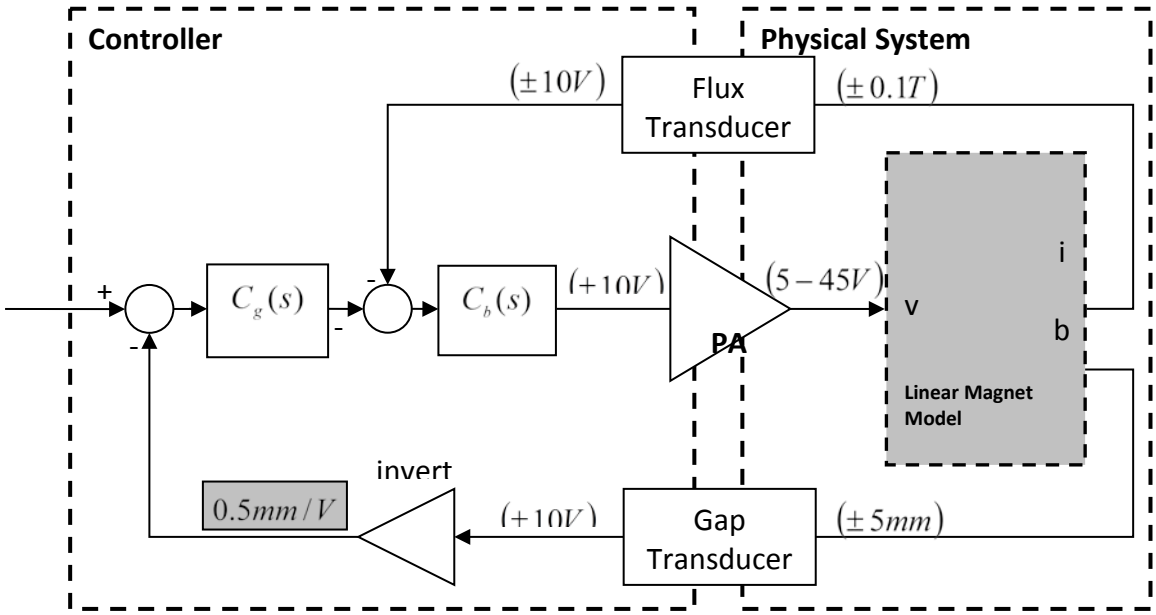


Figure A-32 transducer and power amplification requirements

The following sections describe the design and hardware implementation of each of these trasnducers.

A.3.1. Flux Transducer

The flux transducer design is formed around the concept that a voltage proportional to the value of flux density can be ascertained from a search coil, embedded in the pole face of the magnet. A value of flux is then derived by integrating this signal, and then multiplying by the proportionality constant. It is necessary to include a high pass filter to remove low frequency drift in this application. The process can be demonstrated by the block diagram in Figure A-33:

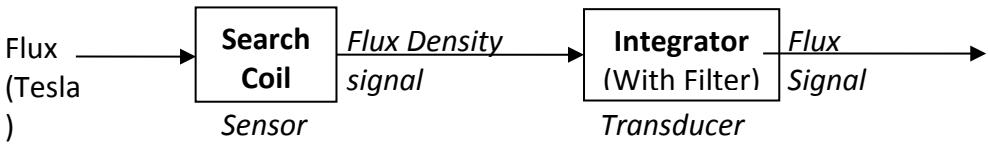


Figure A-33 Scaling considerations for flux measurement

The transfer function for the search coil can be shown to be:

$$\frac{v_{sc}(s)}{b(s)} = N_{sc} A_{sc} s$$

Where v_{sc} is the search coil voltage, N_{sc} is the number of turns in the search coil and A_{sc} is the area bound by the coil. The proportional constant here is based from the physics of the search coil, in that the output voltage is proportional to both the area bound by the coil and the number of turns wound.

The transfer function of the self-zeroing integrator is:

$$F_i = \frac{s}{\left(s^2 + 1.4\omega_i s + \omega_i^2\right)}$$

ω_i needs to be chosen such that it is substantially lower than the bandwidth of the gap loop. Therefore, let:

$$\omega_i = 1.5 rad/s$$

It is also necessary to have the gain of the integrator set in order to scale the sensitivity of flux integrator to an appropriate level.

The transducer will be implemented as an operational amplifier circuit. This will be a simple and effective method of conditioning the signal before passing into the controller. The next sections describe how the operational amplifier circuit was derived to meet the above specifications.

A.3.1.1. Derivations of Op Amp System from Transfer Function

It is necessary to represent the transfer function:

$$F_i = \frac{s}{\left(s^2 + 1.4\omega_i s + \omega_i^2\right)}$$

As a block diagram of the form:

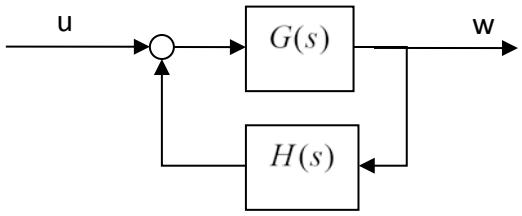


Figure A-34 feedback loop block diagram

The transfer function of the block diagram can be represented as:

$$F_i' = \frac{G(s)}{1 + GH(s)}$$

$F_i(s)$ can be manipulated algebraically into the form of $F_i'(s)$ in the following way:

$$F_i = \frac{s}{\left(s^2 + 1.4\omega_i s + \omega_i^2\right)}$$

$$= \frac{\frac{1}{s}}{1 + 1.4\omega_i \frac{1}{s} + \omega_i^2 \frac{1}{s^2}}$$

Let $G(s) = \frac{1}{s}$

Therefore:

$$F_i = \frac{G(s)}{1 + G(s)1.4\omega_i + G(s)\omega_i^2 \frac{1}{s}}$$
$$= \frac{G(s)}{1 + G(s)\left(1.4\omega_i + \omega_i^2 \frac{1}{s}\right)}$$

Let $H(s) = 1.4\omega_i + \omega_i^2 \frac{1}{s}$

Therefore:

$$F_i = \frac{G(s)}{1 + G(s)H(s)}$$

A.3.1.2. OpAmp representation of G(s)

As derived:

$$G(s) = \frac{1}{s}$$

In OpAmp circuit form, this requires a simple integrator circuit, as shown in the Figure A-32:

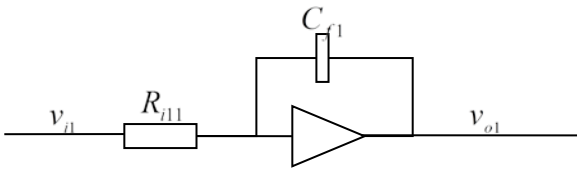


Figure A-35 OpAmp integrator diagram

The transfer function is found using complex impedance analysis:

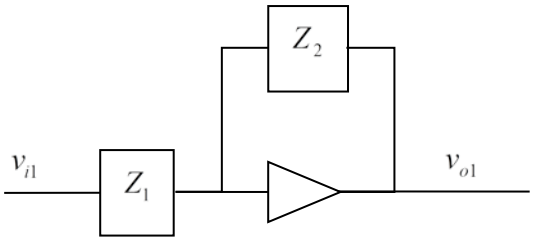


Figure A-36 Impedance Analysis

Where

$$G(s) = \frac{v_{o1}}{v_{i1}} = -\frac{Z_2}{Z_1}$$

So:

$$Z_2 = \frac{1}{C_{f1}s} \text{ and } Z_1 = R_{i1}$$

Therefore

$$G(s) = -\frac{\frac{1}{C_{f1}s}}{R_{i1}} = -\frac{1}{R_{i1}C_{f1}s}$$

As the overall gain is 1, it can be said that:

$$1 = \frac{1}{R_{i1}C_{f1}}$$
$$R_{i1}C_{f1} = 1$$

A.3.1.3. OpAmp representation of H(s)

$$H(s) = 1.4\omega_i + \omega_i^2 \frac{1}{s}$$

This circuit can be considered to be a classic proportional plus integral configuration, represented as:

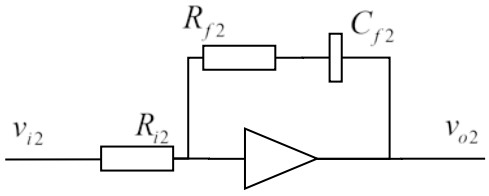


Figure A-37 OpAmp diagram of H(s)

Again, using complex impedance analysis:

$$H(s) = \frac{v_{o2}}{v_{i2}} = -\frac{Z_2}{Z_1}$$

In this case:

$$Z_2 = R_{f2} + \frac{1}{C_{f2}s} \text{ and } Z_1 = R_{i2}$$

So:

$$H(s) = -\frac{Z_2}{Z_1}$$
$$= -\frac{R_{f2} + \frac{1}{C_{f2}s}}{R_{i2}}$$
$$= -\left(\frac{R_{f2}}{R_{i2}} + \frac{1}{R_{i2}C_{f2}s}\right)$$

Therefore:

$$\frac{R_{f2}}{R_{i2}} = 1.4\omega_i \text{ and } \frac{1}{R_{i2}C_{f2}} = \omega_i^2$$

A.3.1.4. Full Circuit

Due to the inverting nature of OpAmps, a further unit gain is required on the feedback loop to ensure that the signs are consistent. The condition for this gain is that $R_{i3} = R_{f3}$.

Therefore, the full circuit is as shown in Figure A-31:

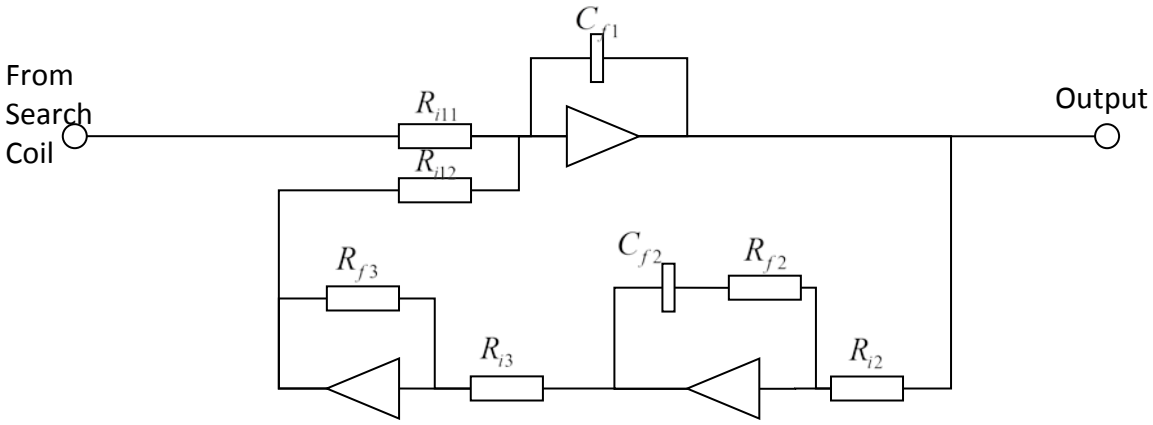


Figure A-38 Operational Amplifier Circuit for Self-Zeroing Integrator

In order for the summation to work, $R_{i11} = R_{i12}$.

