Using an FPGA to Enhance Functional Safety in an Automatic Transfer Switch

Andrew Smith 2198699s
MEng Electronic and Software Engineering

ENG5041P: Individual Project 5

Primary Academic Supervisor: Professor Scott Roy
Secondary Academic Supervisor: Professor Edward Wasige
Industrial Supervisor: Julien Guilhemsang PhD

2019-20

Abstract

Safety-critical applications require reliable power supplies. Due to recent trends in the adoption of renewable power sources, being able to switch between power supplies is necessary to keep safety-critical systems, such as hospitals and data-centres, online. The overall aim of this project was to investigate automatic transfer switch (ATS) technology to propose an FPGA-based solution drawing on the relevant safety standards, representing a new approach for a large power supply manufacturer. A prototype was developed in order to evaluate the differences between an existing microcontroller based system and the FPGA-based alternative designed and implemented by this project. The prototype implemented the safety function of the device: controlling the motor which performs the transfer. The V-model for the design and verification of ASICs was applied to the FPGA development process. The FPGA-based system acted as a proof of concept, providing substance for an evaluation of FPGA adoption in ATS motor control. The FPGA-based system successfully replicated the motor control behaviour of the ATS, improving the response time and accuracy of the regulation process as well as increasing the PWM resolution. Through following the published safety standards and by verifying the solution, the foundations for ATS functional safety have been established by this project. The prototype will lead to a new, safer product which will enhance the existing ATS product line.

Acknowledgements

I would like to thank Professor Scott Roy and Professor Edward Wasige at the University of Glasgow for supervising this project.

I would also like to thank Julien Guilhemsang and Jérôme Viriot at Socomec for their continued support throughout the placement.

Additionally I would like to thank the development team at Socomec for being patient with me as I familiarised myself with the French language.

Report Contents

List of Tables

List of Figures

Abbreviations

ATS Automatic Transfer Switch

ATyS Socomec Brand ATS

E/E/PE Electrical/Electronic/Programmable Electronic

FPGA Field Programmable Gate Array

IGBT Insulated-Gate Bipolar Transistor

OSVVM Open Source VHDL Verification Methodology

PCB Printed Circuit Board

PLD Programmable Logic Device

PWM Pulse Width Modulation

SIL Safety Integrity Level

UUT Unit Under Test

UVVM Universal VHDL Verification Methodology

1 | Introduction

In a safety-critical application, the failure or malfunctioning of equipment can result in loss of life, serious injury or damage to the environment[1, 2]. For example, the failure of hospital equipment can be fatal. In these environments, the effects of power cuts can be devastating[3, 4]. The reliability of the supply of power to these systems is essential especially with recent trends towards renewable sources for which a consistent supply to applications cannot be guaranteed[4, 5].

An Automatic Transfer Switch (ATS) is a device that ensures a reliable power source is available to critical loads in a system[3]. This involves monitoring two separate sources and switching between them if necessary. ATS devices have been adopted in a number of environments, including nuclear and medical, to ensure that safety-critical operations can be performed. Consequently, in the development of ATS systems, functional safety and verification are imperative to guarantee that the critical loads are transferred appropriately.

This report describes the design, implementation and verification of a new ATS processor for use in safety-critical applications. All work described in this report was carried out as part of the MEng project for the Socomec Group. Socomec was established in 1922 in Alsace, France, specialising in the control and security of electrical supplies. Currently, there are 28 subsidiaries around the world and have a turnover of 537 million euros. This project was conducted in the global headquarters in Benfeld, France.

1.1 The Socomec ATyS

The Socomec Group product range includes a microcontroller-based ATS device. An example ATyS, the commercial name given to Socomec ATS devices, can be seen in Figure 1.1. The Socomec range can handle a range of power supplies from 125A to 3200A[6]. It is implemented using two control boards: the first is used to monitor the supplies and determine whether it is necessary to perform a transfer, the other board is responsible for performing

the physical switch between the supplies by driving a universal motor. Each of these control boards use a microcontroller as the processor. An example network configuration of the ATyS, where it controls the connection of the circuit between a mains power supply and a back-up generator, is shown in Figure 1.2. Several configurations are possible for the ATyS but the principle remains the same: ensure a reliable supply of power.

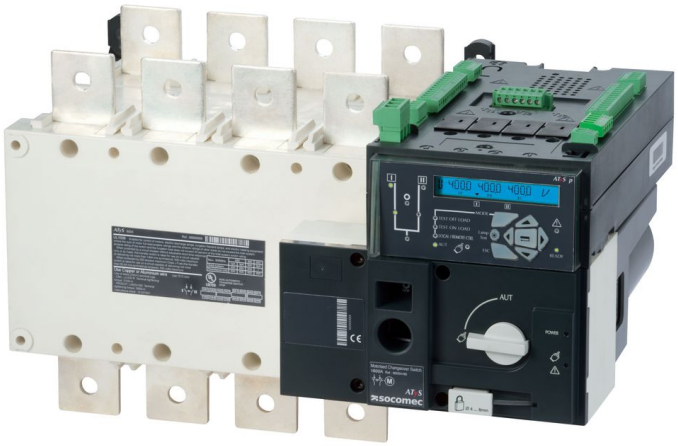


Figure 1.1: The Socomec Automatic Transfer Switch (ATyS)[7]

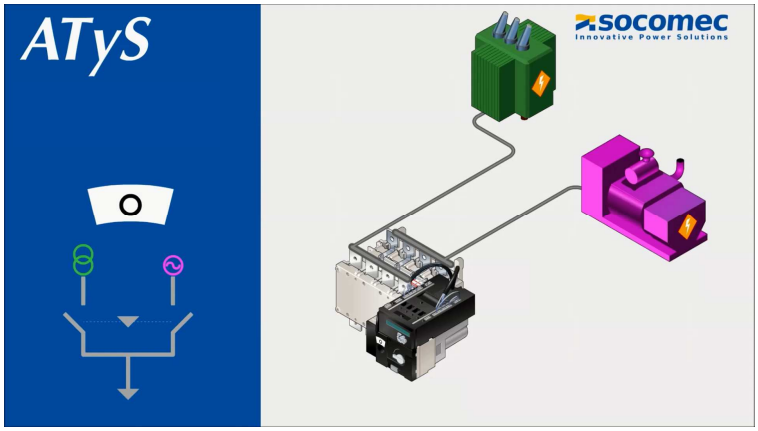


Figure 1.2: Example ATyS configuration controlling a load connection between mains (green) and a back-up generator (pink)[8]

1.2 Project Aims

The project, outlined in this report, involves replacing the micro-controller processor in the ATyS motor control board with a Field Programmable Gate Array (FPGA). The primary motivation for this change is to enhance the functional safety of the ATyS system where functional safety is defined as "ensuring the correct behaviour of a system and detecting or correcting when the behaviour is not as expected"[9, 10]. The overall aim of the project is to investigate the advantages of an FPGA-based solution with a focus on the functional safety of the device. In order to do this, the project had the following objectives:

- Conduct a literature review to discover where FPGAs have been used in similar systems
- Analyse the related safety standards and determine how they could be applied to an FPGA-based system
- Design an FPGA-based alternative solution from the existing micro-controller specification
- Develop VHDL code to implement the design using processes and tools recommended by the safety standards
- Verify the design in accordance with the recommendations of the safety standards
- Physically prototype the FPGA-based solution and compare the response with the original solution

1.3 Report Structure

This report will begin by reviewing literature in areas relevant to this project. As part of this, the theory behind the chosen methodologies for this project is presented and similar projects are discussed. Following this, the design and implementation of the prototype FPGA-based system are presented in Section 3. The verification of this design is then presented in Section 4. The behavioural results of the prototype are discussed and compared against the existing solution in Section 5. The results of the project are evaluated in Section 6 against the initial aims and objectives outlined in this section and suggestions for further work are presented. Finally, the conclusions drawn from the project are summarised.

2 | Literature Review

This section presents the existing literature that underpins the central themes of the project. The relevant safety standards that need to be followed for the project are introduced first. The application of these standards to FPGA-based projects is then reviewed. Examples of the application of FPGAs in safety-critical and industrial switching applications are then explored. To highlight the context of FPGAs as a possible solution, a comparison is made between FPGAs, microcontrollers and ASICs regarding their use in these application areas. Finally, the design methodology which has been adopted for the project, and that is recommended by the standards, is presented and discussed. Verification methodologies are also introduced in this section.

2.1 Safety-Critical Systems

A safety-critical system is defined as "a system whose failure or malfunction may result in death or serious injury to people, damage to equipment or environmental harm"[11]. In the design of these systems, safety standards must be followed[1, 11, 12].

2.1.1 Electronic Safety Standards

The IEC 61508 set of standards covers the use of electrical, electronic and programmable electronic devices in safety-critical systems[13]. Within these standards, Safety Integrity Levels (SIL) are defined. They represent the functional safety level of a device application. The SIL framework has two main aims in providing functional safety: the prevention of systematic errors through thorough design processes and the detection and control of any errors in the device[13]. They also provide a standardised methodology for claiming the safety integrity for an electronic device[2, 14].

2.1.2 FPGAs in Safety Standards

There is an absence of safety-related standards for the application of programmable logic devices (PLDs) such as FPGAs[11, 15, 16]. Without specific reference in safety standards, it can often be unclear which standards should apply to FPGA-based systems[2, 15, 16]. This has resulted in the adoption of standards and techniques for related, general technologies such as for electrical, electronic and programmable electronic devices[11]. In the absence of specific PLD standards, certain aspects specific to PLD/FPGA devices are not fully covered by the standards[15]. These relate to the inconsistency of the classification of FPGAs which contain aspects similar to both hardware and software designs[15, 16]. This can make it difficult to verify the safety of these systems[15, 16]. However, by separating the analysis of the design code and the target device, it is possible to assess the safety of FPGA systems[2, 12]. In this way, FPGAs can be treated as ASICs in regards to functional safety[1, 11, 12].

2.2 Use of FPGAs in Safety-critical Systems

The FPGA market has been growing and expanding into new application areas[17] and safety-critical applications, in particular, have seen increased adoption of FPGAs[1, 11, 12, 15, 16, 17, 18, 19]. Safety-critical application domains which have already adopted FPGA technology include aerospace, industrial control and military[11, 17]. Rodríguez-Andina states that “Digital control of power systems is one of the most interesting current research topics in industrial electronics”[20] and FPGAs are commonly applied to these fields[21, 22, 23, 24].

The time which electronic components remain available on the market has been reducing[25]. The rapid development of new technologies requires components to be regularly updated in order to compete in the market[26]. This poses a major problem for safety-critical systems which, due to their intense verification process, are required to stay operational for extended periods[25, 27]. When system components which implement the safety function become unavailable, the lifetime of the entire system may be cut short[25].

There are a number of classic solutions to this problem including stockpiling components, reverse engineering the unavailable components, requesting manufacturers to fabricate more of the required components, or redesigning affected areas of the system[25]. FPGAs provide an elegant solution to this problem, being able to implement obsolete digital components, for example, microprocessors, to ensure that they are always available for use[25, 28]. Especially with recent advancements in the technology, FPGAs are able to model more complex digital

components and there is an existing market for such Intellectual Property[25].

2.2.1 FPGAs in Industrial Control Projects

In industrial control and safety-critical applications, there is a demand for lower cost and an increased expectation of performance[27, 29]. FPGAs allow for the control functions to be implemented in hardware, giving the possibility of higher performance motor control[30]. The increase in performance means that FPGAs are now fast enough to ensure that processing speed is no longer the bottleneck in IGBT driving which can be applied to motor control[21]. Motor control applications also demand deterministic timings which FPGAs can provide[23, 31]. FPGAs have been specially adapted to these applications with the addition of functional units such as PWM generators and hardware timers[24].

2.3 FPGA Features

There are a number of benefits that FPGAs have over other embedded processors which make them appealing to designers. By performing parallel operations, FPGAs can offer enhanced performance[16, 17, 23, 24, 29, 32, 33] and can reduce calculation time and latency[20, 29, 34]. FPGAs can also monitor multiple inputs and control multiple outputs concurrently and continuously[24, 35, 36]. Since FPGAs operate concurrently, additional diagnostics can be added without impairing the performance of the system[23]. Additionally, diagnostic monitors can run continuously[23].

Modern FPGAs, with dedicated resources, also have enhanced capabilities for interfacing with the analogue world compared to other embedded processors[20]. This means that they can perform like analogue components with minimal delays[37] while providing additional benefits such as reduced noise and ease of control[34]. These benefits are desirable for the development of real-time systems.

FPGAs provide flexibility allowing systems to be redesigned at any stage[32], for example, to respond to changes in the requirements[11]. FPGAs also allows for features to be added to the system later in the design, such as monitors and diagnostics[9, 33]. This flexibility also adds a degree of design freedom, when compared to microcontrollers, allowing a variety of dedicated hardware architectures to be considered for the application[34, 36, 38].

FPGA vs Microcontroller Real-time Safety Comparison		
Property	FPGA	Microcontroller
Timing Behaviour	Deterministic timings: synthesised electronics with predictable timing behaviour that can be accurately simulated[19, 23, 30, 31, 40]	Non-deterministic timings: Adoption of caching mechanisms and interrupts results in unpredictable timings[19, 39]
Input Acquisition Model	Continuous monitoring: inputs are continuously monitored allowing the system to respond to events in realtime[19]	Time-slicing: inputs are monitored on a polling or interrupt-based system which introduces latency[19]
Operating Mode	Real parallelism: multiple calculation streams can be performed concurrently[19, 21, 22, 32, 35, 36, 41]	Simulation of parallel: sequential single-core processors can only handle one task at a time[19, 32]
Processing Speed	Quasi-analogue: adoption of optimised architectures reduces response times to quasi-analogue levels[29, 36, 38]	Fast: micro-controllers can sometimes struggle in high demand environments or with complex calculations[36]
Application of Diagnostics	Parallel diagnostics: can run in parallel with the system operation without affecting performance[9]	Interrupt-based: diagnostics require processing to be taken away from the main system or be performed off-chip[19, 32]
Design Language	Technology independent: HDL designs are independent of the hardware and so a different FPGA device can be used with minimal effort[19, 23, 24, 35, 36, 42]	Technology dependant: microcontroller firmware is often hardware-dependent making it more prone to obsolescence[19, 35]

Table 2.1: Comparison between FPGAs and micro-controllers for use in real-time safety-critical systems

2.3.1 Microcontroller Comparison

FPGAs, implemented in hardware description languages (HDLs), are synthesised as electronic components[35]. This means that functional and timing behaviour is deterministic and predictable[30, 39]. As a result, the verification possibilities of these devices are enhanced[1, 19, 27] with respect to microcontroller solutions developed in sequential programming languages such as C. The interrupt-driven behaviour and the use of caching mechanisms, adopted by microcontroller designs, means that it is difficult to accurately model and verify these systems[19, 27, 39]. The differences between FPGAs and microcontrollers for use in safety-critical and real-time systems, due to this difference in behaviour, are summarised in Table 2.1.

2.3.2 ASIC Comparison

FPGA-based projects have been found to have a shorter time-to-market and lower non-recurring engineering costs than in ASIC development[2, 11, 15, 17, 43]. This has been accredited to FPGA re-programmability meaning that expensive silicon respins, which occur during ASIC development, are not necessary[2, 11]. The balance of costs can depend on a number of factors including package costs, area costs and development costs which all vary depending on the application[43, 44]. FPGAs can be up to 100 times more expensive per-unit[43]. On the other hand, ASIC designs, depending on the fabrication and technology, can cost over $1 million per spin[45]. The reduced time-to-market means that FPGAs are often more cost-effective for low volume production[11, 17]. This has made FPGAs more popular for a variety of applications[15, 23]. However, this re-programmability can encourage a reduced verification cycle and Foster finds that the reduced design life-cycle of FPGAs means that there is usually a higher number of bugs found in production compared to ASIC systems[17]. The use of ASICs was not considered for this project due to the high up-front costs of ASIC prototyping.

2.3.3 FPGA Verification

As FPGA technology advances, the verification overhead increases[15, 46]. There is an increasing need for efficient verification of the functionality of these systems[17, 20]. Modern FPGAs often resemble System-on-Chip (SoC) devices which are commonly referred to as FPSoCs[11, 20, 34, 47]. With multiple functionalities and interfaces, verification becomes more of a challenge[17].

In terms of design safety and verification, FPGAs offer enhanced possibilities[27]. Only the synthesised electronics must be verified whereas in microcontrollers it can be necessary to verify the software, hardware and the operating system[27]. In order to achieve verification in FPGA designs, simulation of individual modules is required alongside the verification of the interactions between modules[35]. System components which control the safety function should have 100% coverage[48]. Simulation can be performed at each level of the design[35] and it remains the dominant form of verification for HDL designs[35, 49]. Accurate timings can be modelled once the place and route are known[35].

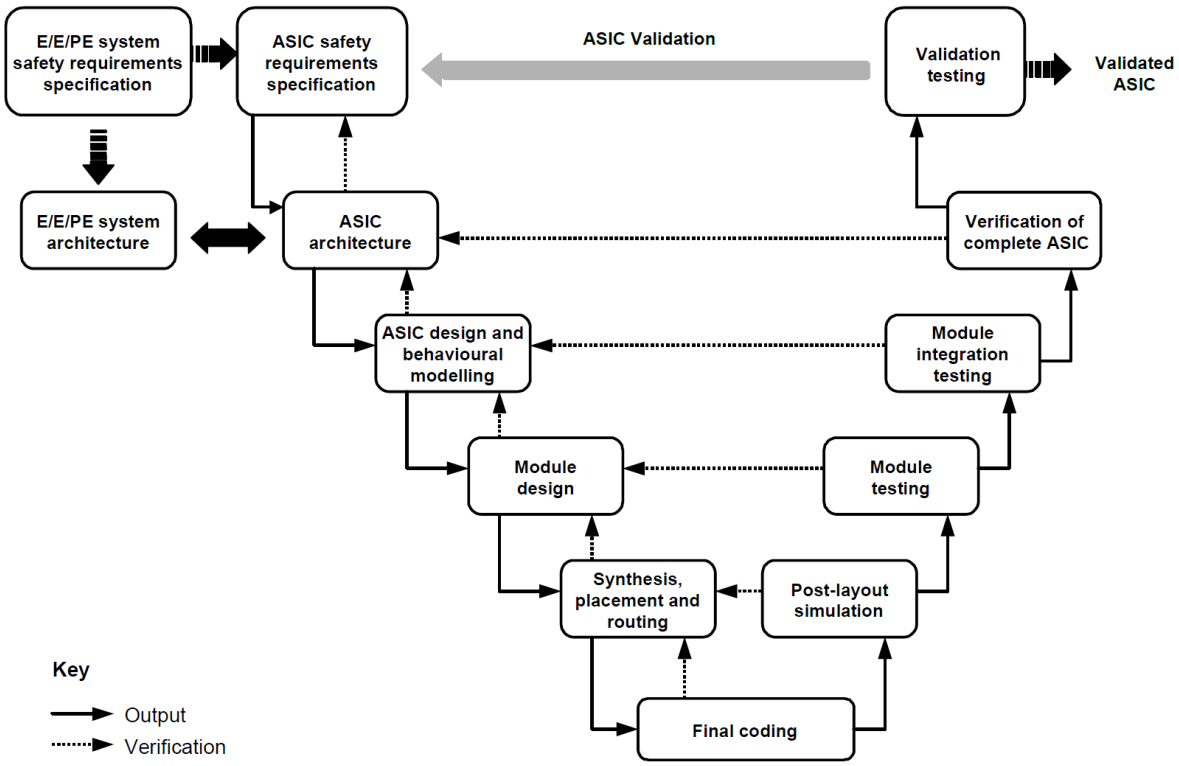


Figure 2.1: The V-model for ASIC development adopted for design and verification in this project[13]

2.4 V-Model Development

Adoption of the V-model, see Figure 2.1, for ASIC and FPGA designs is recommended by Lattice for safety-related development[50] as well as in the safety standards [1, 2, 11, 12, 13]. The model focuses on the verification and validation of the design at each design phase[2]. For each design phase, there is a corresponding verification activity[2, 48]. It incorporates methodologies from software development into the ASIC design process[12]. As well as acting as an overall model for project development, the V-model can be traversed for subsystems and modules within the design[48]. HDLs support the adoption of the V-model as they allow for the behaviour and performance to be verified at each development phase of the project[23].

2.4.1 VHDL Verification Methodologies

In order to tackle the increasing verification effort of FPGA designs, verification methodologies have emerged[17]. There are a number of verification methodologies applicable to FPGAs and VHDL development. Verification is a large component of the design life-cycle.

Tallaksen finds that verification accounts for 50% of the development effort for FPGA-based system[51]. It is a crucial activity which produces a valuable resource: automated and comprehensive test-suites provide a safety-blanket to ensure that any errors in the design are flagged[52]. One methodology, which is rapidly growing in popularity, is the Universal VHDL Verification Methodology (UVVM)[17, 51]. The UVVM framework claims to provide overview, modifiability, debuggability and reusability to unstructured test-benches[51]. OSVVM (Open Source VHDL Verification Methodology) and UVVM have seen a considerable adoption for verification in FPGA-based design projects; in 2018 this was found to be 15% and 10% respectively[17, 51].

2.5 Literature Review Conclusion

FPGAs have seen considerable adoption in safety-critical and motor control. The relevant safety standards for the project are the IEC 61508 standards related to electronics development in safety-critical applications. Although there is no direct mention of FPGAs in this particular standard, FPGAs can be considered as ASICs throughout the development phases. The primary advantages of using FPGAs for safety-critical systems is having the ability to accurately model and verify timings and behaviour. Advancements in FPGA technology have resulted in more complicated designs with multiple interfaces. This has introduced a significant verification requirement meaning that classic test-benches are no longer viable for complete verification. Verification methodologies, which have recently emerged, allow for the functional safety of FPGA systems to be achieved and proven. FPGAs also provide an elegant solution to component obsolescence, for which safety-critical systems are particularly vulnerable.

The V-model, which is recommended for the standards, will be followed throughout the development of this project. It also forms the structure of this report. The following two sections traverse the V-model, discussing the high-level processes within the project. The design section, Section 3, discusses each of the activities on the left side of the V-model. In the verification section, Section 4, the corresponding verification activities for each of the design phases are described. These are the activities on the right side of the V-model.

3 | Design and Implementation

This section presents the design of the FPGA-based control board prototype. The design followed the top-down design stages of the V-model shown in Figure 2.1. This section follows the design flow by first discussing the requirements for the system and how they were identified. This includes the identification of the subset of requirements necessary to implement the safety function of the device. This is followed by a presentation of the architecture that was derived from these requirements. The design and implementation of the modules which form the architecture is then presented. The physical prototype, created as part of this project, is then discussed.

3.1 V-Model Design Process

The design stages outlined in the V-model follow a top-down methodology. The methodology starts with the definition of the system and safety requirements. From these requirements, an architecture is derived. The behaviour of this architecture must then be modelled. Modules which form the architecture can then be designed. The final stage of design, following the V-model, is the synthesis, placement and routing of the design. This stage involves the application of constraints to the synthesis environment as well as the mapping of the design. Each of these activities are described in the following sections.

3.2 Prototype System Requirements

An ATyS specification document was created by Socomec as part of a previous iteration of the device. This existing specification of the microcontroller-based system represents a full set of the requirements for the ATyS motor control. The motor control process has a number of operating modes and possible conditions. This includes the option to change the load motor as well as the command interface. For example, the commands can come from a customer

input or from the source detection board of the ATyS.

For this prototype, the requirements which are necessary for performing the motor control were identified and this became the specification for the prototype. Only safety features should be considered with respect to the functional safety of the device[12]. The comprehensive microcontroller specification was used to derive the subset of requirements to be implemented as a prototype. The motor control process was identified as the safety function and so this was the central focus for the prototype design. It is the core behaviour of the device and performs the essential behaviour of changing the connection from one source to another. External PCB requirements are outside of the scope of this project. Only the motor control requirements that relate to the processor and its interfaces were considered for this design. The requirements for the prototype are outlined in Table 3.1.

The overall process of motor control involves moving from the current position to a desired position and stopping when the position is reached. Several stages, which need to be performed in order to complete this process, are outlined in the existing specification. The external interfacing requirements of the motor control process can be broken down into three sections. The first of which is the command input. This uses a simple command which is input to the device by activating a number of pins. The second requirement is the constant detection of the position. This is done using a series of pushbuttons, the state of which determine the physical position of the motor. The final required interface of the motor control safety function is the control of the motor. There are also internal requirements for the processing of these inputs and outputs. PWM regulation allows for accurate motor control. The regulation scheme is to use the motor current value to adjust the PWM supply to the motor. PWM regulation also dependant on the supply voltage.

3.3 System Architecture

As dictated by the V-model process, the system architecture was derived from the system requirements. Similar FPGA implementations were analysed in order to identify common functional units used in FPGA motor control systems. Several architectures which have been designed in order to perform motor driving were analysed and common functional units include ADCs, position measurement, current regulation and PWM modules[9, 23, 33, 37, 38]. Each of these units translates to the interfacing and processing requirements of the ATyS. The architecture which was derived from the requirements for this system is shown in Figure 3.1. The inputs and outputs were defined by the requirements for accepting commands, reading position, analogue value monitoring and providing motor control signals. With a defined set of interfaces, the processes required in order to fulfil the requirements had to be decided. The

ATyS Motor Control Requirements	
Requirement	Description
Input R1	Translate the three valid command inputs into an encoded format
Input R2	Translate the seven valid position inputs into an encoded format
Input R3	Calculate the direction of movement required based on the position and command inputs
Input R4	Control the motor movement stages: startup, regulation and braking
PWM R1	Control the duty-cycle through each of the preset motor control stages
PWM R2	Regulate the PWM based on the measured current value - limit to 1.5A
PWM R3	Generate a 1kHz PWM signal of a duty-cycle which is calculated by the regulation process
PWM R4	Update the current value in the middle of the PWM on-time
PWM R5	Limit the maximum PWM based on the supply voltage (100% below 170V and reducing linearly to 50% at 333V)
ADC R1	Measure the supply voltage to 8-bit resolution when the motor is not being operated
ADC R2	Measure the motor current to 8-bit resolution when the motor is being operated
Drive R1	Drive the IGBTs based on the control signals generated by the Input Processing module
Drive R2	Drive the Thyristors based on the control signals generated by the Input Processing module

Table 3.1: List of requirements for performing the motor control safety-function of the ATyS

system must make a decision based on the command and position inputs as to what motor control is required. The Input Processing module was added to the architecture in order to achieve this. The system must be able to measure the supply voltage and motor current in order to adjust the control scheme based on the environment. An ADC module was incorporated to provide these values to the regulation scheme. In cases of a stalled or blocked motor, the system must control regulation on the current to prevent high currents. The PWM Regulation module takes the output from the Input Processing and the reading from the ADC in order to control the current. The PWM signal which is generated must be correctly routed to the motor control circuitry. The Motor Driving module was added to the architecture in order to achieve this.

The Input Processing module handles the command and position input. The module needs

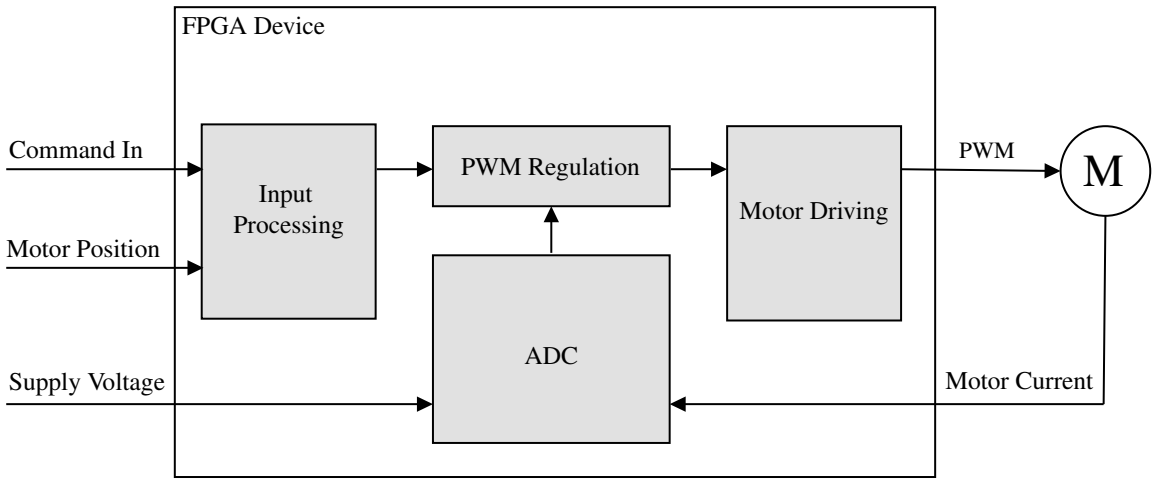


Figure 3.1: High-level system architecture diagram showing system modules

to analyse the values of each of the inputs and determine whether a movement is necessary. It should interpret the received commands and position to determine the direction of the movement if a movement is necessary. This module also controls the motor driving process which handles timings and stopping conditions.