Note: All op amps require wiring up shown in Figure A-39:

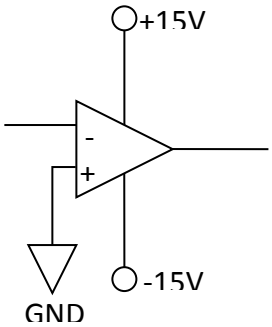


Figure A-39 power requirements of OpAmp

In addition to this, the input and output will require two connections, the second one (not shown on full diagram) will be connected to ground.

A.3.1.5. Derivation of component values

It was shown that for the OpAmp circuit to represent the transfer function, following criteria must be realised:

1. $R_{i11}C_{f1}=1$
2. $\frac{1}{R_{i2}C_{f2}}=\omega_i^2$
3. $\frac{R_{f2}}{R_{i2}}=1.4\omega_i$

The values will be chosen using a standard component values.

For $R_{i11}C_{f1}=1$:

Let $C_{f1}=2.2\mu F$

Therefore:

$$R_{i11}=\frac{1}{C_{f1}}=\frac{1}{2.2\times10^{-6}}=454k\Omega$$

Using the nearest standard value yields:

$$R_{i11}=470k\Omega$$

And also:

$$R_{i11}=R_{i12}=470k\Omega$$

For $\frac{1}{R_{i2}C_{f2}}=\omega_i^2$:

As previously defined:

$$\omega_i = 1.5 rad/s$$

Therefore:

$$\omega_i^2 = 1.5^2 = 2.25$$

So:

$$R_{i2}C_{f2} = \frac{1}{2.25} = 0.444$$

Let:

$C_{f2} = 1.0\mu F$

Therefore:

$$R_{i2} = \frac{0.444}{C_{f2}} = \frac{0.444}{1.0\times10^{-6}} = 444k\Omega$$

Using nearest standard value:

$$R_{i2} = 470k\Omega$$

For $\frac{R_{f2}}{R_{i2}} = 1.4\omega_i$:

Already defined are:

$$R_{i2} = 470k\Omega,\ \omega_i = 1.5 rad/s$$

So:

$$R_{f2} = 1.4\omega_i R_{i2} = 1.4\times1.5\times470000 = 987k\Omega$$

Therefore, selecting nearest standard value gives:

$$R_{f2} = 1M\Omega$$

In summary:

Component	Value
C_{f1}	$2.2\mu F$
R_{i11}	$470k\Omega$
R_{i12}	$470k\Omega$
C_{f2}	$1.0\mu F$

R_{i2}	$470k\Omega$
R_{f2}	$1M\Omega$
R_{i3}	$10k\Omega$
R_{f3}	$10k\Omega$

Table A-2 Component Values of Integrator

A.3.1.6. Calculate Modified Filter Coefficients

In order to do this, the expression for G(s) and H(s) will be calculated using expressions for coefficients based on the chosen component values. This can then be extrapolated to a term for the whole system. This will only provide an approximation to the true filter coefficients.

$$G(s) = -\frac{1}{R_{i1}C_{f1}s}$$

$$H(s) = \left(\frac{R_{f2}}{R_{i2}} + \frac{1}{R_{i2}C_{f2}s}\right)$$

(NB: the term for H(s) now includes the gain of -1).

The system can be represented as:

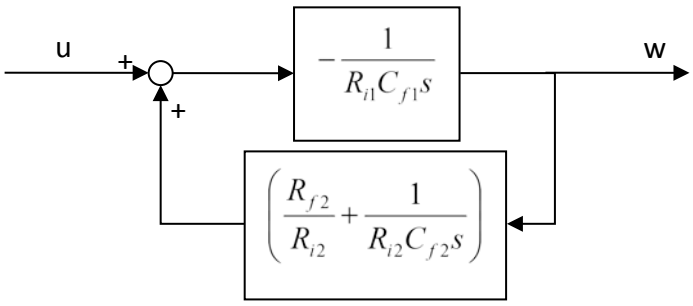


Figure A-40 Block Diagram of transducer circuit

So let:

$$F_i^{'}(s) = \frac{G(s)}{1 - G(s)H(s)}$$

$$= \frac{-\frac{1}{R_{i1}C_{f1}s}}{1 + \frac{1}{R_{i1}C_{f1}s}\left(\frac{R_{f2}}{R_{i2}} + \frac{1}{R_{i2}C_{f2}s}\right)}$$

$$= -\frac{\frac{1}{R_{i1}C_{f1}s}}{1 + \frac{R_{f2}}{R_{i1}R_{i2}C_{f1}s} + \frac{1}{R_{i1}C_{f1}R_{i2}C_{f2}s^2}}$$

$$= -\frac{\frac{1}{R_{i1}C_{f1}}s}{s^2 + \frac{R_{f2}}{R_{i1}R_{i2}C_{f1}}s + \frac{1}{R_{i1}C_{f1}R_{i2}C_{f2}}}$$

It is now possible to substitute the chosen component values into the expression:

$$F_i^{'}(s) = -\frac{\frac{1}{R_{i11}C_{f1}}s}{s^2 + \frac{R_{f2}}{R_{i11}R_{i2}C_{f1}}s + \frac{1}{R_{i11}C_{f1}R_{i2}C_{f2}}}$$

$$= -\frac{\frac{1}{470k \times 2.2\mu}s}{s^2 + \frac{1M}{470k \times 470k \times 2.2\mu}s + \frac{1}{470k \times 2.2\mu \times 470k \times 1.0\mu}}$$

$$= -\frac{0.967s}{s^2 + 2.058s + 2.058}$$

Remember that:

$$F_i = \frac{s}{\left(s^2 + 1.4\omega_i s + \omega_i^2\right)}$$

Substitute in values for ω_i:

$$F_i = \frac{s}{s^2 + 2.1s + 2.25}$$

A.3.1.7. Bode Plot Comparison

The following figures show the bode plot of the system with the desired filter coefficients (represented by F(s)) and the actual filter coefficients (Represented by F'(s)) based on the nearest available component values. The purpose of these plots are two fold. Firstly, it allows a comparison of the two systems to identify any major differences between the two. Secondly, it provides values in terms of gain and phase in order for testing purposes.

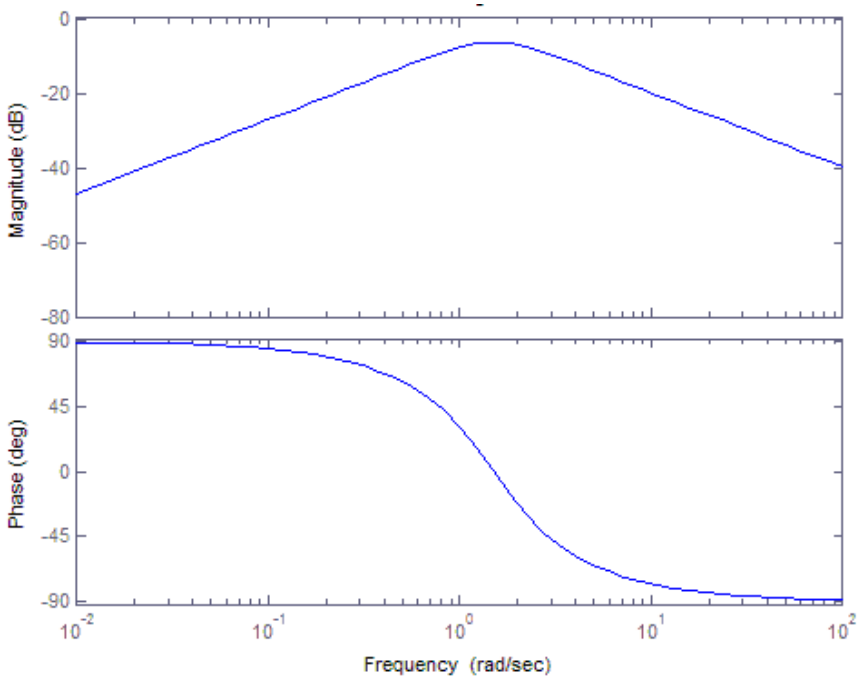


Figure A-41 Bode Plot of F(s)

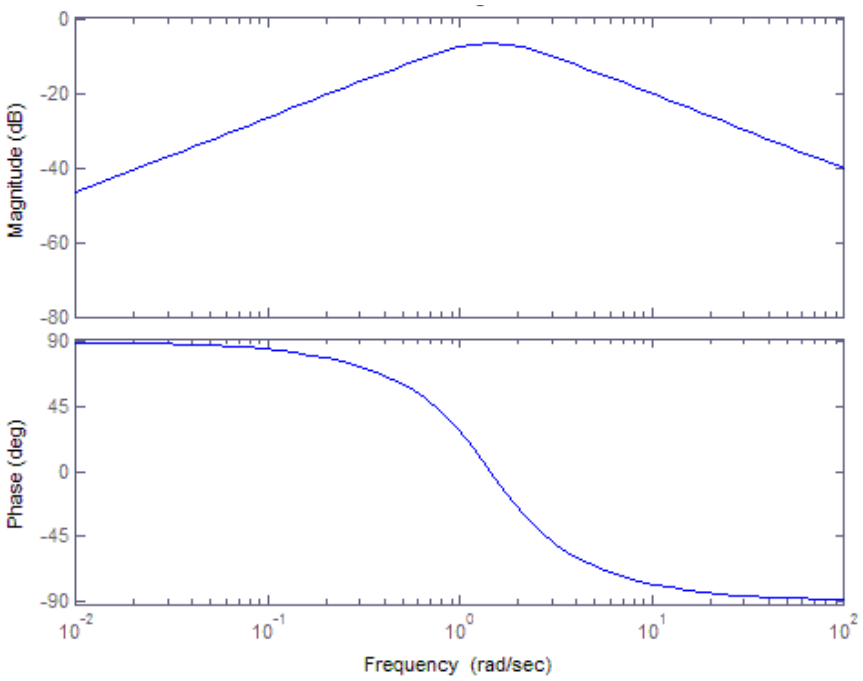


Figure A-42 Bode Plot of F'(s)

It can be seen that both plots are practically identical, and proves that the component values selected are satisfactory for the filter.

A.3.1.8. Scaling

The process of converting the flux produced by the magnet into a signal voltage is represented mathematically in the block diagram in Figure A-43. This is a development of the diagram discussed earlier.