The ADC module performs need to perform two measurements in order to provide voltage and current information to the PWM Regulation module. The supply voltage must be measured when the motor is not being driven. This allows the regulation process to limit the maximum PWM based on the supply voltage to prevent high currents. During motor control, the ADC must measure the motor current so that the PWM Regulation process can update the duty-cycle based on the response of the motor.

The PWM Regulation module calculates the required duty-cycle for motor driving. It does this by monitoring the motor current, calculated by the ADC, and adjusting the duty-cycle accordingly. The upper limit of the duty-cycles is determined by the supply voltage. This is to prevent higher voltages from over-accelerating the motor. The PWM module is also required for generating a PWM of the calculated duty-cycle value.

The motor driving module takes the PWM output by the regulation unit and supplies it to the correct motor driving transistors. The driving is performed by two IGBT / Thyristor pairs. This module needs to ensure the correct pair is driven based on the required direction of movement. It also performs the Thyristor and IGBT startup by driving them full for a given time specified in the requirements. The braking phase, also performed by the motor driving module, involves fully driving both IGBTs to stop the rotation of the motor.

3.4 Module Design and Implementation

After the modules and the architecture have been defined, the V-model recommends performing module design. This process involves breaking down the module requirements into sub-modules which can then be designed in accordance to the V-model. For each of the four modules defined in the system architecture, the inputs, outputs and processing requirements are discussed. The implementation of the design is also presented for each module.

3.4.1 Input Processing

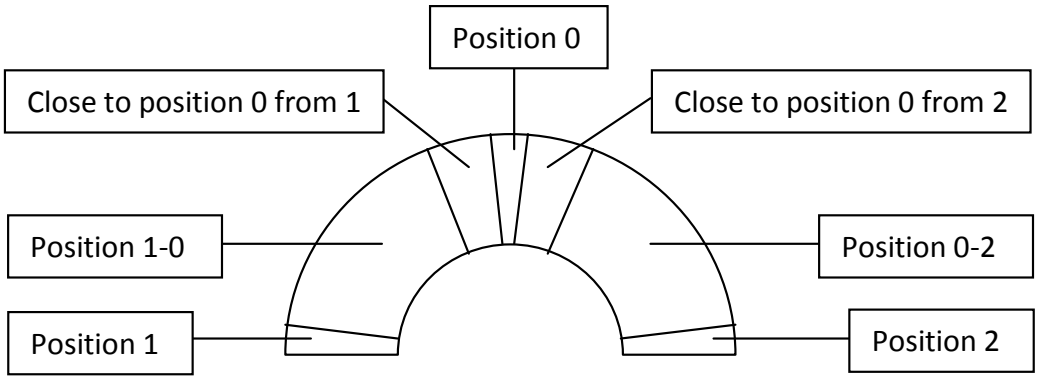


Figure 3.2: Valid motor positions from original ATyS specification

The position of the motor has to be monitored continuously throughout the control process in order to determine whether the desired position has been reached. The position is read from a series of push-button switches. The valid motor positions can be seen in Figure 3.2. This diagram shows the three motor positions zero, one and two representing a connection between no sources, source one and source two respectively. The diagram also shows the readable intermediate positions. In total there are seven valid positions for the motor to be in. The position measurement needs to be filtered in order to debounce the signal, ensuring that only valid position changes are propagated through the system.

The existing acquisition method was used in this design to determine the position of the motor. There are four buttons read into the system, along with their inverse value to confirm the state of the button, in order to determine the position of the motor. The position acquisition process translates these into a commonly understood positional encoding. The code which performs this translation is shown in Appendix Item A.1. This encoding scheme indicated the absolute and relative position (for example to the left of position zero). This allowed the required movement to be calculated more easily and requiring less logic. It also handles movement from intermediate positions.

For this prototype, only the customer command interface is being considered. This means that commands are supplied by stimulating a voltage input. There are three possible commands: go to position zero, go to position one and go to position two. Each of these should be processed and the motor should move to the position indicated by the command input. The command is encoded in a similar way to the position. The required movement is encoded as a position in order to perform a comparison and calculate the required movement. This also allows the desired position to be determined so that the system can stop the motor when the position is reached.

A shared debouncing process was used for both the position acquisition and the command interpreting in order to prevent glitches in the reading of inputs. It uses a shift register and comparison in order to ensure the input value is consistent. This shift register is currently only three bits; however, this value can be expanded to allow for more sophisticated debouncing. In this application, it was found that three bits was sufficient to avoid misreadings.

The device must interpret from the current position and the command which has been supplied whether a movement is necessary. It uses the shared position encoding scheme of the command and the position reading to compare the value and determine a direction of movement unless they are the same then no movement is required. After determining the required position, it provides directional information to the modules which perform the movement. The code which performs this decision-making logic is shown in Appendix Item A.2. The position must then be continually monitored to detect when the motor movement has been reached. Once this module detects the position has been reached, it must signal this to the movement modules to halt the motion. Taken from the specification, this module will signal the motor to stop if the position has not been reached after five seconds.

3.4.2 ADC

There are two values which have to be measured by the processor: the supply voltage and the motor current. These allow for the accurate control of the motor. The value of the supply voltage determines the maximum duty-cycle of the PWM which drives the motor. This is to prevent over-driving which could cause current spikes. This value does not need to be updated during the motor driving phase. The motor current is used to regulate the PWM value to allow for optimal control of the motor during the motor driving phase.

The ADC which has been adopted in this FPGA is a sigma-delta ADC implemented in VHDL. This ADC has been implemented by Lattice Semiconductor and the structure of it is shown in Figure 3.3. Each of the modules is implemented internally to the FPGA with the exception of the RC Network and the Comparator which are implemented on the peripheral

of the FPGA. Since these measurements are never taken at the same time and in the interest of saving space on the FPGA, these two values are measured using the same ADC. The value measurement of the Lattice ADC is multiplexed to allow each of the values to be measured. The implementation code which performs this input multiplexing is shown in Appendix Item A.3.

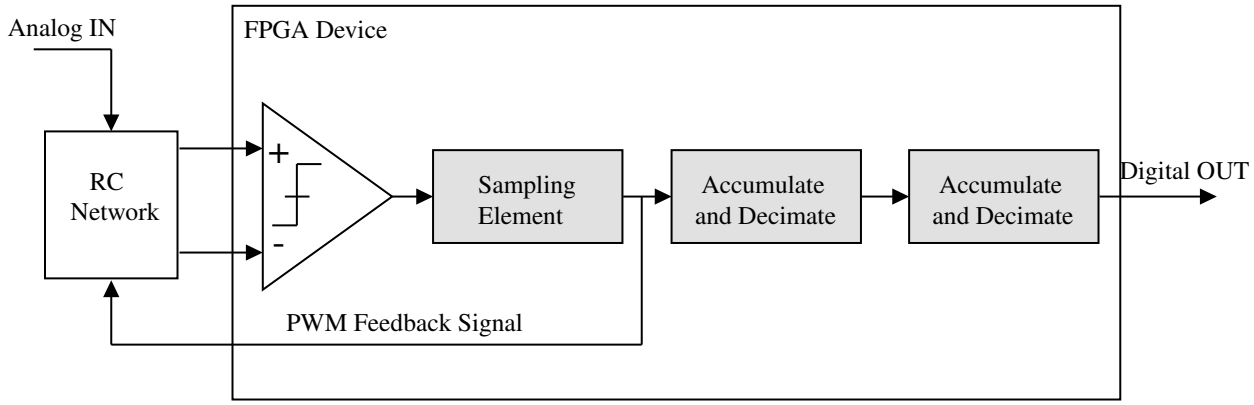


Figure 3.3: Schematic of the ADC obtained from Lattice[53]

3.4.3 PWM Regulation

The regulation of PWM limits the motor current which prevents dangerous current spikes. The measured motor current should be controlled against a predefined reference[37]. The duty-cycle of the applied PWM can be regulated in order to control the speed of the motor[54]. Duty-cycle controlled motor drive simplifies the regulation of motor speed[54]. From the specification, the PWM needs to be regulated using a simple proportional control based on the motor current. Linear regulation of PWM is a simple and efficient control system[54].

Since the motor is being driven by a PWM signal, the current spikes up when the PWM is high and it drops when the PWM is low. In order to filter out the effect of this and get an average reading, the current should be measured in the middle of the PWM on-time. This allows the value to be averaged and provides a consistent reading. FPGAs are able to synchronise the measurement of the current and the PWM duty-cycle in order to obtain a more accurate measurement[21, 22]. This current value is used to update the PWM value to allow for optimal control.

The PWM regulation scheme was implemented using a first-order proportional control scheme shown in Figure 3.4. This control scheme was chosen because it can easily be implemented in a VHDL design using a series of basic calculations and shifts. Techniques such as fixed point

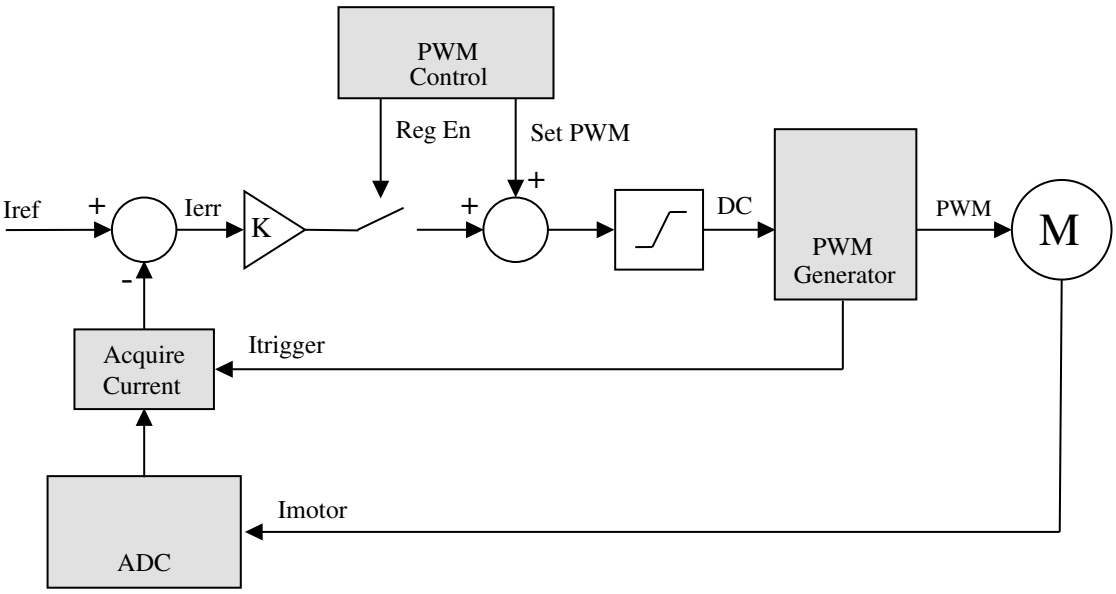


Figure 3.4: PWM regulation process control loop

arithmetic can be used to implement more complex control systems. However, a first-order scheme was found to be efficient and sufficient for this application. In this control scheme, the current value measured by the ADC is compared against the nominal value of 1A. This error is then translated into an error in PWM duty-cycle by applying a gain. Then, if the regulation is enabled in the control phase, the PWM duty-cycle value is updated. The value is truncated to ensure that it stays between the valid boundary set by the minimum and maximum values. The code which implements this PWM regulation process is shown in Appendix Item A.4. The calculated value is provided to the PWM generator which translates this value into a PWM signal with a duty-cycle specified by the regulation scheme. The PWM Generator module triggers the current measurement at the midpoint of the duty-cycle in order to smooth the current measurement, as mentioned above. When triggered, the ADC value is taken in and used in the comparison. The value updates at 1kHz, once for every PWM cycle.

This control scheme was used to map to the response from the microcontroller-based regulation scheme. The output of the regulation scheme was analysed and modelled. The microcontroller output that was analysed and modelled is shown in Section 4.3.3. It was found to be a simple proportional controller. The simple control response was modified to include processes outlined in the requirements. For example, the current is acquired half-way through the on-time using a trigger from the PWM duty-cycle. Truncation was performed on the calculated value to ensure that it stayed between the defined minimum and the calculated maximum.

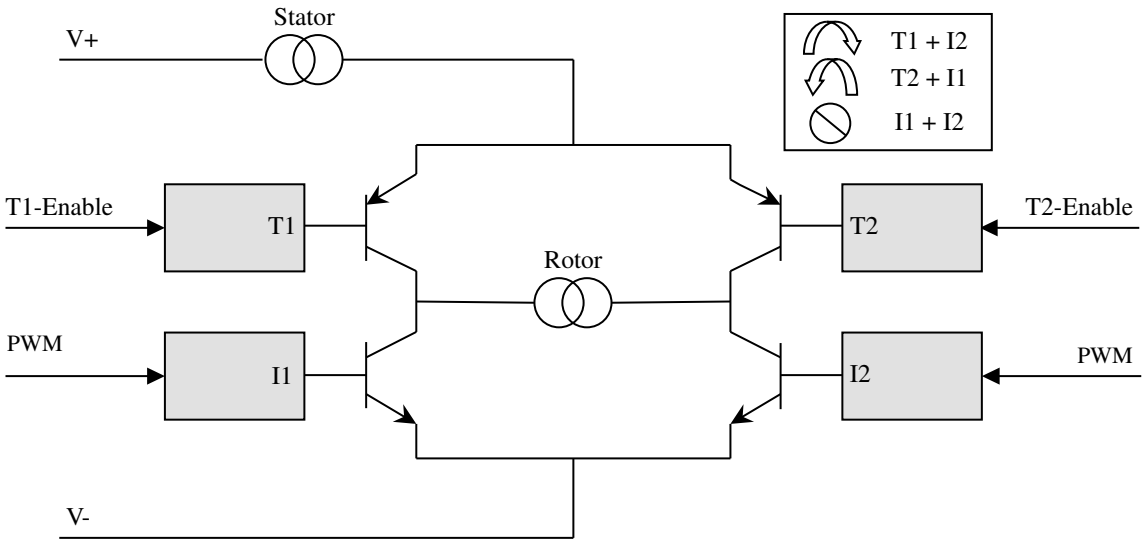


Figure 3.5: Motor driving circuit diagram showing a pair of Thyristors (T1 and T2) and a pair of IGBTs (I1 and I2) redrawn from the ATyS specification (requirements: Drive R1 and Drive R2)

3.4.4 Motor Driving

The motor is driven using an H-bridge scheme consisting of two IGBT and Thyristor pairs. This circuit can be seen in Figure 3.5. The motor drive module has to multiplex the PWM and control signals generated by the regulation phase to drive the correct IGBTs and Thyristors. The driving sequence is being modelled on the current set of requirements. It contains several discrete stages in order to drive the Thyristors and IGBTs to move the motor to the required position.

Indicated in the diagram (Figure 3.5) are the control signals required to perform certain actions on the motor. This includes the direction of rotation which is achieved by stimulating opposite Thyristor and IGBT combinations. Additionally, the brake functionality is shown which involves driving both of the IGBTs. The PWM from the regulation scheme is provided to the correct IGBT by the Motor Driving module. The code which provides the necessary control to the IGBTs is shown in Appendix Item A.5.

3.4.5 Clock Design

As defined in the micro-processor specification, the requirement for the PWM produced is a 1kHz PWM signal with a resolution of 100. This means that the PWM can be controlled to an accuracy of 1%. From this, it can be determined that the PWM must be driven with a clock of at least 100kHz. The ADC requires a higher clock frequency in order to perform

oversampling. The speed of the ADC determines the accuracy of the measurement used in current regulation. Therefore, the faster the result of the ADC is produced, the better. Due to the ADC implementation, this metric translates directly to the clock frequency.

The internal oscillator on the MachXO2 was used to generate the higher clock for the ADC. The maximum value of the clock, 133 MHz, was used in order to maximise the data-rate of the current measurements. This clock signal was divided down to 200kHz using the phase-locked loop in order to drive the rest of the system. This is sufficient to allow for the regulation process to update the PWM before the next cycle. It also allows for the PWM signal to have double the resolution than is specified.

3.4.6 Diagnostics

System diagnostics are highly recommended by the IEC 61508 SIL standards as a measure to reduce the effect of errors[13]. They help to improve the functional safety of a device by detecting, and possibly correcting, dangerous conditions[9]. Where good design practices can prevent systematic errors in the system, diagnostics help to prevent and reduce the effect of random hardware faults. The implementation of diagnostics for this system is beyond the scope of this project. A number of applicable diagnostics are discussed in the remainder of this section.

A number of diagnostics are discussed by Jeppesen, Rajamani and Smith[9]. Diagnostics which have been applied to the FPGA solutions include a PWM checker, watchdog timer, ADC out-of-range detection and module self-tests by error injection[9]. Each of these diagnostic measures could be applied to the ATyS processor developed in this project to enhance the functional safety of the solution.

Additionally, full redundancy architectures can be utilised in order to verify system behaviour[9, 55]. The use of a redundancy architecture, which is possible in FPGAs[12], is recommended by the standards[2, 13] as it increases the reliability of the system[1]. Duplex redundancy is a common methodology which requires the system to be synthesised twice and the outputs are compared[12]. An example duplex redundancy architecture, 1oo2 (one out of two), is shown in Figure 3.6. The system must be synthesised twice in order to compare the results and ensure that the results are consistent as shown in the diagram.

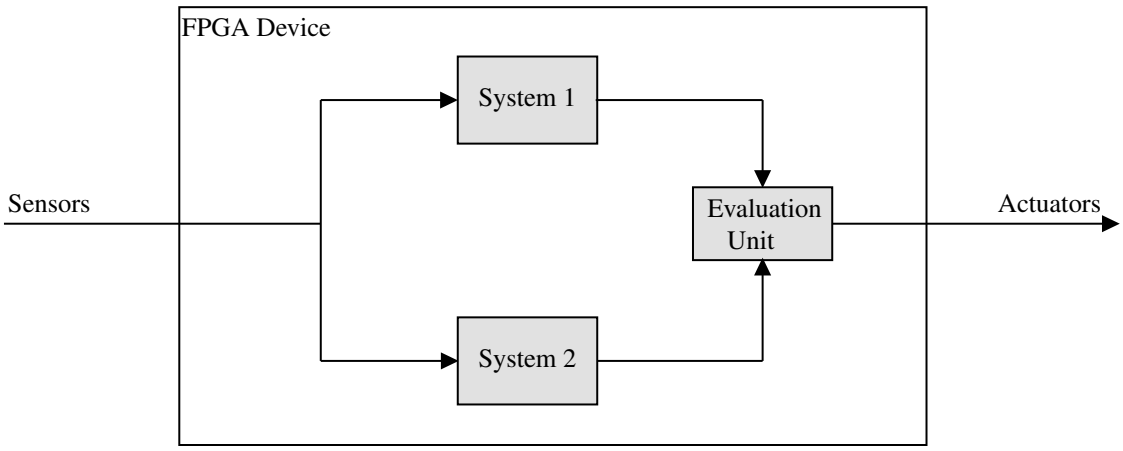


Figure 3.6: FPGA (1oo2) on-chip redundancy architecture (based on diagram in [2])

3.5 Physical Prototype

As part of the development of this project, a physical prototype was created. The prototype consists of a modified version of the PCB used in the microcontroller solution. In this section, the adoption of tools and components used in the implementation of the prototype are discussed. The PCB changes, made by the Socomec design team, in order to incorporate an FPGA into the existing commercial ATyS PCB are also discussed in this section. The results of the synthesis process for the design are presented. Additionally, the implemented behaviour of the prototype is presented.

3.5.1 FPGA and Tool Adoption

A number of factors need to be considered in the selection of an embedded controller. These include cost, size, performance, availability and verification capability[22]. The FPGA which has been adopted for the project is the Lattice Mach XO2-2000. This FPGA has 2112 LUT4 logic units. The larger package was chosen to provide as much flexibility as possible for the prototype. The MachXO2 FPGA has in-built diagnostics including configuration checks. The safety compatible version of Lattice Diamond (Lattice Diamond 2.0) was used for the development in this project. The safety standards require proven-in-use tools to be used in the development of safety-critical systems. This version of Diamond has been marked as proven-in-use by Lattice.

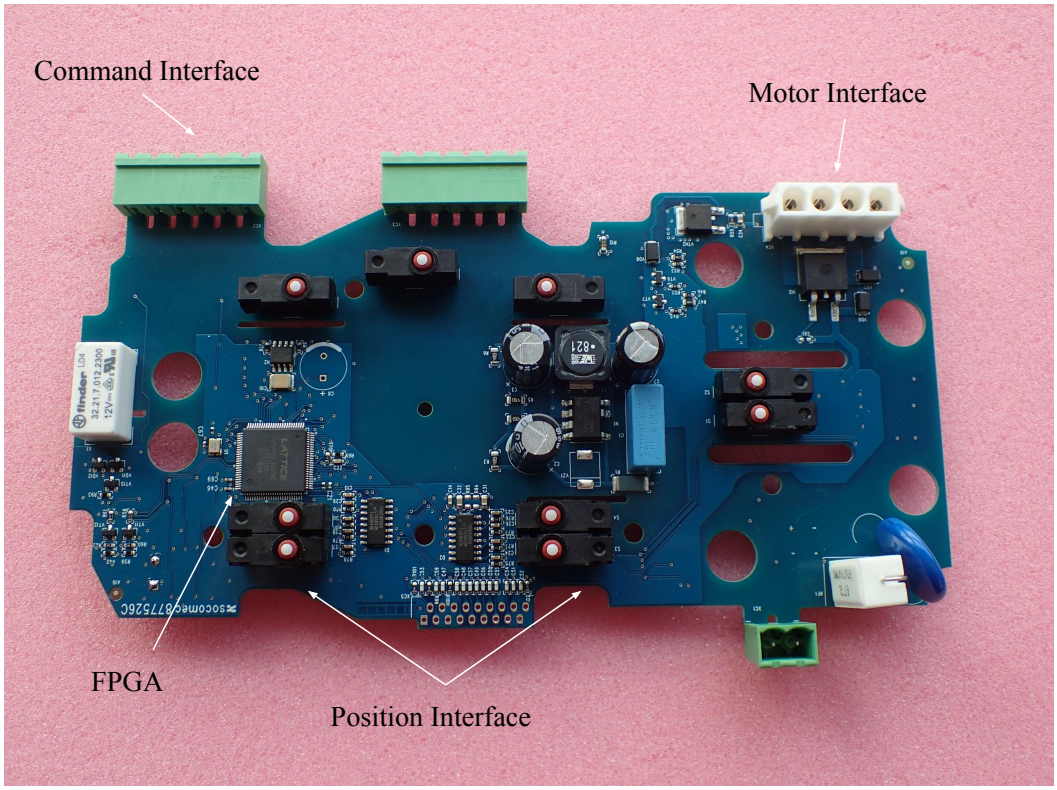


Figure 3.7: The prototype motor control PCB with an FPGA processor

3.5.2 PCB Re-design

The PCB was re-designed to allow for the prototyping of an FPGA solution. This PCB can be seen in Figure 3.7. This task was performed by design engineers at Socomec. The movement to an FPGA solution required minimal changes to the existing design. The most considerable change was the switch of the FPGA in the place of the microcontroller. Since most of the FPGA pins are freely configurable, the change from a microcontroller to an FPGA is flexible in terms of routing[35]. The nature of the design implementation allowed the ADC to be implemented using a simple RC network. External memory units could be removed as the FPGA has internal memory units. A JTAG programming interface was added to the design to allow the FPGA to be reprogrammed.

3.5.3 Prototyping Strategy

An incremental prototyping strategy was used in order to test each of the design modules independently. This involved mapping inputs to the series of push-buttons used for position acquisition and mapping the outputs to general I/O ports. This allowed the inputs of modules to be simulated and the outputs to be analysed on an oscilloscope. This provided insight into

the physical behaviour of each module and allowed the design to be implemented incrementally. This process was performed after the module verification, discussed in Section 4. This meant that each of the individual modules had already been verified which resulted in a rapid implementation process with each of the modules working as expected.

Despite the fact that each of the modules had been functionally verified, there were a number of problems relating to the specification which were identified while prototyping the design. For example, the naming convention was inconsistent between the schematic for the microcontroller solution and the FPGA board. This resulted in the positions being read in reverse. For example, position 1 was read as position 2. However, direction calculation and the position translation was updated and the expected behaviour was realised.

Components which were not utilised by the prototype were used as debugging interfaces. The interface used to communicate with the second control board was used to view the values produced by the ADC. This allowed the RC components to be tuned to the optimum range for the measurement voltage. It also allowed for debugging when testing the regulation control process as the values could be stepped through on the oscilloscope and the response could be analysed.

3.5.4 Synthesis Results

Mach XO2 FPGAs come in different package sizes for which the number of logic units and the number of pins varies as well as the cost. The synthesised prototype design used 36% of the capacity of the Mach XO2-2000 FPGA used in the project. This leaves 1352 of the 2112 logic units in the FPGA available for use. This additional space could be used for the diagnostic components and redundancy architectures discussed in Section 3.4.6. Alternatively, the package size could be reduced. The design used 20 of the 100 FPGA I/O ports. This was composed of eight for the position measurement, three for the command input, three for each of the ADC measurements and six for the motor control signals.

3.5.5 Prototype Behaviour

The prototype successfully implemented the safety function of the device. Physical prototyping allowed for the behaviour of the designed system to be verified and compared against the existing solution. Functionality of the physical prototype includes performing the core safety function of the device, performing a transfer from one power source to another. This behaviour requires the correct acquisition of commands and position and calculation of the

required movement as well as the translation of this into a control sequence to drive the motor. Additional functionality includes the regulation of the current in order to control the response in cases of stalled or blocked motors. The timeout and movement retry behaviour in the microcontroller solution was also replicated in the design and prototype.

3.6 Design Conclusion

The FPGA-based system was designed based on the original microcontroller-based system specification. The motor control behaviour was identified as the safety function of the device. The requirements that allow this to be performed were derived from this specification and became the requirements for the design of the FPGA system. Following the V-model process, a system architecture was derived from these requirements. Four main system processes were identified in order to meet the requirements and were integrated as modules to form the system architecture. These modules are Input Processing, ADC, PWM Regulation and Motor Driving. The design and implementation of these modules has been described in this section. Additionally, the clock design was discussed. The planning of diagnostics was also presented; however, the implementation of these diagnostics was outside the scope of the project. Finally, the physical prototype which was created as part of the development in this project was presented in this section. The prototype successfully replicated the safety function behaviour of the microcontroller board in the ATyS.

4 | Verification

This section describes the verification activities undertaken as part of this project, giving examples of how they were applied to each of the modules in the design. Firstly, the use of the V-model process for verification in this project is discussed. This is followed by a discussion of how verification methodologies were utilised throughout verification, simulation and testing in this project. The verification plan, which maps directly to the original project requirements, is presented. The verification results are then presented. The verification of each module in the system architecture is discussed with example simulation and verification results.

In the interest of maintaining clarity of terminology used in this context, terms will be explained. In this section, verification will refer to the direct mapping and confirmation of functionality to the requirements of the system. Simulation will comprise of the analysis of produced waveforms. Testing will cover activities that incorporating timings to ensure the correct physical behaviour of the design. The term Modules will be used to refer to the high level system modules in the system architecture. There are four modules in the design: Input Processing, PWM Regulation, ADC and Motor Driving. Sub-modules will be used to refer to any design unit used within the high-level module. These sub-modules directly map to an individual requirement whereas the modules refer to a group of requirements.

4.1 Verification Stages

The V-model, shown in Figure 2.1, encourages verification of every level of the design. The verification stages follow a bottom-up methodology with a complimentary verification stage for each design activity. The first stage involves post-layout simulation to check the conformity of the design to the design rules set by the FPGA. This activity is performed automatically by the Lattice Diamond tool-suite. Following this, individual modules are verified. This stage benefits from small contained modules with a single testable functionality. After the modules have been verified, the integration of and interactions between the modules can

be verified. This process can be repeated until all modules are integrated, at which point the complete design is verified. Direct validation of requirements is the final verification stage in the V-model. Each of the requirements of the design must be validated in order to produce a validated VHDL design.

Requirements at each phase must be directly testable in order to produce a validated design. During the design phases, a test plan was produced in order to verify the requirements of that phase. This includes behavioural verification which involves ensuring the outputs produced by the unit under test correspond correctly to the input sequence. Using VHDL, it is also possible to accurately model the timing behaviour. This makes it easier to verify the real-time behaviour required by motor control applications.

4.2 Verification Methodologies

Verification methodologies were adopted to aid the automation of verification, simulation and testing in this project. Two methodologies, introduced in Section 2, were adopted in order to perform the required verification for this project: UVVM (Universal VHDL Verification Methodology)[56] and OSVVM (Open Source VHDL Verification Methodology)[57]. UVVM has been adopted in order to provide stimulus and value checking whereas OSVVM has been adopted to provide automatic coverage collection. The application of these methodologies is described below.