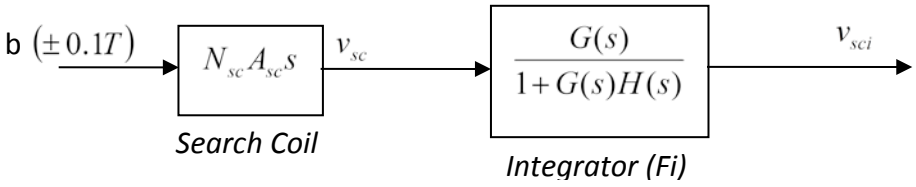


Figure A-43 Signal scaling considerations

Where v_{sc} is the search coil voltage and v_{sci} is the voltage produced by the search coil integrator.

The overall low frequency gain of the system can be represented as:

$$G_{sc} = N_{sc}A_{sc}G(s)$$

The sensitivity required for the system is 20V/T. This is because it is expected for the flux to vary by +/-0.1T, and the voltage to be measure between +/-2.0V.

Therefore it can be said that:

$$G_{sc} = N_{sc}A_{sc}G(s) = 20V/T$$

This formula is true if the search coil is made form a single coil of wire. However the search coil in this application is formed by a pair of wire windings embedded in each of the 2 polefaces of the magnet. The configuration of the search coils are shown in Figure A-44. As this system is design with redundancy in mind, 2 independent search coils exist.

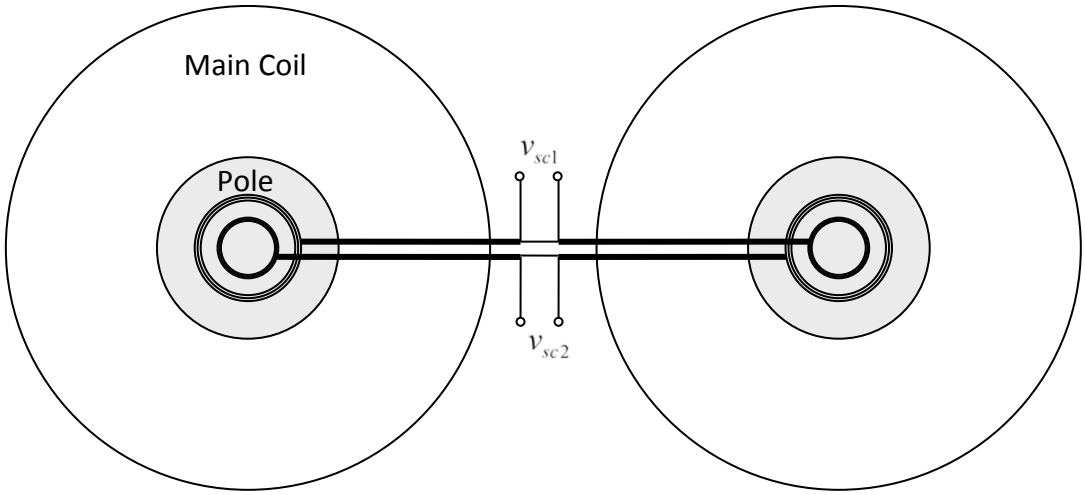


Figure A-44 Flux coil connections

Each of the two search coils are formed using an inner coil and an outer coil. Therefore the equation for the gain of the system must be modified as follows:

$$\begin{aligned} G_{sc} &= N_{sc}A_{sc}G(s) \\ &= (N_{sci}A_{sci} + N_{sco}A_{sco})G(s) \end{aligned}$$

Where the suffix ‘i’ indicates an inner coil, and ‘o’ indicates an outer coil

As the number of turns in each of the windings is the same:

$$G_{sc} = N_{sc}(A_{sci} + A_{sco})G(s)$$

These areas as measure directly from the poleface were found to be:

$$A_{sci} = \frac{0.027^2 \pi}{4} = 5.726 \times 10^{-4}$$
$$A_{sco} = \frac{0.017^2 \pi}{4} = 2.270 \times 10^{-4}$$

It is also known that the number of turns in each coil is 200. Therefore:

$$G_{sc} = N_{sc}(A_{sci} + A_{sco})G(s)$$
$$G_{sc} = 200(5.726 + 2.270) \times 10^{-4} G(s)$$
$$G_{sc} = 200 \times 7.996 \times 10^{-4} G(s)$$
$$G_{sc} = 0.1599 G(s)$$

The value required for the flux integrator gain can now be calculated:

$$20 = 0.1599 G(s)$$
$$G(s) = \frac{20}{0.1599}$$
$$\therefore G(s) = 125$$

A.3.1.9. Adjusting the Circuit

In block diagram form, the gain of the system can be increased by 125 in the following way:

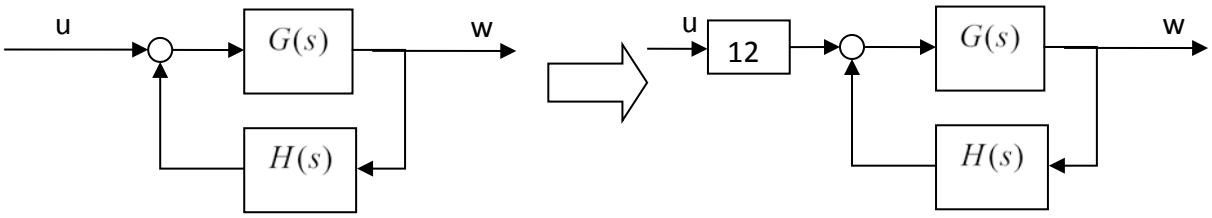


Figure A-45 Introducing scaling

The above figure shows that the dynamics of the system will be identical to the one designed, but a gain of 125 is now present. It is possible to introduce this gain of 125 by changing resistor R_{i11} as shown in the circuit in Figure A-46:

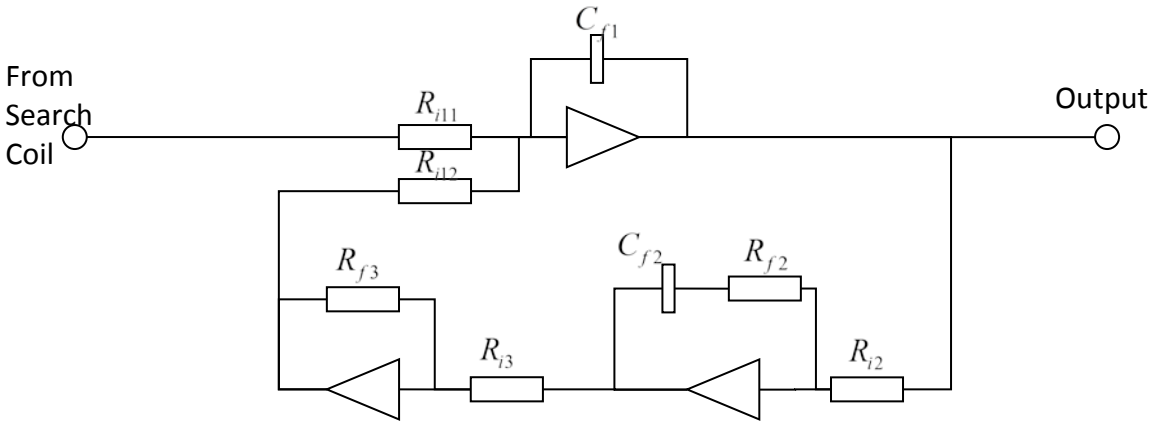


Figure A-46

It is to be noted that this change is to be made only to R_{i11} and not to R_{i12} as this will affect the H(s) part of the system as described in the sections above, and affect the dynamics of the filter.

The change in resister value can be described quite simply as:

$$R_{i11new} = \frac{R_{i11}}{125}$$

This essentially changes the gain of the forward looking Op amp on the input channel from 1 to 125.

So, the new resistor value is:

$$R_{i11new} = \frac{R_{i11}}{125} = \frac{454k}{125} = 3632\Omega$$

The value for R_{i11} used is the desired value, not the nearest preferred component value.

The nearest preferred component value to be implemented for R_{i11new} is:

$$R_{i11new} = 3k9\Omega$$

Therefore, the updated component table is:

Component	Value
C_{f1}	$2.2\mu F$
R_{i11}	$3k9\Omega$
R_{i12}	$470k\Omega$

C_{f2}	$1.0\mu F$
R_{i2}	$470k\Omega$
R_{f2}	$1M\Omega$
R_{i3}	$10k\Omega$
R_{f3}	$10k\Omega$

A.3.2. Component Selection and Design

A.3.3. Gap Transducer

The gap transducers installed on the rig are Sensagap SG10 produced by RDP Electronics. They are a non-contact displacement transducer that use a capacitive technique to measure the distance from the front ceramic face of the sensor to the target.

The actual output of the gap transducer will be a voltage in proportion to 0-15mm. This will have to be converted to +/- 5mm. Either that or the signal +/-10V will have to be reconsidered to represent the range 0-15mm.

A.3.4. Power Amplifier

These power amplifiers are designed to be driven by a 20kHz PWM signal from the controller, and output a 50V DC PWM drive signal to the magnet coils. Along with the circuit design, a 1.5kVA 50DC power supply was manufactured to provide the power source.

A.3.4.1. Circuit Diagram

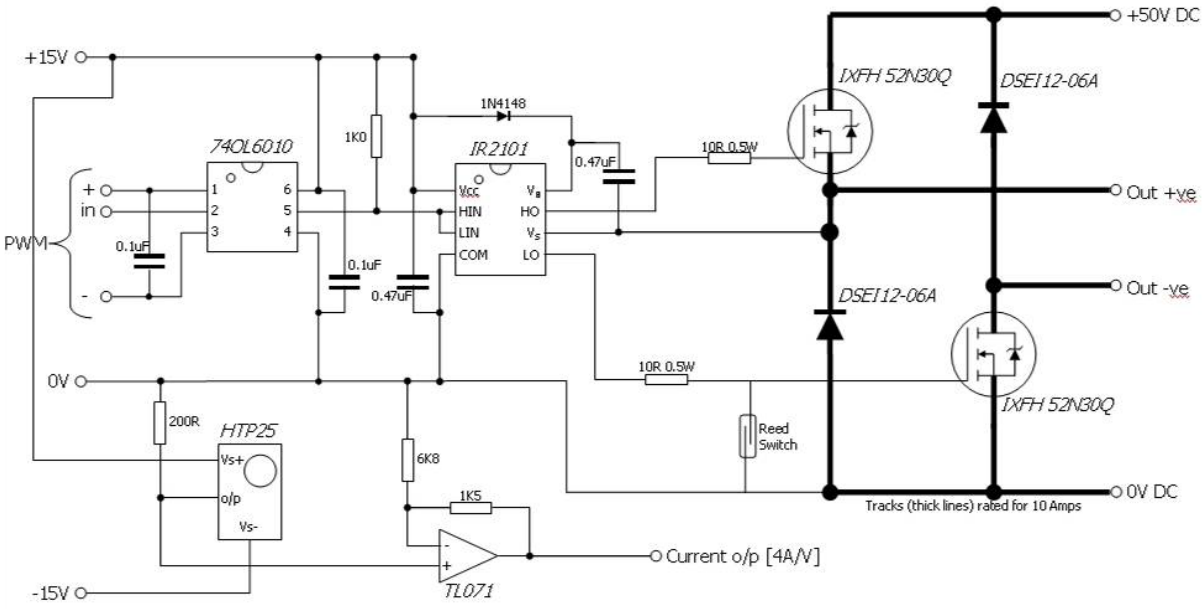
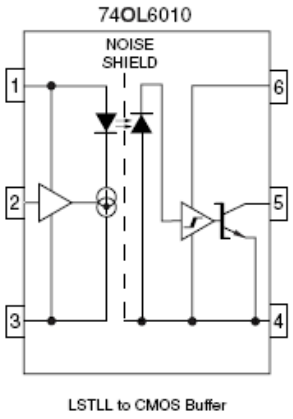


Figure A-47 Power Amplifier Circuit Diagram

This circuit is an adaptation of a previous design. The basics of the circuit are that it is a standard 2 quadrant H bridge power amplification design. It includes an opto-coupler in the form of a 74OL6010 chip and a IR2101 MOSFET driver.

A.3.4.2. Logic to Logic Optocoupler

This component is included to isolate the input signals from the power part of the circuit. The component used is a 74OL6010 logic to logic optocoupler.



The IC is essentially used as a driver for the power electronics and uses LEDs to isolate the input signals from the power section.

A.3.4.3. Internal Rectifier

The internal rectifier IR2101 is use to drive the MOSFETS. Figure A-48 shows the basic make-up of the driver circuit.

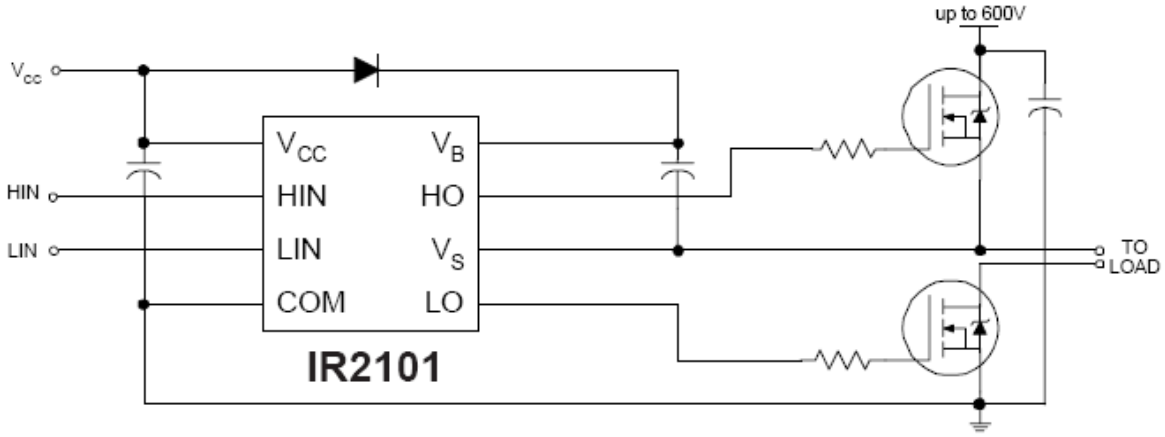


Figure A-48 Internal Rectifier Circuit

A.3.4.4. Hall Effect Current Transducer

The HTP25 is a Hall effect current transducer and is used to measure the current produced by the circuit. The out +ve wire (as shown in Figure A-34) is threaded through the eye of the transducer in order to excite the coil within it.

The original Hall Effect Transistor circuit is shown in theFigure A-49:

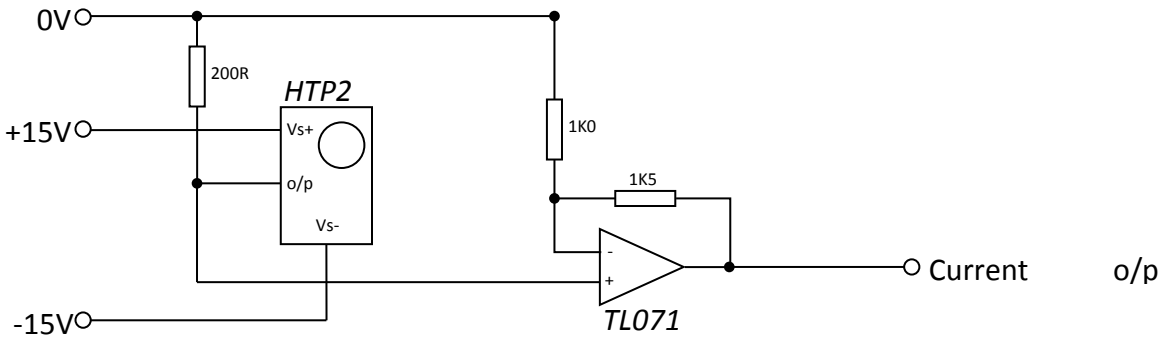


Figure A-49 Current transducer schematic

The output of the transducer is required to be input to an analogue to digital converter with a range of +/- 5V. As such, the output put through a non-inverting amplifier to achieve

the appropriate ratio. The resistor values were chosen as a result of the calculations derived below.

The required gain of the overall transducer will be a range of 20A to every 5V output and therefore requires a gain of 0.25 V/A. The block diagram in Figure A-50 represents the signal processing aspect of the above system:

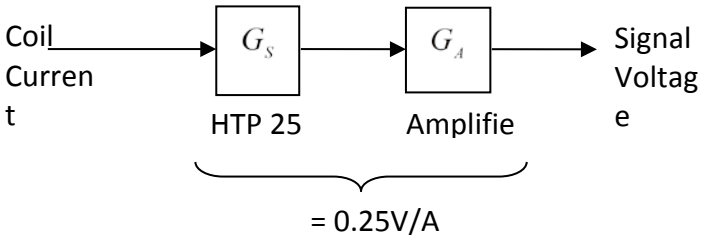


Figure A-50 Scaling considerations

Where:

G_S = Gain of Hall Effect Sensor (Volts/Amp)

G_A = Amplifier Gain (Volts/Volt)

As demonstrated in the above figure, but represented here mathematically,

$$\frac{V_I}{I} = G_s \times G_A = 0.25$$

The value of G_S is known from the use of the HTP25 data sheet, and is found to be:

$$G_S = 0.2$$
$$\therefore G_A = \frac{0.25}{0.2} = 1.25$$

The gain of a non-inverting amplifier can be expressed as:

$$G_A = 1 + \frac{R_1}{R_2}$$

In order to find the resistor values, we substitute in the desired value of gain:

$$1.25 = 1 + \frac{R_1}{R_2}$$
$$\frac{R_1}{R_2} = 0.25$$

By using preferred component values, the following were chosen.

$$R_1 = 1.5k\Omega$$
$$R_2 = 6.8k\Omega$$

This errs slightly to the lower side of the overall ratio, but still falls within acceptable parameters.

A.3.4.5. Power MOSFET

The MOSFET used is the IRFP260N produced by International Rectifier. It is a standard MOSFET with appropriate current and voltage capacities to drive the coils at 50V and 20A.

A.3.4.6. Reed Switch

During testing, it was found that there was the potential for the coil to draw 50A if a fault in the controller software arose. The latest circuit diagram, shown in Appendix C, includes a reed switch as an emergency current limiting device. The output +ve connection as shown in the diagram requires wrapping around the reed switch the correct number of times (3 for the component selected) such that a cut out is induced at around 15-20 Amps. The cut out works by pulling the MOSFET input low when the current is above the threshold. The circuit will then "limit cycle" until the current demand is reduced.

A.3.5. Power Supply

Table A-3 shows the list of functional components that will require power, and the details of this requirement.

Title	Description	#	Power connection ref:	Type (DC/AC)	Voltage (Volts)	Current (Amps)
Coil power amplifiers	Amplifies controller outputs to necessary drive signal	2	1	DC	+50	40
			2	DC	0	40
			3	DC	+15	
			4	DC	-15	
			5	DC	+5	
			6	DC	0	
Processing Module	Performs all functions for control purposes	N	1	AC	240	1.5

Network switch	Directs Ethernet packages	2	1	AC	240	
Flux Integrator	Integrates flux density signal to achieve flux	8	1	DC	+15	
			2	DC	0	
			3	DC	-15	
Gap Sensor	Measures airgap	4	1	DC	15	
			2	DC	0	

Table A-3 Power supply requirements

It is evident from the table above that there are 4 power sources required:

- 240V AC
- +50V DC
- +/- 15V DC
- +5V DC

The power to supply these components is to come directly from a standard 250V mains socket. Therefore, a power supply/converter unit will be needed to convert the 250V AC into the required DC values above. Table A-4 details these units.

Supply Rail	Detail of Device
+50V DC	Custom made Power Supply
+/-15V DC	Standard lab PSU
+5V DC	Standard lab PSU

Table A-4 Power Supply sources

A.3.5.1. Supply Configuration for Single Magnet Control

The project has to go through a staged commissioning process to develop the full four-corner control of the rig. One of the most significant early stages of the project is the control of 1 magnet of the rig in isolation. This phase requires a considerably less complicated supply configuration than that of the full installation.