4.2.1 Universal VHDL Verification Methodology

UVVM was applied to the design for simulation stimulus, value checking and timing verification. A number of functions are provided by the methodology for analysing the values of signals at different points in a simulation which can be triggered by events or time-windows. This allows the behavioural response of the design to be verified alongside the timing of the response. As well as producing graphical simulation results which display the states of each signal in the unit under test (UUT), textual results can be produced to automate the checking of required values at different points in the simulation. The use of these functions allowed the functional and timing behaviour of the system modules to be verified.

Figure 4.1 shows the application of UVVM to an example design module, the Position Acquisition module. The diagram, generated by the VHDL editor Sigasi, shows how the inputs of a UUT can be stimulated and how its outputs are monitored by the UVVM process (labelled uvvm_driver in the diagram). This is shown in Figure 4.1. The diagram shows the

input clock, reset and position buttons being provided to the left of the UUT by the UVVM driver. This process allowed each of the test cases to be applied to the UUT. The response of the calculated position, shown on the right of the UUT, was then compared to the expected position for each case. This methodology was used for each module in the design. Example UVVM code used to verify the Position module is shown in Appendix Item A.7.

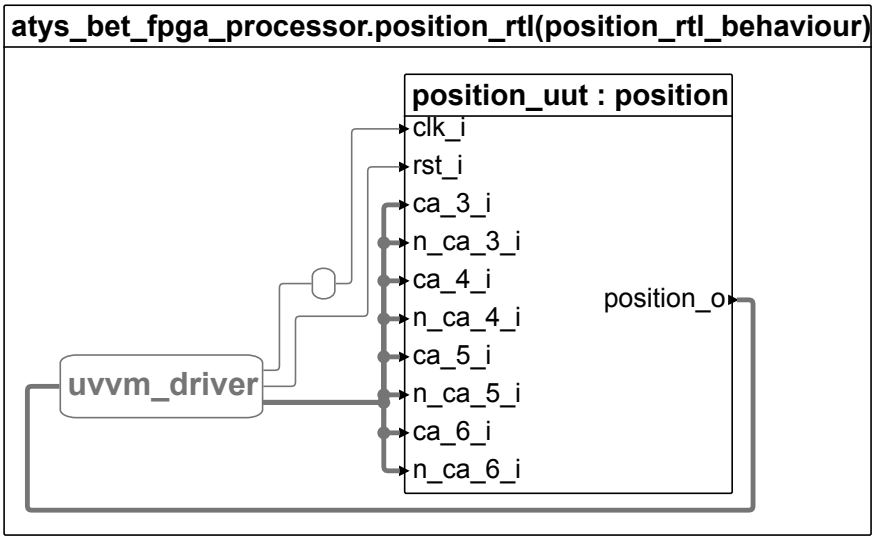


Figure 4.1: UVVM simulation setup showing the input driving and output sensing for the Position Acquisition module of the ATyS design (diagram generated by Sigasi from the design code of the project)

4.2.2 Open Source VHDL Verification Methodology

The application of OSVVM allowed for the 100% coverage, required by SIL, to be proven automatically. Functional coverage was applied to the design to ensure that the test plan had been completed. By setting up transition bins, the states of the UUT were tracked throughout the simulations and a calculation of the coverage was performed by the OSVVM process. Example OSVVM code, showing the mapping of state machine transitions for the example UUT is shown in Appendix Item A.8. The operation of this tracking can be seen in Figure 4.2 where the OSVVM coverage process monitors the UUT throughout the simulation. This diagram, also generated using Sigasi, shows how the OSVVM process monitors the stimulus provided by the UVVM process as well as the states of the example UUT. This was applied to all state machines in the design to ensure that all valid states of the system were realised in the simulation process.

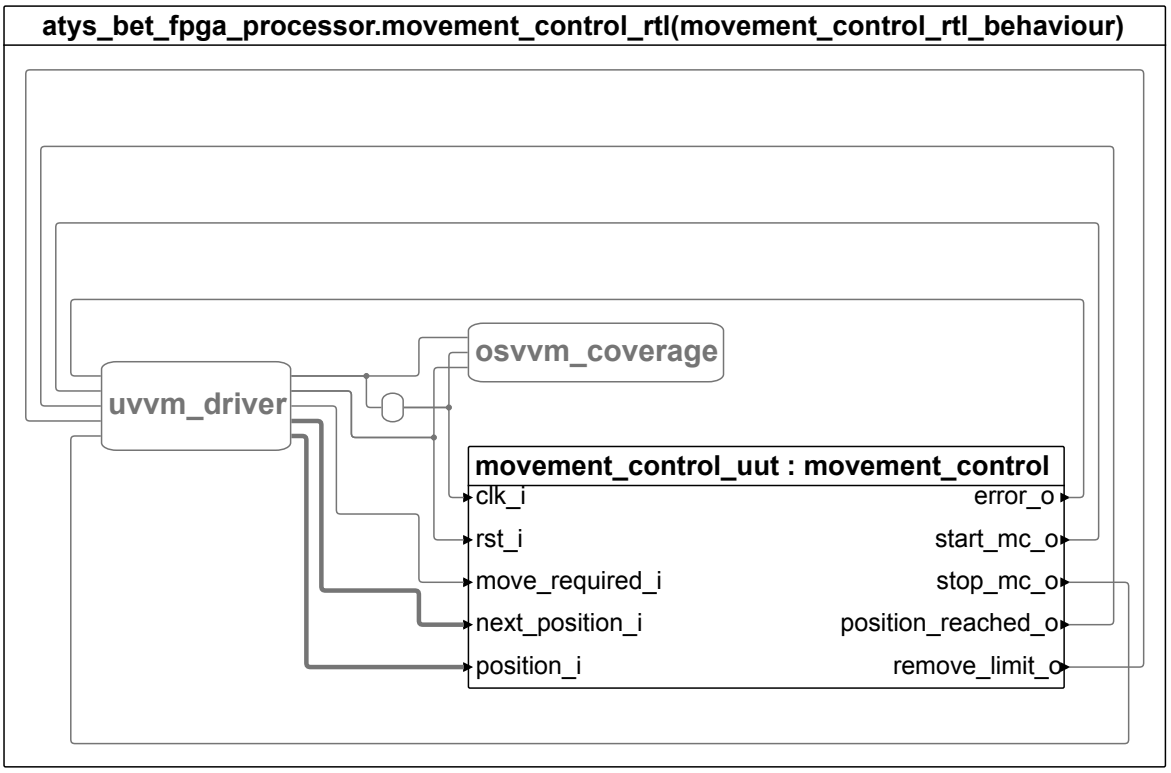


Figure 4.2: OSVVM process tracking the coverage on a UVVM driven simulation (diagram generated by Sigasi from the design code of the project)

4.3 Verification of Requirements

In accordance with the V-model, the project requirements were directly mapped into the verification plan. This was to ensure a fully validated design. The requirements table (Table 3.1) in the Design section shows each of the project requirements. It is divided into requirements for each of the four modules of the system architecture. The requirements displayed in this table have been mapped directly into the verification plan shown in Table 4.1. It shows the requirement identifier, a description of the tests carried out and a summary of the results for each module requirement. The verification results for each module are presented in the following sections.

4.3.1 Input Processing Verification

The primary input stage of the design, the Input Processing Module, comprised of a number of sub-modules; each of which were verified in isolation. First of all, the response to different position readings was verified against the specification. The simulation results for the Position sub-module are shown as an example in Figure 4.3. The simulation shows the

ATyS Motor Control Verification Table		
Requirement	Description	Verification Results
Input R1	Provide all possible input combinations and compare the results with a look-up-table	A sweep through all command inputs was done and the translation was verified against the original specification
Input R2	Provide all possible input combinations and compare the results with a look-up-table	Each position was input and the translation was verified against the specification (see Figure 4.3)
Input R3	Check all 27 combinations of position and command and verify the correct movement is taken	Each combination produced the correct direction and control signals
Input R4	Run through each of the possible control stages and ensure the correct timings and values of processes	Timings and control signals match the specification
PWM R1	Check the duty-cycle output for a motor movement over each of the defined stages	The duty-cycle value produced in each of the stages was verified against the specification
PWM R2	Model a stalled motor in simulation (motor resistance measured to be 47Ω) and ensure regulation around the correct current value	The simulation model allowed the regulation process to be fine-tuned and it settled on an accurate value (see Figure 4.6)
PWM R3	Check the frequency and duty-cycle of the output against the specified input duty-cycle value	A range of different duty-cycles were provided to the generator and the output was verified to match exactly
PWM R4	Verify that the current flag is raised half-way through the generation process	A range of different duty-cycles were input and each produced a flag exactly in the middle of the on-time
PWM R5	Input the range of valid voltages (150V - 333V) and verify that the output matches the formula	A sweep of the input values obtained a response which matched the specification
ADC R1	Model the RC input and supply an analogue value and verify the digital output	ADC accurately simulated for a sawtooth input (see Figure 4.5)
ADC R2	Model the RC input and supply an analogue value and verify the digital output	ADC accurately simulated for a sawtooth input (see Figure 4.5)
Drive R1	Verify that the correct IGBT is driven for each direction (see Figure 3.5 which shows the required IGBT signals for each movement)	The motor control process was simulated and the IGBT stimulus was verified against the expected signals (see Figure 4.8)
Drive R2	Verify that the correct Thyristor is driven for each direction (Figure 3.5 shows the required Thyristor signals for each movement)	The motor control process was simulated and the Thyristor stimulus was verified against the expected signals

Table 4.1: Verification table for the ATyS motor control requirements

buttons on the bottom eight rows, which were stimulated using UVVM, going through each valid combination. The value for the position is then calculated. It can be seen visually that the calculated position (output_position_s) is exactly the same value as the stimulated test case (position_in_s). The simulation also displays the behaviour of intermediate signals (buttons_read_in_s). The numbers in the simulation do not correspond to the actual position due to the position encoding scheme, for example, Position one is represented by a number three in the encoding scheme. The translation logic can be seen in Appendix Item A.1.

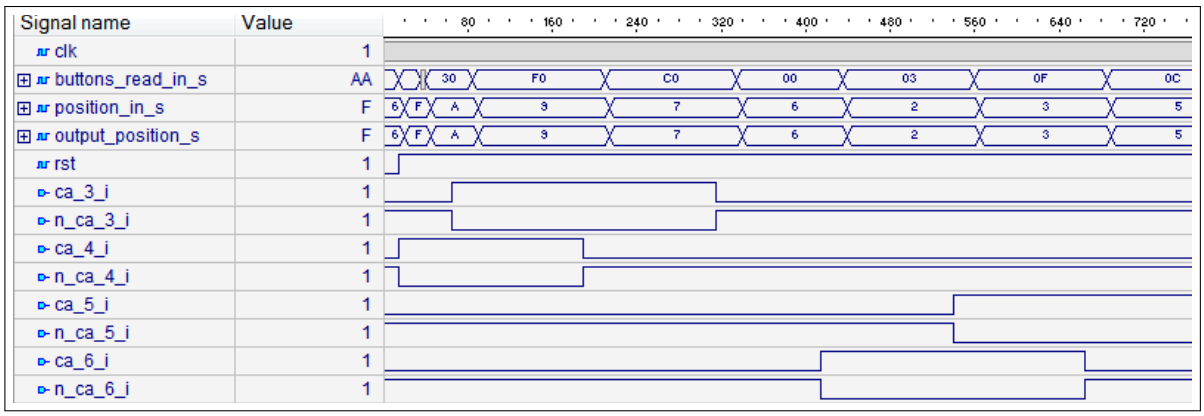


Figure 4.3: Simulation of the Position Acquisition module for each input combination (requirement: Input R2)

In addition to the simulation results for each module, value checking was also applied to the design using UVVM to ensure the correct value is calculated. The results of the UVVM checking for the Position sub-module are shown in Figure 4.4. The textual output corresponds to the same simulation displayed for the Position sub-module. As the simulation progresses, the values at any time along the simulation can be verified and the results are summarised at the end of the simulation. If a value is found to be incorrect for any of the checks then it is flagged and the actual value is presented which aids debugging. The textual response for this module shows each of the stages of the verification plan. The module was swept through from Position one, through each intermediate position, to Position two. Example UVVM code for a single check, which corresponds to a line in the summary table, is shown in Appendix Item A.7. The code shows the stimulation, value checking and timing verification that are applied using UVVM methods. This pattern was applied for each position in the simulation. The results were also checked for invalid position inputs. Timing verification, ensuring the filter is working as expected, is also performed and the results are presented.

The other sub-modules in the Input Processing module were also simulated and verified. The same process that was applied to the Position Acquisition sub-module was applied in the verification of Command interpretation. Each of the valid commands were swept through and the output of the Command sub-module was monitored and verified using UVVM. The

```
     0.0 ns  TB seq.                    ÿÿÿÿStarting clock TB clock
 15000.0 ns  TB seq.                    ÿÿÿÿPulsed to 0 for 15 us. 'Pulsed reset signal'
 65000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"2"'. 'Check output is position 1'
115000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"3"'. 'Check output is position 1 to 0'
165000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"5"'. 'Check output is position 0 from 1'
215000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"6"'. 'Check output is position 0'
265000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"7"'. 'Check output is position 0 from 2'
315000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"9"'. 'Check output is position 1 to 0'
365000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"A"'. 'Check output is position 2'
415000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"F"'. 'Check output is invalid position'
430000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"F"'. 'Checking value has not updated yet'
445000.0 ns  TB seq.                    ÿÿÿÿcheck_value() => OK, for slv x"6"'. 'Checking the value has updated'
450000.0 ns  TB seq.                    ÿÿÿÿStopping clock TB clock
```

Figure 4.4: UVVM verification output example for the Position module simulation

other sub-modules in the Input module relate to the control of direction and timings of the movement. For the verification of the sub-module for requirement Input R3, responsible for the decision making in terms of movement and direction, each combination of command input and position was provided and the expected result was analysed. The response included raising a flag to indicate that movement is necessary and providing the correct direction of movement. The sub-module which was responsible for the timing control of the motor control process was verified using UVVM timing functions. The UVVM functions wait for a given event and then can verify that the timing of that event matches the required time. Alternatively, they can wait a given time and verify that the value is correct at that time. This was applied over several of the application cases including a successful movement first time, a successful movement upon a retry and an unsuccessful movement in order to verify that the response of the design matched the specification for each case.

4.3.2 ADC Verification

In order to verify the ADC, an analogue input must be modelled by the simulation. This required creating a model of the RC input stage. The ADC, which was obtained from Lattice, came with a verification file. In this verification file, integrators are used to simulate the RC input stage and the comparitor at the input of the FPGA. A saw-tooth input was generated in this simulation in order to verify the response of the ADC.