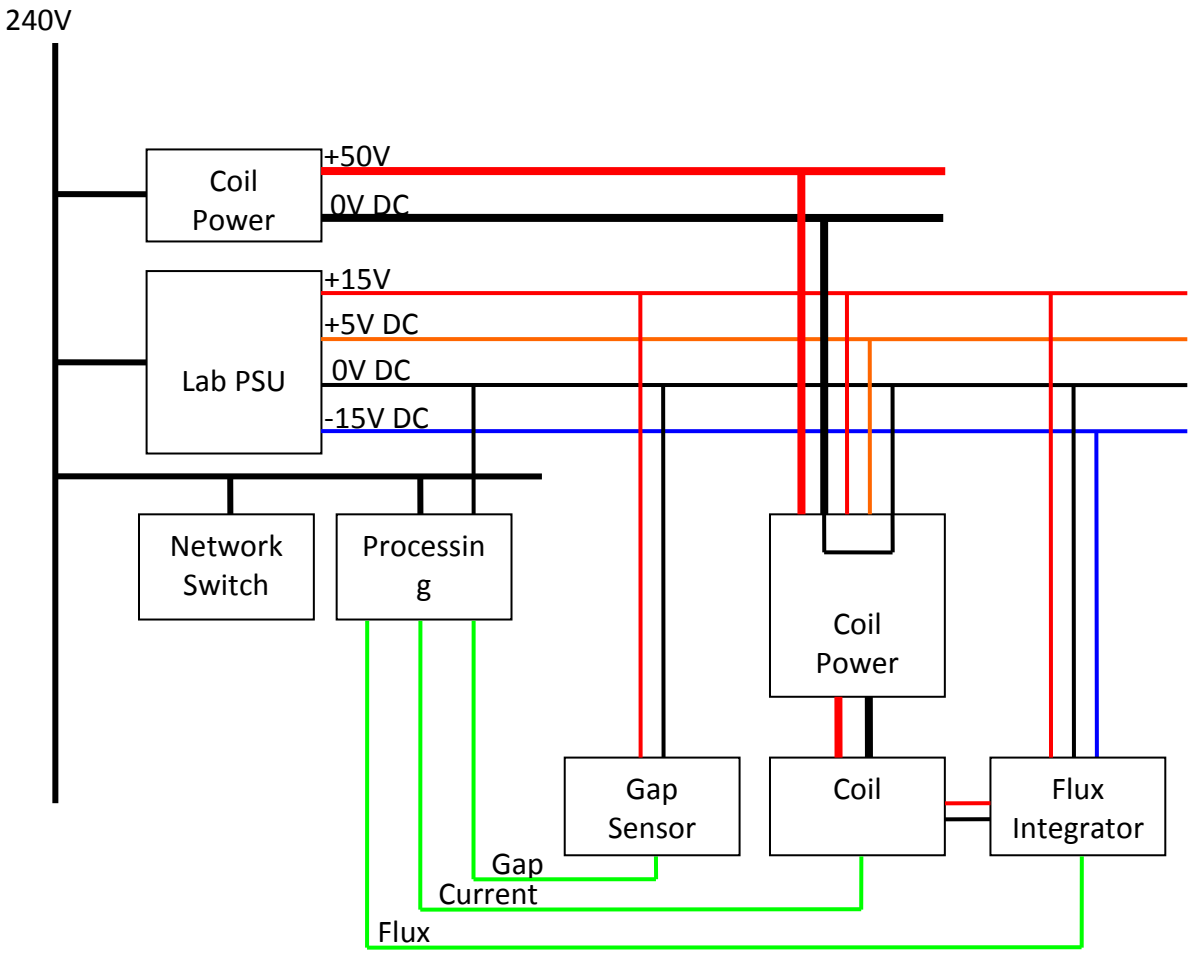
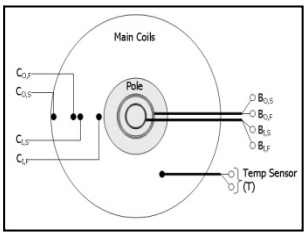
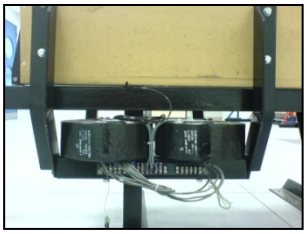


Figure A-51 Power Supply to system components

It can be seen how the scheme in Figure A-51 can be easily expanded to facilitate additional processing modules, sensors and coils, as long as the current is not exceeded by any of the sources.





APPENDIX B:

Requirements Compliance

B. Requirements Compliance

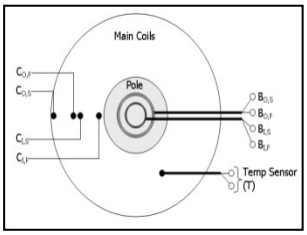
Req. #	Description	Compliance
Top Level System		
1	The IMA shall be able to display to the operator appropriate internal systems management information (e.g. current functional allocation, and current component health) for verification purposes	Results in Chapters 5 & 6 show screen shots of system configuration information
2	The IMA shall be able to display to the operator real-time information regarding the target platform (Maglev rig), such as current position	A Real-time demonstration was given to project supervisors
3	The IMA shall be capable of receiving control commands for the target platform	Results in Chapter 6 shows the system response to a changing setpoint command
4	The IMA shall perform the real time control functions necessary to maintain magnetic suspension of the Maglev rig	Results in Chapter 6 show a consistent, stable system response to airgap demand changes
5	The IMA shall allow user to inject fault signals into the system	Chapter 6 details a number of tests that involve injected a fault signal into the system
6	The IMA shall perform some identified techniques in order to manage the fault	Chapter 4 discusses the methods by which this can be achieved. Chapter 6 shows the systems manages faults by
7	The IMA should maintain service in the presence of fault if possible	Results in Chapter 6 show that through a series of faults injected, the system maintains service
8	The IMA shall provide a tangible output of the systems management actions during this response	Screenshots of the user interface recorded in Chapter 6 show how internal systems management states are communicated
IMA Top Level Requirements		
9	The design of the system shall follow fundamental IMA principles where appropriate	Chapter 4 discusses in detail how this was realised
10	Each IMA processing module shall follow the three layer stack architecture	Details of the hardware/software architecture can be found in Chapter 4
11	The hardware and software shall be loosely coupled such that, within a limited scope, a change in either shall not infer a change in the other	Details of the hardware/software architecture can be found in Chapter 4

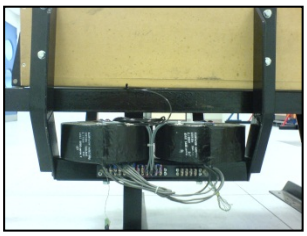
Req. #	Description	Compliance
12	The operating systems layer, or middle layer, shall manage the availability of the hardware resources to the application layer	Details of the hardware/software architecture can be found in Chapter 4
13	The operating systems layer shall manage the arising of a limited number of faults within the system	Details of the hardware/software architecture can be found in Chapter 4. Results of experiments supporting this can be found in Chapter 6.
14	A tangible output shall be created showing the management of the fault	Screenshots in Chapter 6 show how information was conveyed to the user
15	The service provided by the system shall not be interrupted during the management of the fault	Time-response graphs throughout Chapter 6 show this to be the case
IMA Hardware Requirements		
16	The hardware shall be capable of accommodating the required system functionality in terms of processing resource, communication resource and I/O expectations	Details of the hardware/software architecture can be found in Chapter 4
17	To a limited scope, hardware shall be generic and a change in a hardware component will not affect the fundamental design of the rest of the system	Details of the hardware/software architecture can be found in Chapter 4
18	The hardware shall have data acquisition capabilities where appropriate	Details of the hardware/software architecture can be found in Chapter 4
19	The hardware shall have a networking capability of deterministic communications	Results of Chapter 5 show how the communication structure is deterministic and repeatable
20	The hardware shall be expandable to allow additional hardware capabilities to be included	Results in Chapter 5 show how the number of modules in the system can be easily accommodated
IMA Middleware Requirements		
21	The middleware shall manage the administrative aspects of the IMA across all modules, inclusive of start-up, communications, configuration and faults	Chapter 4 describes these processes in detail
22	The middleware shall manage accessibility of hardware resources to applications	Chapter 4 and Appendix D describe the processes

Req. #	Description	Compliance
23	The middleware shall enable the execution of applications and communications to be performed in a deterministic manner	Chapter 4 discusses how the communications and applications are managed system wide in a deterministic fashion
24	The middleware shall allow the software and hardware disparate such that a change in one should not infer a change in the other	Chapters 5 & 6 show a number of tests where different applications were defined and implemented on the hardware resources
25	The middleware shall prevent any unwanted interaction between applications to a reasonable degree	Chapter 4 describes the systems architecture and Chapter 6 shows that functions were able to be switched on and off without cross interference
26	The middleware shall share systems resources between applications	The deterministic results in Chapters 5 & 6 show that system resources have been partitioned effectively
27	The middleware shall partition applications from all others	Chapter 4 describes the systems architecture and Chapter 6 shows that functions were able to be switched on and off without cross interference
28	The middleware shall ensure that applications are executed in order to meet their timing constraints	The deterministic results in Chapters 5 & 6 show that system resources have been synchronised effectivel
29	The middleware shall be modular and portable	Chapter 4 discusses the limits of this requirement with the chosen development environment
Configuration Management Requirements		
30	The configuration manager shall manage assignment of applications to resource following application assignment requirements	Experiments conducted in Chapters 5 & 6 shows successful application assignment for a number of different cases
31	The configurations generated shall ensure the applications can execute in a deterministic fashion	Experiments conducted in Chapters 6 shows successful application assignment for a deterministic system
32	The configuration/reconfiguration tasks shall be performed in a timely manner	The time response graphs in Chapter 6 show the time required for a reconfiguration to occur
33	The process of configuration/reconfiguration shall not disrupt the application execution flow	Time responses in Chapter 6 show that the control algorithms continue to execute during reconfiguration activities
34	To recalculate a new configuration based on constraints from fault signals	Results in Chapter 6 show successful reconfiguration in response to fault signals

Req. #	Description	Compliance
35	To Implement a new configuration without interrupting the service	Time response graphs in Chapter 6 show that continual service is provided
Fault Management Requirements		
36	The system shall be capable of detecting simple faults in various system components, such as applications or hardware modules	Chapter 4 shows how faults can be monitored, detected and communicated to higher level functions
37	The system shall be capable of handling the occurrence of certain errors	Errors are defined in Chapter 4 and demonstrated in Chapter 6.
38	The system shall be capable of recovering from error by initiating the reconfiguration mechanism with new constraints	System management schematics presented in Chapter 6 show how this was achieved.
39	Where possible, the system shall confine faults such that their occurrence does not propagate system wide	The system design discussed in Chapter 4 considers how errors are contained
40	The system shall maintain a log recording the error that has arisen and the resultant action taken	Data presented in Chapter 6 was supported by a text activity log captured during testing
Communications Management Requirements		
41	It shall provide deterministic communications between applications for assigned configuration	Chapter 5 demonstrated the deterministic communication capabilities of the system
42	It shall ensure that the network communications methodology is transparent to the applications	Chapter 4 discussed how the blueprint is formed and translated into a deterministic network description.
43	It shall provide communications between systems management components	Chapter 4 discusses how systems management components communicate
44	It shall Ensure that communications of systems management do not impact upon the deterministic communications	Chapter 4 discusses the time partitioning of the network such that this does not occur
45	It shall ensure smooth transition between configurations	the time responses in Chapter 6 shows how deterministic processes continue during transitions
IMA Applications Requirements		
46	They shall perform a specified task	Discussed in Chapter 4, demonstrated in Chapters 5 & 6

Req. #	Description	Compliance
47	They shall be controllable by the system manager	Discussed in Chapter 4, demonstrated in Chapters 5 & 6
48	They shall communicate with other applications via the communications structure	Discussed in Chapter 4, demonstrated in Chapters 5 & 6
49	They shall have an assignment specification	Discussed in Chapter 4.
Graphical User Interface Requirements		
50	The GUI shall provide an input capability for the demand of the Maglev rig	Demonstrated to project supervisors
51	The GUI shall provide an output of sensor data from the Maglev rig	Demonstrated to project supervisors. Supported by recorded data presented in Chapter 6.
52	The GUI shall provide a test input capability	Results of test inputs are captured in Chapter 6
53	The GUI shall display the configuration of the system in a tangible manner	Screenshots in Chapter 6 show how this was achieved
54	The GUI shall highlight the process flow of the functions	Screenshots in Chapter 6 show how this was achieved
55	The GUI shall display the health of the system components	Screenshots in Chapter 6 show how this was achieved
56	The GUI shall display detailed information about each component	Screenshots in Chapter 6 show how this was achieved
57	The GUI shall provide a fault injection capability	Demonstrated to project supervisors. Supported by recorded data presented in Chapter 6.
Maglev Rig Requirements		
58	The Maglev rig shall be controllable and respond as expected to inputs.	The design of the experimental setup is described in Appendix C. Results graphs in Chapter 5 and 6 show system under control.
59	The Maglev rig shall provide clean, repeatable signals	The design of the experimental setup is described in Appendix C
60	The Maglev rig shall allow faults to be injected into it	The design of the experimental setup is described in Appendix C





APPENDIX C:

Magnetic Levitation Experimental Rig

C. Magnetic Levitation Experimental Rig

C.1. Introduction

This chapter will describe the design, build and test of the magnetic levitation (maglev) rig that is used as a target platform for the IMA installation produced.

The chapter will begin by discussing the relevancy of using a maglev rig as a test platform for an avionics system. It will then move on to talk about the rig itself, and the demands that the rig has on a control system, by the development and analysis of a mathematical model. It is also in the chapter that a controller is derived in order to maintain magnetic suspension. The final sections of chapter detail the hardware aspects of implementing the controller upon the rig inclusive of verification and test.

C.2. Magnetic Levitation as a Concept Demonstrator

The purpose of the overall project is to produce a concept modular avionics system that can maintain operational service in the presence of faults. In order to prove that the system can perform this function whilst maintaining real time control of a high demand control system, a test platform of some form was required for the IMA to drive.

It was conceived that a maglev rig (as a legacy of a previous project) would be useful to this IMA investigation due to fundamental similarities between the control law problem between maglev airgap control and modern aircraft stability. It can be concluded that both systems require:

- Real time multi input, multi output control for open loop unstable
- Computing for items such as health management, navigation, etc
- Certifiable systems that perform to a high level of reliability
- Lifecycle management in terms of managing obsolescence and changing requirements

The main difference is of course that the maglev system will remain attached (albeit magnetically) to the guide rails whereas an aircraft is free to move in 6 degrees of

freedom. The complexity and risk of a project is significantly reduced when using a maglev rig compared to an airborne system.

It is also interesting to note that an IMA technology is directly beneficial to commercial maglev systems, even though it is currently an unrealised solution in this area.

C.3. Electromagnetic Levitation

Available for use for this investigation is a 200kg capacity Magnetic levitation (Maglev) test rig facility. It is considered that the Maglev rig will be an appropriate demonstrator as it requires a high demand multi-input, multi-output real time controller to maintain stability of a safety critical open-loop unstable system, synonymous to modern, fighter-style aircraft. The application of an IMA to this situation will be proof that the architecture is capable of deterministic actions whilst reducing the technical risk and certification issues of an initial implementation upon an air platform.

In addition to this, many definitions of IMA suggest that the architecture should have platform independency such that the system should be transferable between applications with minimal change.

This section therefore summarise the investigation of relevant literature regarding the operating requirements of Maglev vehicles.

C.3.1. Maglev background

The concept of Maglev, or electromagnetic levitation, as a means to provide propulsion and levitation for trains has been around for a number of years. The first commercial maglev system was implemented at Birmingham International Airport in order to transport passengers from the airport terminal to the nearby railway station (R. Goodall, 1985). Maglev has an advantage over wheeled vehicles as it generates considerably less friction. This means that it can be used in situations where very low friction or very high speed is required.

C.3.2. Maglev Control Systems

Goodall (R. Goodall, 1985) suggests a number of different configurations for arranging the electromagnet with the track. The following simplified diagram represents the basic configuration for a simple single actively controlled magnet on a Maglev vehicle.

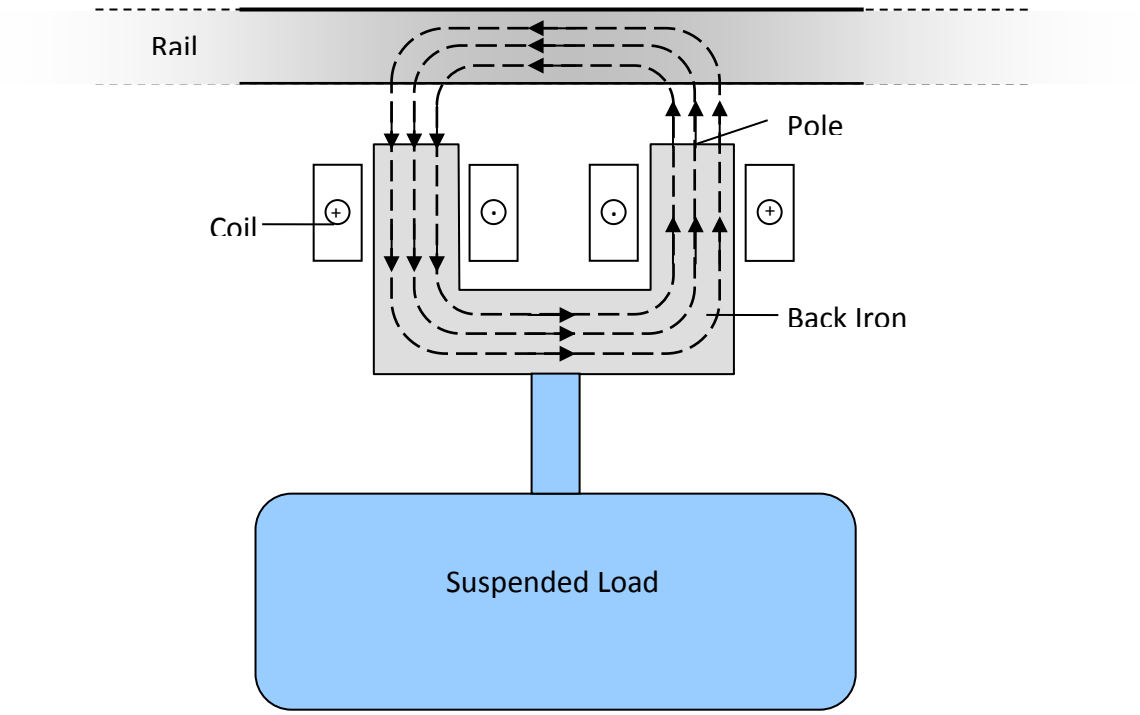


Figure 4-1 Basic Electromagnet (Goodall 1985)

Figure 4-1 represents the railway bogie as a suspended load. In reality, this load will be mounted above the track, but fixed to the magnets on the lower side of the track. It is suggested that a more accurate description of the process is electromagnetic suspension, rather than electromagnetic levitation.

The problem remains of developing a suitable robust control solution for this electromagnet configuration. Much work has been previously performed on this topic by Goodall et al (R M Goodall, 2004; R. Goodall, 1985; Roger M. Goodall, 2000). These papers state that an appropriate method to control the airgap between the pole face and the track is to use an inner control loop to maintain stability. It is mentioned that three possible variables are available to the designer that may be used as an internal loop,

namely applied voltage, current and flux density. It has been shown that using flux density as an inner control loop provides favourable response characteristics.

C.3.2.1. Suspension

When Maglev was first conceived, it was thought that a particularly smooth ride could be achieved because of the lack of contact between the pole face and the track (R M Goodall, 2004). The reality is that the individual magnets will respond to small deviations caused by track irregularities, generating vibrations and a potentially poor ride quality. Electromagnets in a maglev vehicle must do more than just suspend the load, they must provide appropriate suspension characteristics. These requirements are defined as:

- To support the load of the vehicle
- Guide the vehicle such that it follows the track profile
- Isolate the vehicle body and its passenger from vibrations caused by track irregularities

A solution to the suspension problem is suggested at a later stage in this paper using the Maglev system at Birmingham airport as a working example. It is shown that to provide appropriate suspension characteristics to each electromagnet, a complementary filter arrangement is required, as shown in the following diagram.

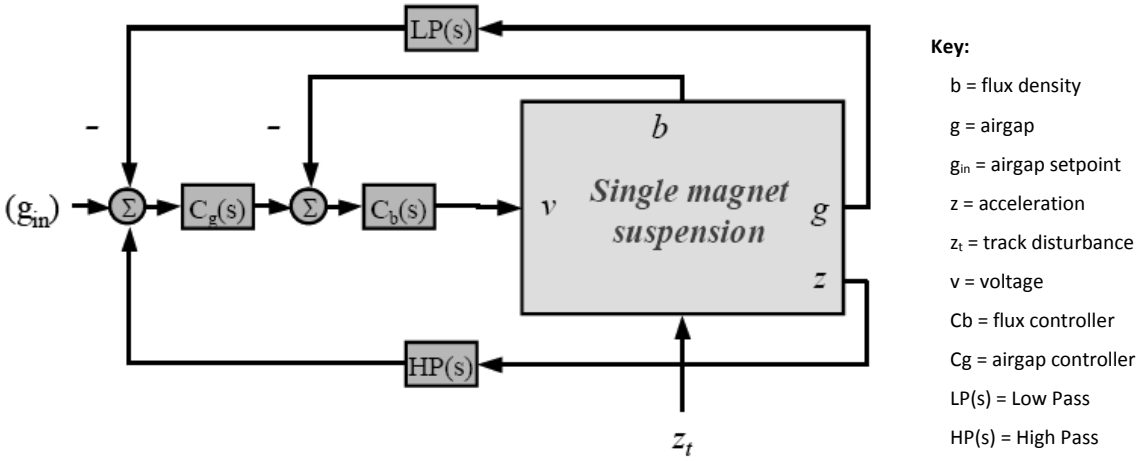


Figure 4-2 Complimentary Filter Control (Goodall 2004)

This scheme implements a Low Pass filter (shown as LP(s) in) on the airgap feedback in order to provide guidance to the track. A High Pass fliter (shown as HP(s) in) is

implemented on the accelerometer feedback to provide isolation from track disturbances. In general, this means that the system will maintain minimal changes in vertical acceleration, whilst providing good track following properties.