The simulation results in Figure 4.5 show the response of the ADC to this saw-tooth input. The response can be seen to follow the saw-tooth ramp closely. When the saw-tooth resets and falls from the maximum value to the minimum value, the ADC can be seen to follow the value with the behaviour of an averager as it falls through an intermediate value before increasing again. UVVM was applied to this simulation throughout to ensure the value is within a one-bit tolerance of the input value for each reading. The ADC produces periodic values and holds the value until it has calculated the next one so the response is more stepped than the simulated analogue input.

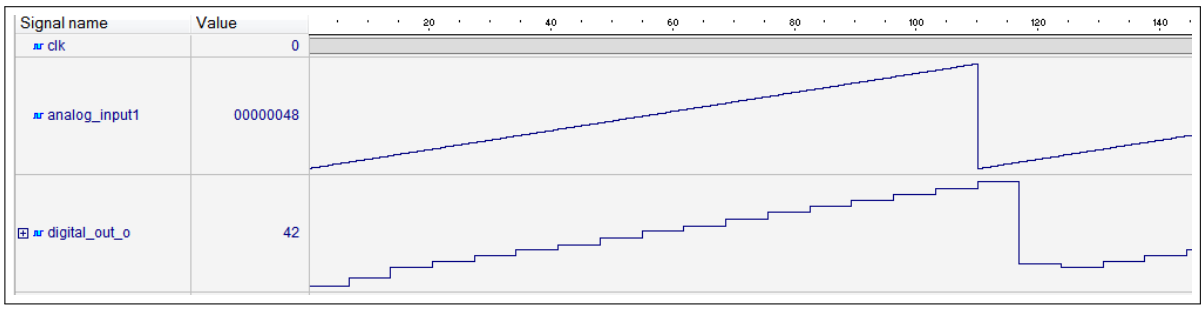


Figure 4.5: ADC simulated response to a saw-tooth waveform (requirements: ADC R1 and ADC R2)

4.3.3 PWM Regulation Verification

In order to test the PWM regulation scheme, the regulation loop, shown in Figure 3.4, must be simulated. This includes modelling the motor in order to verify the response to an actual motor. Under normal switching conditions, regulation of motor current is not needed as the current is consistently below 1.5A. The regulation process operates when the motor is stalled or blocked. In order to verify the response of the motor to these conditions, a stalled motor has to be modelled in the simulation. It is difficult to accurately simulate the actual response of an analogue motor controlled by a VHDL design[23]. Matlab can be used for mixed-signal simulations[23], however; it will not be adopted by this project as a simplified model of the stalled motor was sufficient for regulation testing and fine-tuning. The model consisted of a VIR calculation based on the PWM signal (V), the resistance of the motor (R) and the calculated resulting current (I) which was fed back into the regulation unit. UVVM was used to perform the calculations required to model the motor and provide the new stimulus.

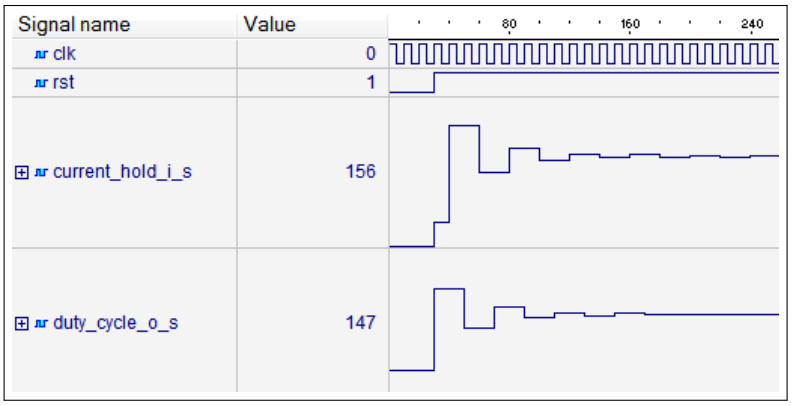


Figure 4.6: PWM regulation simulation of a stalled motor modelled in UVVM (requirement: PWM R2)

The simulation results for the PWM Regulation module are shown in Figure 4.6. The diagram shows the response of the duty-cycle to the measured current for the regulation process. The

regulation module aims to keep the current value at a nominal value. It achieves this by adjusting the PWM duty-cycle, which drives the motor, based on the measured current from the ADC. In this simulation, the current is calculated from the effect of the duty-cycle value on a fixed resistance. It can be seen in the diagram that the initial value of current is measured to be too low, compared to the regulation value. In response to this, the regulation module increases the duty-cycle. The response of the motor model produces a current which is now above the regulation value, the duty-cycle is then reduced accordingly. This process repeats until the values stabilise and the regulation current is achieved. In this example, the regulation current was set to the ADC value of 155 and the value alternates between 155 and 156. The response, which can be seen in this simulation, is typical to that of a proportional controller. UVVM checking was also applied to ensure that each of the PWM Regulation requirements were met by the design. The additional sub-modules which were responsible for providing the correct control signals to the regulation process were generated using UVVM for this simulation and were verified separately.

The PWM module design contains a number of state machines in order to generate the required control signals; each of which needs to be fully verified. OSVVM was applied to each of them to ensure that each machine reaches each possible state during the simulation. The application of state transition monitors provides instant and automatic feedback. An example response from OSVVM, applied to the PWM control state machine, can be seen in Figure 4.7. The response shows a calculation of the state coverage. In this case the coverage is 100%. The results for each transition are printed in the response. This includes a count of the number of transitions between states throughout the simulation for each mapping and a flag indicating that the transition occurred. The OSVVM code which set up these transition bins can be seen in Appendix Item A.8. This instant feedback ensures that the test plan has been completed by ensuring that all states are realised in the simulation. OSVVM was applied to all state machines in the design and identified cases which were not originally covered in the test plan. The test plan was modified in iterative cycles and the final test-suite provided 100% state coverage.

```
# EXECUTION:: NOTE   : State Coverage = 100
# EXECUTION:: Time: 1097615 us,  Iteration: 0,  Instance: /motor_control_rtl,  Process: uvvm_driver.
# KERNEL: %% WriteBin: Test FSM Transitions
# KERNEL: %% Transition from Wait State to Thyristor Startup  Bin:(0) (1)   Count = 3  AtLeast = 1
# KERNEL: %% Transition from Thyristor Startup to PWM Startup  Bin:(1) (2)   Count = 3  AtLeast = 1
# KERNEL: %% Transition from PWM Startup to PWM Increase  Bin:(2) (3)   Count = 2  AtLeast = 1
# KERNEL: %% Transition from PWM Startup to Brake  Bin:(2) (5)   Count = 1  AtLeast = 1
# KERNEL: %% Transition from PWM Increase to PWM Regulation  Bin:(3) (4)   Count = 1  AtLeast = 1
# KERNEL: %% Transition from PWM Increase to Brake  Bin:(3) (5)   Count = 1  AtLeast = 1
# KERNEL: %% Transition from PWM Regulation to Brake  Bin:(4) (5)   Count = 1  AtLeast = 1
# KERNEL: %% Transition from Brake to No Command  Bin:(5) (6)   Count = 3  AtLeast = 1
# KERNEL: %% Transition from No Command to Wait State  Bin:(6) (0)   Count = 3  AtLeast = 1
```

Figure 4.7: OSVVM state coverage results for the PWM control state machine (requirement: PWM R1)

4.3.4 Motor Driving Verification

The Motor Driving module is required to drive the correct IGBT and Thyristor combinations based on control signals provided by other modules. The behavioural response to each of the control signal combinations must be verified against the specification to ensure that it is not possible for invalid combinations of IGBTs and Thyristors to be driven. The output control signal combinations from the Input Processing and PWM Regulation modules were generated using UVVM and the response was verified using the same methodology.

The simulation results for the IGBTs during a motor movement can be seen in Figure 4.8. The control signals which were created as part of the motor control process were simulated. The conditions for each respective IGBT being driven are checked by the simulation. The PWM signal, produced by the simulation to model the accurate behaviour of the system, was sent to the IGBT for the correct direction indicated. For example, the simulation shows the completion of one of the sub-module requirements for the IGBT: if the IGBT is enabled (en_db_o_s is set to 1) then the PWM generated by the regulation process (pwm_i_s) should be passed directly to the IGBT for the given direction (pwm_db_o_s). Each of the combinations of the control signals was verified using UVVM. The braking phase can also be seen in the diagram where the brake_i_s signal is driven. For this phase, which is required to stop the motor, each of the IGBTs are driven.

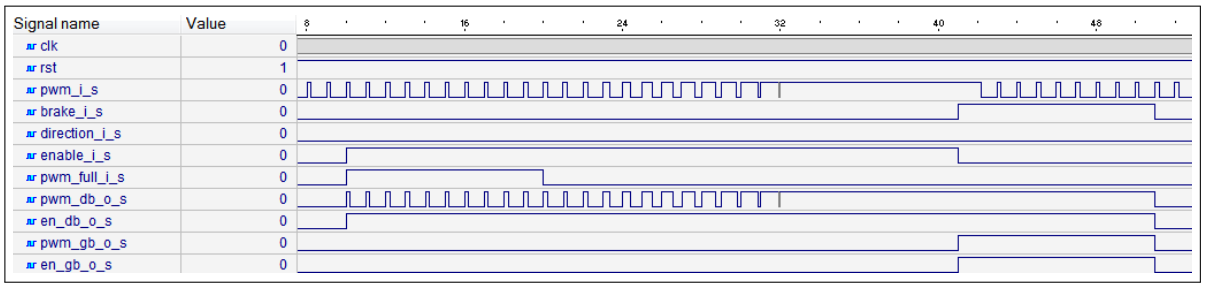


Figure 4.8: Simulation of the IGBT throughout the simulated motor control process (requirement: Drive R1)

4.4 Further Verification

Behavioural verification of each module of the design was performed using UVVM; however, there were some verification activities, required by SIL, which were not completed in simulation for this project. These activities include system-level verification which is related to the interaction between different modules. Additionally, gate-level simulations which give insight into the synthesised timings of the design were not performed using UVVM.

4.4.1 System Verification

System behaviour can be validated by verifying multiple modules and their interactions. For this project, this process was done on the physical prototype. As described in the prototype creation Section 3.5.3, the modules were tested on the PCB by incrementally integrating sub-modules. By verifying this using UVVM the benefits of automatic verification could be realised. This would provide the design with a more comprehensive test-suite which verifies each level of the module. Any number of modules can be incorporated into a simulation for UVVM. This could scale up to the entire design where the requirements of the entire end-to-end system can be validated. This is the final process of the V-model.

4.4.2 Gate-level Simulations

Gate-level simulations can be performed on the synthesised design. This allows accurate modelling of the timing behaviour of the system. Where register-transfer-level simulations provide behavioural verification, gate-level simulations provide insights into true time-based behaviour. Behavioural verification was the focus for this project. The prototyping activities on the physical PCB gave insight into and verification of the general timing behaviour of the system; however, by applying verification methodologies to the gate-level design, the timing verification of the design could be realised.

4.5 Verification Conclusion

Verification methodologies were systematically applied to directly verify the project requirements. UVVM was used to verify the functional behaviour of each modules in the design to ensure that they matched the specification. The automated value checking of UVVM reduced the need for manual inspection of simulation graphs and allowed for instant verification feedback. OSVVM was applied to each of the state machines in the system to ensure that they were all fully covered by the simulation. OSVVM was applied to map each of the transitions of state machines in the design to provide coverage metrics for the simulations. The verification of the design validated all of the project requirements with a 100% state coverage. System and gate-level simulations were applied on the physical prototype of the project. By applying UVVM to these verification areas, the benefits of methodologies could be realised for integration and timing behaviour. All of the modules in the design were verified individually and each of the requirements were directly mapped into the verification plan providing a comprehensive coverage of validation of the requirements of the design.

5 | Results

The behaviour of the alternative FPGA solution needs to accurately replicate the behaviour of the existing microcontroller solution. There are two behavioural processes which were implemented by the prototype: the behaviour for a successful movement from one position to another and the regulation process which limits the current. The first of these scenarios occurs during normal operation to transfer the connection from one power supply to another. The regulation process must be used to limit the motor current in the event of a stalled or blocked motor. As part of the results for this project, the behavioural response to each of these scenarios is compared. The motor control function is compared between the two solutions in Section 5.2. The response of the current regulation process is compared in Section 5.3.

The motor control process output is a PWM signal of a duty-cycle which is calculated and generated by the ATyS processor. This duty-cycle value is compared for each of the solutions as part of the results. Additionally, the motor current is also displayed and compared which represents the response of the motor. For each of these tests, the data was taken directly from the Oscilloscope. One million points were taken over five seconds for each reading. The duty-cycle calculations were performed in Excel using the times between the rise and fall of the supplied signal.

5.1 Experimental Setup

The experimental setup for each of the tests can be seen in Figure 5.1. The equipment used for this experiment included an AC power supply, an oscilloscope, an ATyS casing and an ATyS command interface. Each of the PCB solutions being compared were placed into the same model of ATyS casing for experimentation. The oscilloscope had two inputs for each of the experiments: the motor current which was measured from the connection between the rotor and the stator, and the voltage supply to the IGBTs which was measured directly on the PCB. The supply voltage was set to 250V for the motor movement test and was reduced to 200V for testing the regulation processes. The same model of motor was used for both the

Figure 5.1: The experimental setup for the FPGA vs microcontroller comparisons

FPGA and microcontroller boards. For the stalled motor test, the rotor from one motor was connected to the stator of the other to prevent any movement. The same ATyS switch, the load for the motor, was used for each set of results.

5.2 ATyS Movement Test

The core functionality which had to be replicated by the design in this project is the motor control of the ATyS system. This safety function allows the power supply connection to be transferred. The first test will be a comparison between how each of the processors, microcontroller and FPGA, perform this core activity.

As described in the design section, the motor control processes follow a number of defined phases which determine the PWM provided to the motor. The motor control contains five discrete phases: PWM startup, PWM minimum, PWM increase, PWM regulation and braking. These phases are represented as states in a state machine in the FPGA design code which is shown in Appendix Item A.6. The stimulation of the motor during the control process has

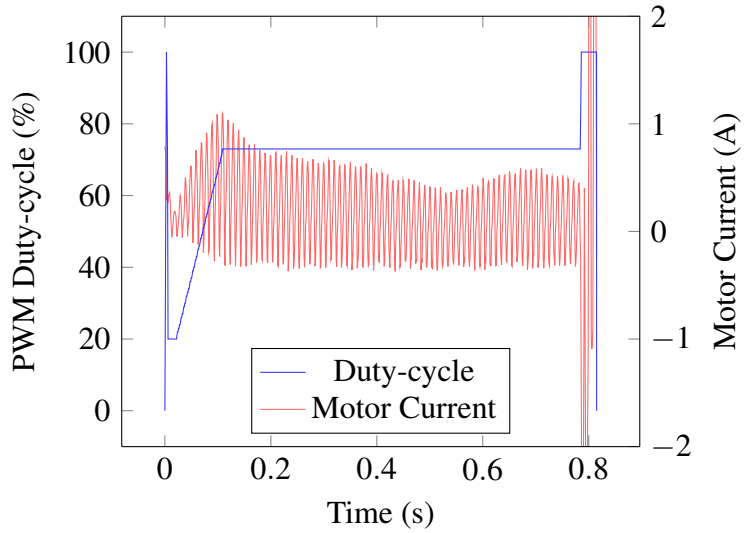
been defined for previous iterations of the ATyS device. An initial startup phase is used in order to activate the IGBTs and Thyristors required. The PWM, which supplies the IGBTs, is then set to the minimum for a short period before being increased to the maximum calculated value. This value is held until the required motor position is reached. At this point, braking is applied to each of the IGBTs. The results of the test, which involved the movement from position one to position two, are shown in Figure 5.2.

The microcontroller motor control process starts with a preset startup phase which involves the transistors activation before the duty-cycle is reduced to 20%. This is followed by increasing of the PWM duty-cycle gradually to the maximum value. The maximum value is calculated based on the value of the supply voltage. In this experiment, the maximum PWM was calculated to be 75%. When the maximum value is reached then the regulation phase is triggered. However, under normal conditions and the conditions for this test, the motor current never increases above the regulation value of 1A and so no regulation is performed. Therefore, throughout this phase, the duty-cycle is held constant. This phase ends when the correct motor position is reached. Upon reaching the position, the braking phase is triggered in which both IGBTs are driven full. This causes a spike in the current in order to stop the motor.

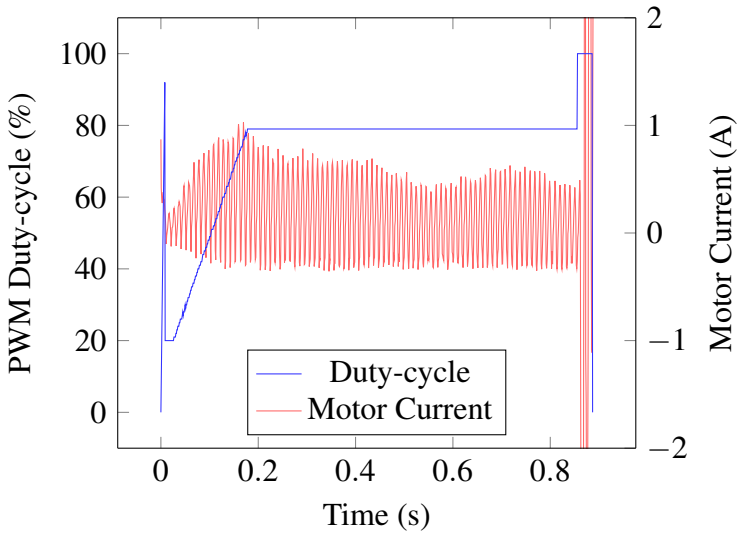
In the graph showing the response from the FPGA system to a motor movement, the same stages of motor control can be seen. It starts with an activation spike in duty-cycle. The PWM then drops to the same minimum value of 20%. The “increase phase” is done over a longer time in the FPGA. This is due to the settings in the control algorithm. The maximum value, for the same voltage, was calculated at 80% and so this was the constant duty-cycle value for the regulation stage. The same current spike can be seen in the FPGA solution.

The same stages can be seen in each of the designs. Some of the calculation details in the FPGA solution differ slightly. For example, the maximum PWM calculation and the slope of the PWM increase. The current response is similar in each motor control process. The current follows an initial peak in the increase phase and then stabilises at around 0.5A before spiking at the braking phase of the control.

Overall, the current response for the microcontroller and FPGA solutions are almost identical. They follow the same distinct phases. Some of these phases differ slightly as the calculation of different values, for example, the maximum PWM differs. The rate at which the PWM increases is slower in the FPGA. This could be updated in the control algorithm calculation to obtain the same response.



(a) Micro-controller Motor Control



(b) FPGA Motor Control

Figure 5.2: Graphs showing the motor response to the control sequence applied by the microcontroller (a) and FPGA (b) for a successful movement

5.3 Stalled Motor Test

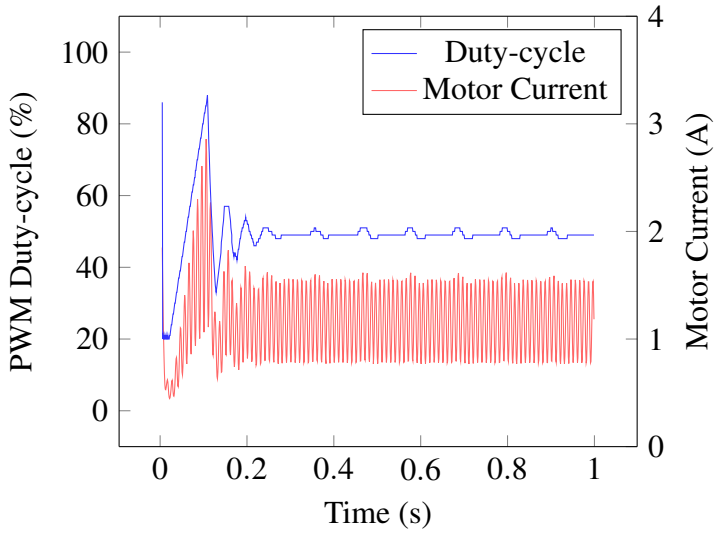
If the motor of the ATyS is blocked then the current can spike to very high values. The current therefore needs to be regulated to prevent these damaging high currents. In order to test this behaviour, a stalled motor was used. This was achieved by connecting the rotor of one motor to the stater of the other. With the exception of switching the motor for a stalled version, the setup is the same as for the previous test and can be seen in Figure 5.1. For this test, the supply voltage was set to 200V. This means the calculation of the maximum PWM is different when compared to the motor movement tests.

The regulation process is intended to regulate the current around 1A. The device works in AC and so there will be 50Hz variations but these should be contained by the process. The response of the microcontroller to these conditions can be seen in Figure 5.3a and the FPGA response can be seen in Figure 5.3b. In these graphs, the PWM Duty-cycle produced by the processor, which drives one of the IGBTs, is graphed in blue against time. This allows the regulation process, and it's effect the duty-cycle, to be visualised. The motor current, shown in red, allows for the effect of the regulation process to be seen.

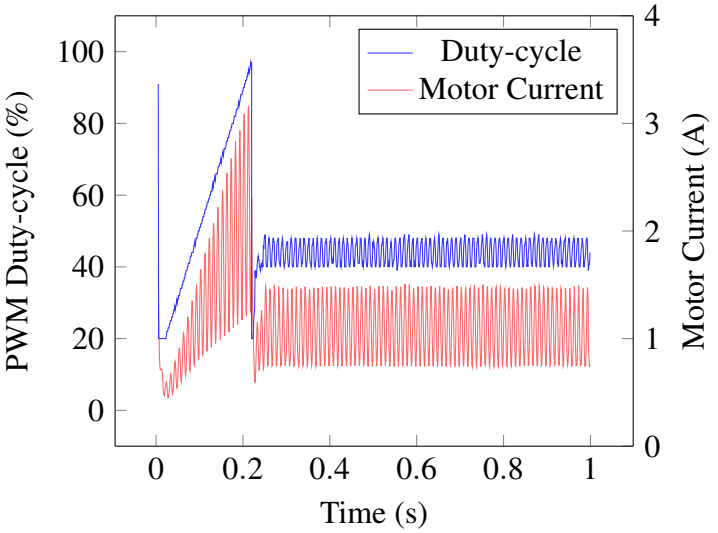
The microcontroller response initially follows a preset startup phase. This is the same as the process for the first test as no regulation is done in this phase. The regulation phase is triggered by the reaching of the maximum PWM value which is calculated by the regulation process. When this phase is complete, the motor current is higher than 1A, around 3A, and so the system reacts and drops the duty-cycle of the PWM to 35% which reduces the power being supplied to the motor. The current reduces due to this action and so the PWM duty-cycle increases in response. The behaviour of a proportional controller can be seen. It continues to overshoot before settling on a value of around 50% duty-cycle. The current regulates consistently around 1A with some periodic variations.

The FPGA response also starts with the preset startup phase which increases the current over the desired limit. Once the regulation process starts, the duty-cycle is reduced dramatically to 20%. The duty-cycle is then increased in response. The value then toggles between 40% and 50% in order to regulate the current. The current response regulates consistently around 1A throughout the regulation phase.

For this test, the regulation process of each of the processors is being compared. Each of the regulation processes drops the duty-cycle in response to the initially high current. The motor current for each of the solutions is around 3A after the preset startup phase. The current response to the drop in duty-cycle, performed by each of the processors, is the same. After the initial response, the microcontroller acts as a proportional controller and both the duty-



(a) Micro-controller Current Regulation



(b) FPGA Current Regulation

Figure 5.3: Graphs showing current regulation process in the microcontroller (a) and FPGA (b) by applying the control process to a stalled motor

cycle and the current settle around a value after a few overshoots in the regulation process. In the FPGA, the behaviour after the initial drop is slightly different. The value increases and begins to regulate around 1A more quickly and without any overshoot.

Once the values begin to regulate, the processes are slightly different. Each of the solutions regulates the current successfully between 0.5A and 1.5A. In the microcontroller, periodic variation is seen in the duty-cycle but it remains fairly constant. The variations that do occur can be seen to correspond to variations in the motor current. In the FPGA, the duty-cycle varies more throughout the regulation process. The device toggles the response continually between 40% and 50% in response to the 50Hz fluctuations. The response of the current to this regulation process is much smoother than in the microcontroller solution.

Overall, the current response of the two systems to a stalled motor is almost identical (see Figure 5.3). In these conditions, spikes in current could be dangerous and so the current must be regulated. The FPGA regulation process was modelled from the existing microcontroller and so the similarities and differences have been compared as part of this test. It was found that, although the regulation processes differ, the current response is almost identical. The FPGA reaches the regulation value, 1A, quicker than in the microcontroller process. Additionally, the current variations present in the microcontroller solution are eradicated in the FPGA solution.

5.4 Results Conclusion

The behaviour for both a successful and an unsuccessful motor movement has been replicated successfully in the FPGA solution. The current response to a successful movement is almost identical for the microcontroller and the FPGA. The device successfully performs the movement with the same process stages as the microcontroller system. The current regulation scheme, despite slight differences in behaviour, is also well matched. The current regulation in the FPGA solution reaches the regulation value faster than in the microcontroller and is a much smoother regulation.

6 | Evaluation

In this section, the results of the project will be evaluated against the initial aims and objectives set out in Section 1. The overall aim of the project was to enhance the functional safety of the ATyS device through the design and development of an FPGA-based system. This was to be done by following the standardised safety integrity level (SIL) framework. The extent to which this overarching aim was achieved is discussed below in Section 6.1. Using the SIL framework, the functional safety of the device will be discussed. A number of objectives were laid out in the introduction in order to achieve this aim. The extent to which these objectives were achieved is discussed below in Section 6.2 along with a suggestion of how they can be built upon in the future.

6.1 Evaluation of Project Aims

The overarching aim for the project was to investigate the advantages of an FPGA-based ATyS processor solution with a focus on the functional safety. Safety certification is a process of verification that the system developed follows the relevant SIL guidelines. In this section, the evaluation of the achievement of this aim will relate to how closely these guidelines were followed throughout the development phases of the FPGA-based ATyS prototype. An implicit aim of the project was to replicate the safety-related behaviour of the existing microcontroller solution.

6.1.1 Safety Integrity Level Evaluation

As described above, the SIL framework standardises the claim of functional safety of a device application and there are two sets of processes which need to be followed to achieve this[13]. The first of which is related to the prevention of systematic errors through following the recommended processes. The second set of processes are concerned with the detection and correction of online errors. Each of these sets of processes must be followed in order to

claim a safety integrity level. The success in achieving each of these is discussed in this section.

The recommended V-model development process was followed closely throughout the design of the FPGA-based system. The logical flow from safety requirements elicitation through derivation of system architecture and down to the module implementation was performed in this project and is described in the design section. The initial verification phases of the process were followed throughout this project. Each module was functionally verified against its requirements. System-level verification was conducted on the physical prototype built as part of the project. The overall validation phases of the V-model were also performed on the physical prototype rather than in an automated test-bench. This allowed the behaviour to be compared with the existing solution.

The V-model was a powerful tool in driving the development in the project. As well as encouraging verification throughout development, the V-model process required the understanding of project requirements before proceeding with the design. This was especially important because of the nature of this project. The safety requirements, which were derived from the system requirements, had to be well understood in order to contain the scope of the prototype and focus on safety-related aspects of the design, rather than a straight functional migration from microcontroller to FPGA.

Diagnostic measures were planned as part of this project. As discussed above, redundancy architectures and on-line verification components can be added to FPGA designs without impairing the performance[23]. These diagnostics, which are required by the standards in order to detect and control errors, were not implemented as part of this project and form part of the identified further work (see Section 6.2 below).

Overall, the foundations for claiming a SIL for this device application have been laid through the adoption of recommended processes. With the completion of a number of additional activities, such as gate-level and system-level verification and the addition of diagnostic components, a safety integrity level could be claimed[13].

6.1.2 Microcontroller Behaviour Execution Accuracy

As seen in the results section, the FPGA-based solution has replicated the behaviour of the existing microcontroller for both a successful and unsuccessful motor movement. The successful motor movement, which was identified as the safety-function of the device, produced a similar response to the microcontroller system. The FPGA system designed, developed and verified was able to perform the core behaviour. The additional current regulation, which

allows the device to control the behaviour under stalled and blocked conditions, was also replicated in the FPGA solution. The FPGA regulation process produced a smoother response from the current and it reached the nominal value more quickly after regulation was triggered. This prototype, which uses FPGA technology, now represents a safer alternative, as described in Table 2.1, with the same behaviour.