C.3.2.2. Whole Vehicle Controller

The main problem to solve is not the control of an individual magnet, but the suspension of an entire vehicle consisting of a number of electromagnets. Along with this problem will arises issues such as structural coupling. A solution to this problem is to transfer the four control loops to a 'modal controller', essentially controlling the vehicle in terms of pitch, roll and yaw (R M Goodall, 2004). The idea is that these three variables are mainly uncoupled, and thus can be dealt with independently.

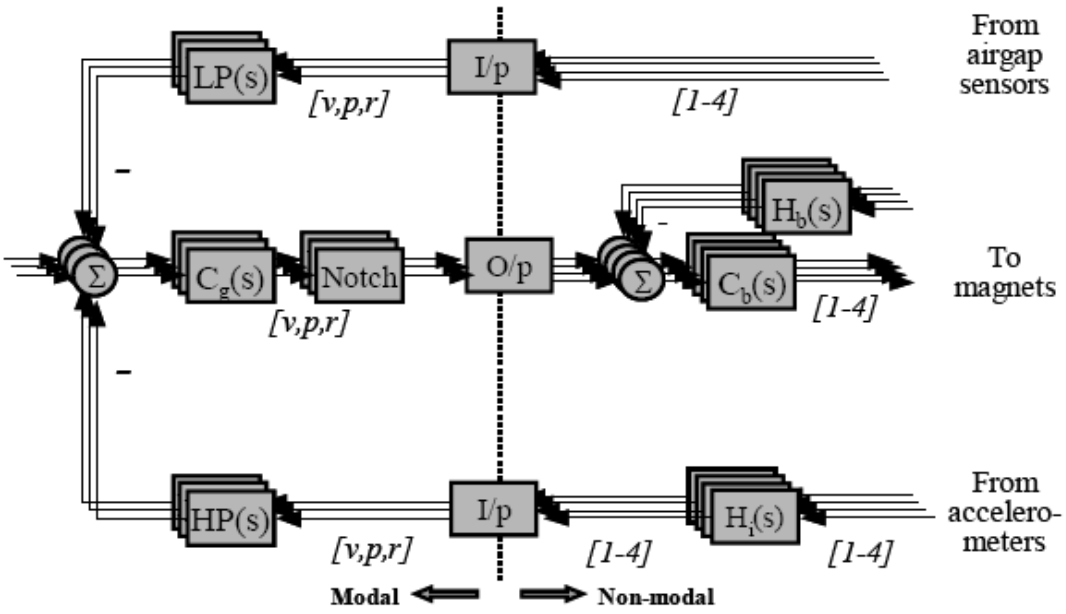


Figure 4-3 Vehicle Controller - Modal form (Goodall 2004)

This method has been proved to be successful and robust, using classical control solutions. It should be possible to extend the ideas of a LQ state-feedback controller (Michail, 2009).

C.4. Overview of the Experimental Rig

The maglev test rig is a legacy of a previous project and as such elements such as sensors and actuators were already present.

Figure 4-4 Maglev Test Rig

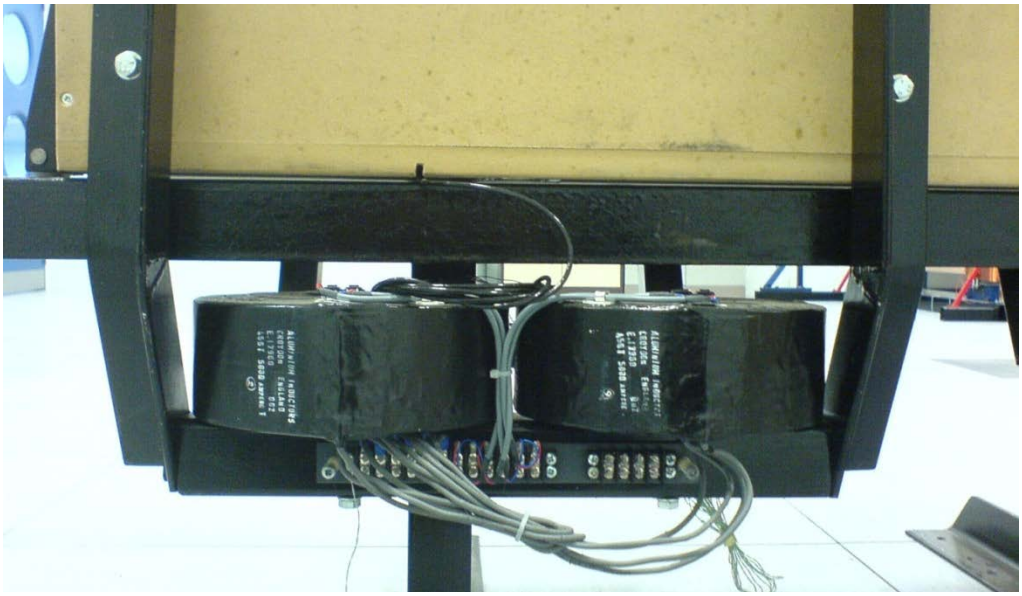


Figure 4-5 View of a Single Magnet

Each magnet is comprised of two cores each of which are dual wound in that each magnet can be stimulated from an independent control unit. Also, each pole face contains two

'search coils' for sensing magnetic flux, as shown in Figure 4-6. This means that each magnet is designed with a redundant actuation channel available.

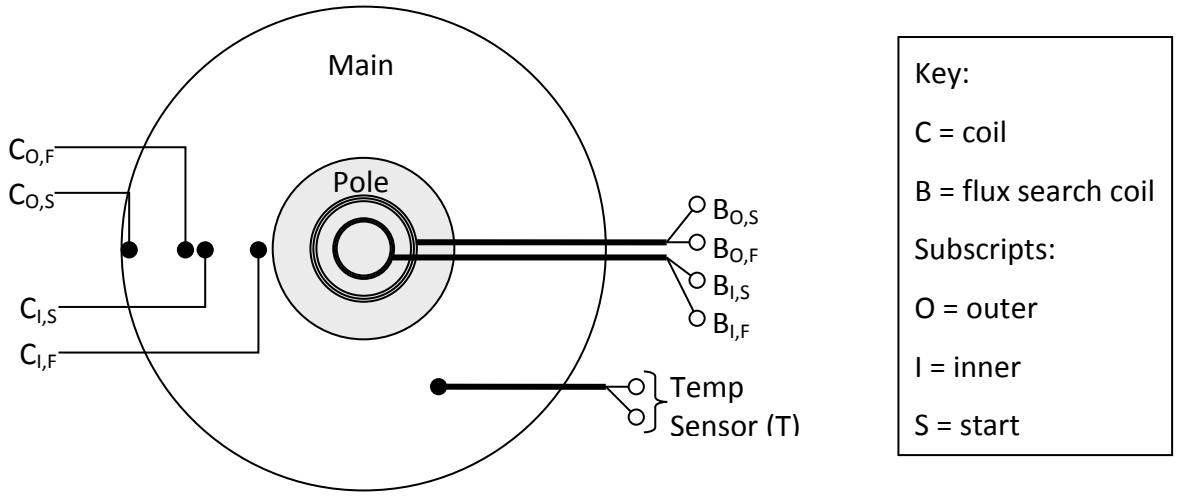


Figure 4-6 Windings around the magnet poles

C.5. Control Design and Implementation

The control strategy for the test rig can be resolved by developing a controller for a single magnet, then replicating this controller for each magnet on the vehicle. This can be resolved into a full vehicle controller (Section 4.3.2.2) as necessary. The control for a single magnet is performed using an inner loop controlling magnetic flux, and an outer loop controlling the air gap, as shown in Figure 4-6.

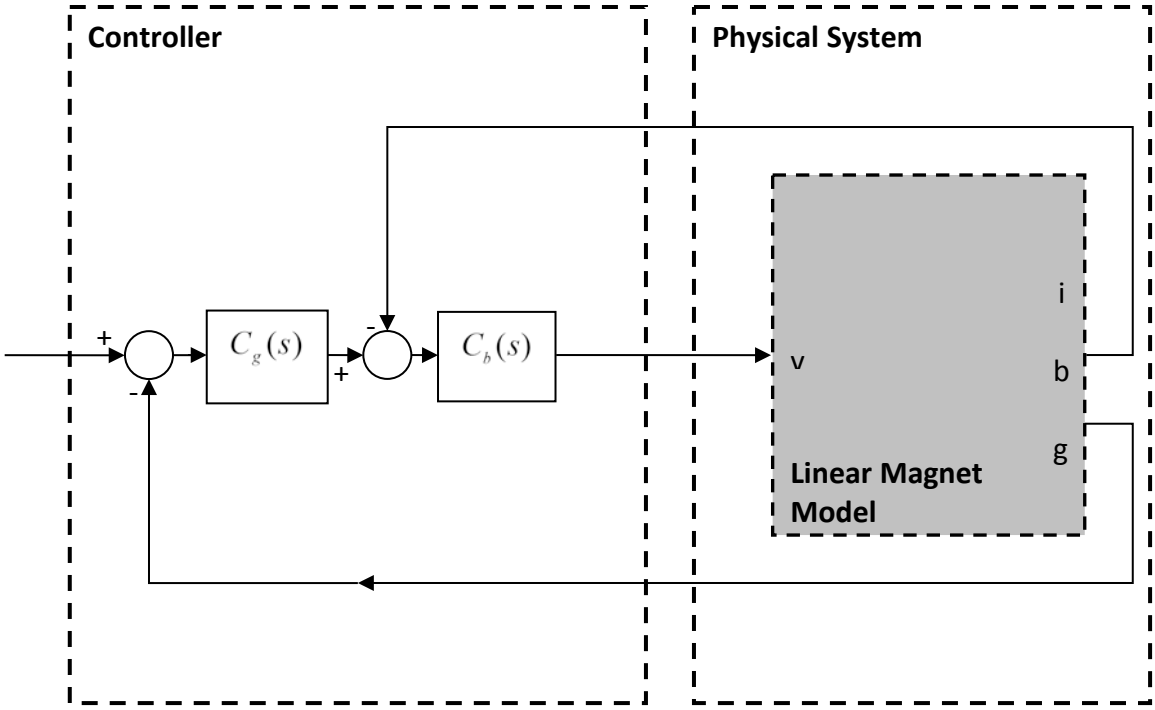


Figure 4-7 Control Scheme for a Single Magnet

A derivation for the linear magnet model is performed by considering small deviations around a nominal operating airgap. A simplified block diagram representation of the model is shown in Figure 4-8 which is taken from a full derivation that can be found in Appendix A. This shows the relationship of the input voltage $'v'$ (defined as a small perturbation around the operating point $'V_0'$) to airgap change $'g'$ (defined as a small perturbation around the operating point $'G_0'$).

The operating parameters of each magnet are found in Table 4-1.

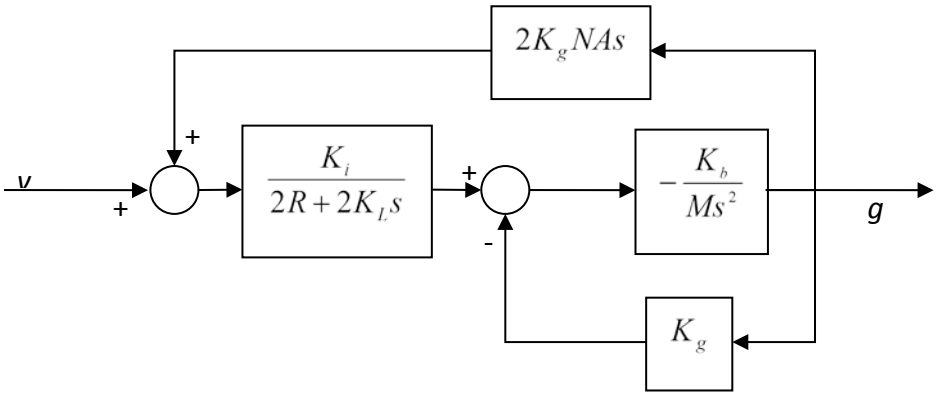


Figure 4-8 Simplified Block Diagram of the Linear Magnet Model

Description	Parameter Name	Value
Operating Voltage	V_0	11V
Operating force of each magnet	F_0	500N
Operating Airgap	G_0	0.01m
Operating flux density	B_0	0.5T
Operating current	I_0	11A
Poleface Area (single magnet)	A	0.0025m2
Number of turns (single coil)	N	456
Coil Resistance (single coil)	R	0.5 Ohm
Coil inductance (single coil)	L	2.59 mH
B_0/I_0	K_i	0.0455 T/A
B_0/G_0	K_g	50 T/m
$2F_0/B_0$	K_f	2000 N/T
$L+NAK_i$	K_L	0.0545 H

Table 4-1 Summary of derived Maglev system variables

The outer gap loop controller $C_g(s)$ and the inner loop flux controller $C_b(s)$ shown in Figure 4-6 are defined by the transfer functions:

$$C_b(s) = 700\left(\frac{0.008s+1}{0.008s}\right)$$

$$C_g(s) = 10\left(\frac{0.05s+1}{0.01s+1}\right)$$

The inner flux controller requires digitising to operating at around 1000Hz. The outer gap loop controller requires operation at 20Hz or faster.

C.6. Interfacing with the IMA

It is clear that the interface between the actuators and sensors of the Maglev Rig and the IMA architecture (in terms of software and hardware) require definition. In simple terms, the IMA will capture gap and flux signals from the magnet coils and calculate the control commands to the actuators. The control commands are presented in low-power signals that will require amplification to drive the coils of the magnet.

The design of the IMA suggests that any real time control strategy implemented will have to take into account the distributed nature of the processing modules. The sensed values may have to be communicated across the network to the control algorithm, for the command value to be returned. It is therefore sensible that the flux control loop should not be required to operate across the network as the time period in which it is required to operate is very small (0.001s) and will not allow enough time to transmit sufficient data amongst other network traffic. The gap control loop requires a slower time period (0.05s) and can operate across the network. Figure 4-9 describes this allocation of function.

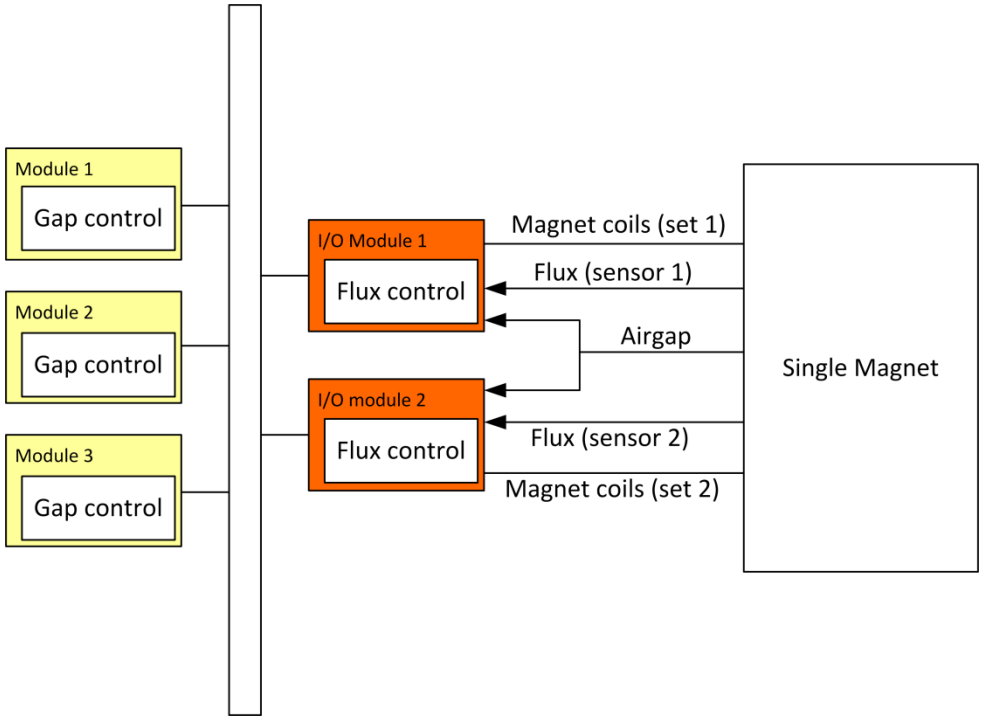


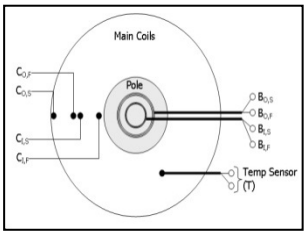
Figure 4-9 Possible interface between IMA and magnet

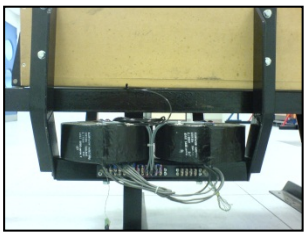
The advantages of this arrangement provides flexibility in arranging parallel processing of the gap control loop for redundancy, akin to fly-by-wire technologies on-board aircraft. Although the flux control loop is executed locally to the sensors and actuator signals, the point failure is removed by duplicating this process for each magnet by utilising a second I/O module, the dual-wound cores and dual search coil installations. Figure 4-9 represents just one possible arrangement for the control implementation of the magnets.

C.7. Experimental Rig Summary

This chapter has outlined the basic control strategy for maintaining the air gap between pole-face and rail and as such infers requirements for the design and distribution of applications across the IMA. Omitted from this chapter are details regarding the designs of power amplification, power supply and sensor configuration – all of which require consideration for the commissioning of the system. This chapter is therefore supported by Appendix A that details designs and practical considerations for these items.

Also highlighted within this chapter is the possibility of operating parallel, redundant processing channels that infer a further set of requirements on the IMA, such as voting mechanisms and appropriate allocation of function. The inclusion of these aspects to the IMA will be highlighted in Chapter 5.





APPENDIX D:

Pseudo-code of Configuration Algorithm

C. Pseudo code of Configuration Algorithm

C.1. Placement Algorithm Process

```
SUBROUTINE: Order functions by levels of criticality

FOR all applications
      SUBROUTINE: Select next function (using defined functional
      relationships and criticality order)

      IF function is not already placed:
            SUBROUTINE: Assign function to resource
                  IF Unable to place function
                        Exit routine
            SUBROUTINE: Update application execution times
      END
END

Record results in systems blueprint
Record results in communication timing
Send system blueprint to all modules
Send communication timing information to all modules
```

C.1.1. Subroutine: Order functions by levels of criticality

```
FOR all applications
      Record all applications listed as highly time critical
END
FOR all applications
      Add to record all applications with actuator outputs
END
FOR all applications
      Add to record all applications with user outputs
END
Add to record all other applications

Search record and delete duplicates where listed with a lower
criticality
```

C.1.2. Subroutine: Select next function

```
IF first call
      Select first function from ordered function record

ELSE
      FOR all applications
            Identify inputs from current selected function from
            Blueprint

            Use blueprint and stored search path to select the next
            input to the selected function

            Make a record of current search path through functional
            topology

            IF no more inputs
                  IF this application has any outputs
                        Retrace the search path to parent function
                        Select this function
                        SUBROUTINE: Update communication and
                        application execution times

                  ELSE
                        Select next function from ordered function
                        record
                        BREAK
                  END
            ELSE
                  Record that the input that has been explored
                  Select the function that provides this input
                  BREAK
            END
      END
END

RETURN selected function
```

C.1.3. Subroutine: Assign function to resource

```
FOR all resources
      Retrieve assignment criteria for selected function from
      blueprint
      Retrieve resource availability information from module
      specification

      FOR all assignment criteria

            For all functions already assigned to selected resource

                  IF selected function incompatible with assigned
                  functions
                        GOTO ReferencePoint
                  END

            END

            IF assignment criteria incompatible with module
            specification
                  GOTO ReferencePoint
            END

      END

      REPEAT
            FOR all functions assigned to selected resource (reverse
            order)

                  For all other functions assigned preceding selected
                  function

                        IF selected function is a pre-requisite for
                        preceding function
                              Swap position of assignment
                        END
                  END
            END
      UNTIL: no change in position

      BREAK

      ReferencePoint
END

Return ability to place function
Return current selected resource
```

C.1.4. Subroutine: Update application execution times

```
For all applications on selected resource
      IF first call
            Create running sum of execution time - initialise to
            zero
      END
      IF function part of network topology of selected function
            Assign function start time as running sum time
      END
      Update running sum of execution time: Previous total + worst
      case execution time of current function
END
```

C.1.5. Subroutine: Update communication and application execution times

```
Obtain all outputs from previous child function from Blueprint

For all outputs

      IF transaction not yet allocated to communication time
            Obtain end time of previous child function
            Search communication record

            IF bus available immediately after expected function end
            time
                  Assign communication transaction to this available
                  slot
                  Update communication record
            ELSE
                  Search communication record for earliest time slot
                  available following execution end time
                  Assign communication transaction to this slot
                  Update communication record
            END

      END
END

Extract assigned start time of selected function

IF assigned start time is earlier than scheduled pre-requisite
transaction
      Assign new start time to function
END
```