6.1.3 Adoption of Verification Methodologies

Although simulations were used, the main form of verification in this project was done by observing the textual output of the verification methodologies (see Figure 4.4 for an example). UVVM allowed for value checking in a simulation and provided instant feedback which could be immediately compared to the fulfilment of project requirements (see Table 4.1). This automation of testing prevents time-consuming and error-prone manual checking of simulation results. This speeds up and enhances the verification process and allows for reproducible results. OSVVM was also adopted to ensure that the system was being fully verified by tracking the states of the system during the verification process. The verification-suite achieved 100% state coverage for this project. This methodology provides key coverage metrics which are required by the SIL process.

6.2 Evaluation of Project Objectives

There were six project objectives outlined in the introduction of this report. The completion of these project objectives has been evaluated in Table 6.1. This table displays the objective and an evaluation of the extent of their completion. A suggestion of how each objective could be built on in the future is also given in the table.

In the Evaluation column of Table 6.1, the extent of completion for each of the objective is discussed. This column describes the final state of the prototype, including the size of the design. The design consisted of 2967 lines of VHDL code and the verification code consisted of 4034 lines of VHDL code. This design was made up from 21 entities. When synthesised, the design consisted of 35 design units. This is because a number of the units were used multiple times in the design. Additionally, the Evaluation column notes that the safety-function behaviour was accurately replicated in the FPGA solution.

Further work, which could be applied to build on the project objectives, is presented in Table 6.1. The table mentions adapting the project and the design to the safety-standards for when they include FPGA specific considerations. It is expected that future safety-standards will

Evaluation of Project Objectives		
Objective	Evaluation	Further Work
Conduct a literature review	The application of safety-standards to FPGA systems and the application of FPGAs to safety-related and industrial switching equipment were investigated	Investigate alternative technologies which could be used as a solution (for example ASICs) and how an FPGA solution could benefit other products
Analyse the safety standards	The project applied the relevant safety standards and recommended processes for FPGAs (as demonstrated in Sections 3 and 4)	Anticipated changes to the safety standards for the inclusion of FPGA specific considerations should be adopted by the project
Design an FPGA-based alternative solution	The existing specification was used to derive the requirements for the FPGA solution and the V-model process was followed closely to create the design	The entire specification, rather than the safety-related subset, could be designed for including the additional device interfaces and inputs
Develop VHDL code to implement the design	The modules that were designed were implemented using 21 entities (2967 lines of design code) which synthesised as 35 units	Diagnostics, planned as part of this project, could be implemented to further enhance the functional safety of the device
Verify the design	Each of the individual modules were tested directly against the project requirements (4034 lines of verification code)	Gate-level and system-level simulations would provide insight into accurate timing and module interactions
Physically prototype the FPGA-based solution	The safety-function behaviour of the FPGA solution was compared to the micro-controller solution and accurately replicated the outputs	The additional, non-safety-related behaviour could be replicated to produce a functionally identical solution (the design only used 36% of the FPGA logic)

Table 6.1: Evaluation of project objectives and suggestions for further work

have specific mention of FPGA and considerations for their use in safety systems. Additional further work for the project includes the expansion of the requirements from the subset required to perform the safety function to the entire set of requirements for ATyS motor control. The remainder of the requirements could be built into the current design for a future iteration of the FPGA-based system. Diagnostics could be applied to this design to further enhance the functional safety of the device. Example diagnostics are discussed in Section 3.4.6, each of which could be applied to this FPGA design. As mentioned in the table, the design only uses 36% of the logic of the FPGA. The additional space could be used for the application of diagnostics and redundancy architectures. The evaluation table also mentions the completion of gate-level and system-level simulations. Application of these techniques provides insight into accurate timing and system interactions respectively.

6.3 Evaluation Conclusion

Overall, the foundations for claiming a SIL for the ATyS device have been laid. The processes intended to prevent systematic errors were followed closely throughout the design and verification phases. Although each module was verified individually and directly against the project requirements, there are some verification activities (gate and system-level) which must be undertaken before a safety integrity level can be fully claimed.

Each of the objectives of the project was successfully completed and suggestions for how each of them can be built upon in the future have been given. The safety function of the ATyS motor control processor was successfully replicated using an FPGA. The PWM regulation behaviour of the FPGA was faster to respond and produced a smoother current response than the microcontroller solution.

7 | Conclusion

Safety integrity levels define a standardised approach to the claiming of functional safety for programmable electronic devices. SIL processes were followed closely in the design, implementation and verification of an FPGA-based alternative solution to motor control in an Automatic Transfer Switch. This included following the V-model for ASIC development and by following the processes outlined in the IEC 61508 safety standard. With the addition of diagnostic components and by performing gate-level and system-level simulations, a safety integrity level could be claimed for this device. Thus allowing the functional safety of the ATyS device to be enhanced and standardised.

Verification methodologies were adopted to automate and provide metrics for the verification process. UVVM was applied and provided automatic verification of the unitary requirements for each module. OSVVM allowed for the coverage of the simulation to be measured to ensure that the full design had been verified. The verification-suite provided 100% state coverage and directly tested each of the project requirements for motor control.

The physical prototype that was developed in this project accurately replicated the behaviour and response of the safety function of the existing microcontroller solution. The FPGA-based ATyS motor control board successfully performed the transfer from one supply of power to another. Additionally, the regulation process in the FPGA-based solution reacted more quickly and performed a smoother regulation of the motor current than in the microcontroller-based system.

The company stipulated the requirement for an FPGA-based Automatic Transfer Switch in order to offer a premium product, enhancing their existing family of commercial ATS products. The verified and tested prototype developed for this project replicated the functionality of the existing commercial product and enhanced the functional safety possibilities of the device.

References

[1] Hayek, A and Boercsoek, J. "Safety-related ASIC-Design in terms of the standard IEC 61508". In: *The third International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO)*. 2013, pp. 16–21.

[2] Börcsök, Josef. "Miniaturized safety systems: A way for future tasks in safety engineering". In: *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*. IEEE. 2015, pp. 1–6.

[3] Azeem, M Qaisar, Ahmed, Sheeraz, Khattak, Amjad, et al. "Design and analysis of switching in automatic transfer switch for load transfer". In: *2016 International Conference on Open Source Systems & Technologies (ICOSST)*. IEEE. 2016, pp. 129–134.

[4] Onipede, Bamidele, Joseph, Samuel, and Odiba, Omakoji. "Developing an Automatic Switch for Home or Industrial Power Supply Changeover". In: *Current Journal of Applied Science and Technology* (2017), pp. 1–7.

[5] Liserre, Marco, Sauter, Thilo, and Hung, John Y. "Future energy systems: Integrating renewable energy sources into the smart power grid through industrial electronics". In: *IEEE industrial electronics magazine* 4.1 (2010), pp. 18–37.

[6] Socomec. *ATyS t - ATyS g*. 2019. URL: https://www.socomec.co.uk/range-automatic-transfer-switches_en.html?product=/atys-t-atys-g_en.html.

[7] Socomec. *ATyS p*. 2019. URL: https://www.socomec.com/range-automatic-transfer-switches_en.html?product=/atys-p_en.html.

[8] Socomec. *Transfer switching technology by Socomec - ATyS (125-3200A)*. 2019. URL: https://www.youtube.com/watch?v=wT7XTB8C_qI.

[9] Jeppesen, Benjamin Peter, Rajamani, Meenakshi, and Smith, Kevin Mark. "Enhancing functional safety in FPGA-based motor drives". In: *The Journal of Engineering* 2019.17 (2019), pp. 4580–4584.

[10] Commission, International Electrotechnical. *Functional Safety*. 2019. URL: https://www.iec.ch/functionalsafety/explained/.

[11] Bernardeschi, Cinzia, Cassano, Luca, and Domenici, Andrea. "SRAM-based FPGA systems for safety-critical applications: A survey on design standards and proposed

methodologies". In: *Journal of Computer Science and Technology* 30.2 (2015), pp. 373–390.

[12] Hayek, Ali and Börcsök, Josef. "SRAM-based FPGA design techniques for safety related systems conforming to IEC 61508 a survey and analysis". In: *2012 2nd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)*. IEEE. 2012, pp. 319–324.

[13] *IEC 61508*. 1-7. Edition 2.0. International Electrotechnical Commission. 2010.

[14] Redmill, Felix. "Understanding the use, misuse and abuse of safety integrity levels". In: *Proceedings of the Eighth Safety-critical Systems Symposium*. Citeseer. 2000, pp. 8–10.

[15] Silva Neto, Antonio Vieira da, Vismari, Lucio Flávio, Gimenes, Ricardo Alexandre Veiga, Sesso, Daniel Baraldi, Almeida, Jorge Rady de, Cugnasca, Paulo Sérgio, and Camargo, João Batista. "A Practical Analytical Approach to Increase Confidence in PLD-Based Systems Safety Analysis". In: *IEEE Systems Journal* 12.4 (2017), pp. 3473–3484.

[16] Conmy, Philippa and Bate, Iain. "Component-based safety analysis of FPGAs". In: *IEEE Transactions on Industrial Informatics* 6.2 (2010), pp. 195–205.

[17] Foster, Harry. "2018 FPGA Functional Verification Trends". In: *2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV)*. IEEE. 2018, pp. 40–45.

[18] Cilardo, Alessandro. "New techniques and tools for application-dependent testing of FPGA-based components". In: *IEEE Transactions on Industrial Informatics* 11.1 (2014), pp. 94–103.

[19] Salewski, Falk and Kowalewski, Stefan. "Exploring the Differences of FPGAs and Microcontrollers for their Use in Safety-Critical Embedded Applications". In: *2006 International Symposium on Industrial Embedded Systems*. IEEE. 2006, pp. 1–4.

[20] Rodríguez-Andina, Juan J, Valdes-Pena, Maria D, and Moure, Maria J. "Advanced features and industrial applications of FPGAs–A review". In: *IEEE Transactions on Industrial Informatics* 11.4 (2015), pp. 853–864.

[21] Monmasson, E, Idkhajine, L, Bahri, I, Charaabi, L, et al. "Design methodology and FPGA-based controllers for power electronics and drive applications". In: *2010 5th IEEE Conference on Industrial Electronics and Applications*. IEEE. 2010, pp. 2328–2338.

[22] Gomes, Luis and Rodriguez-Andina, Juan J. "Guest editorial special section on embedded and reconfigurable systems". In: *IEEE Transactions on Industrial Informatics* 9.3 (2013), pp. 1588–1590.

[23] Dubey, Rahul, Agarwal, Pramod, and Vasantha, MK. “Programmable logic devices for motion control–A review”. In: *IEEE Transactions on Industrial Electronics* 54.1 (2007), pp. 559–566.

[24] Castro, Angel de, Zumel, Pablo, García, Oscar, Riesgo, Teresa, and Uceda, Javier. “Concurrent and simple digital controller of an AC/DC converter with power factor correction based on an FPGA”. In: *IEEE Transactions on Power Electronics* 18.1 (2003), pp. 334–343.

[25] Guzman-Miranda, Hipólito, Sterpone, Luca, Violante, Massimo, Aguirre, Miguel A, and Gutierrez-Rizo, Manuel. “Coping with the obsolescence of safety-or mission-critical embedded systems using FPGAs”. In: *IEEE Transactions on Industrial Electronics* 58.3 (2010), pp. 814–821.

[26] Sandborn, Peter. “Forecasting technology and part obsolescence”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 231.13 (2017), pp. 2251–2260.

[27] Salewski, Falk and Taylor, Adam. “Fault handling in FPGAs and microcontrollers in safety-critical embedded applications: A comparative survey”. In: *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*. IEEE. 2007, pp. 124–131.

[28] Malinowski, Aleksander and Yu, Hao. “Comparison of embedded system design for industrial applications”. In: *IEEE transactions on industrial informatics* 7.2 (2011), pp. 244–254.

[29] Monmasson, E, Idkhajine, L, Bahri, I, Charaabi, L, et al. “Design methodology and FPGA-based controllers for power electronics and drive applications”. In: *2010 5th IEEE Conference on Industrial Electronics and Applications*. IEEE. 2010, pp. 2328–2338.

[30] Jabeen, Shaista, Srinivasan, Sudarshan K, Shuja, Sana, and Dubasi, Mohana Asha Latha. “A formal verification methodology for FPGA-based stepper motor control”. In: *IEEE Embedded Systems Letters* 7.3 (2015), pp. 85–88.

[31] Al-Mahmood, Ali and Opoku Agyeman, Michael. “A study of FPGA-based System-on-Chip designs for real-time industrial application”. In: *International Journal of Computer Applications* 163.6 (2017), pp. 9–19.

[32] Al-Dhaher, AHG. “Development of Microcontroller/FPGA-based systems”. In: *International Journal of Engineering Education* 20.1 (2004), pp. 52–60.

[33] Monmasson, Eric, Idkhajine, Lahoucine, and Naouar, Mohamed Wissem. “FPGA-based controllers”. In: *IEEE Industrial Electronics Magazine* 5.1 (2011), pp. 14–26.

[34] Naouar, Mohamed-Wissem, Monmasson, Eric, Naassani, Ahmad Ammar, Slama-Belkhodja, Ilhem, and Patin, Nicolas. “FPGA-based current controllers for AC machine drives–A review”. In: *IEEE Transactions on Industrial Electronics* 54.4 (2007), pp. 1907–1925.

[35] Salewski, Falk and Taylor, Adam. “Systematic considerations for the application of FPGAs in industrial applications”. In: *2008 IEEE International Symposium on Industrial Electronics*. IEEE. 2008, pp. 2009–2015.

[36] Zhang, Da and Li, Hui. “A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms”. In: *IEEE Transactions on Industrial Electronics* 55.2 (2008), pp. 551–561.

[37] Aime, Martin, Gateau, Guillaume, and Meynard, Thierry A. “Implementation of a peak-current-control algorithm within a field-programmable gate array”. In: *IEEE Transactions on Industrial Electronics* 54.1 (2007), pp. 406–418.

[38] Idkhajine, Lahoucine, Monmasson, Eric, Naouar, Mohamed Wissem, Prata, Antonio, and Bouallaga, Kamel. “Fully integrated FPGA-based controller for synchronous motor drive”. In: *IEEE Transactions on Industrial Electronics* 56.10 (2009), pp. 4006–4017.

[39] Wells, B Earl and Loo, Sin Ming. “On the use of distributed reconfigurable hardware in launch control avionics”. In: *20th DASC. 20th Digital Avionics Systems Conference (Cat. No. 01CH37219)*. Vol. 2. IEEE. 2001, 8B1–1.

[40] Tahoori, Mehdi. “Application-dependent testing of FPGAs”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.9 (2006), pp. 1024–1033.

[41] Naouar, Mohamed Wissem, Monmasson, Eric, Naassani, Ahmad Ammar, and Slama-Belkhodja, Ilhem. “FPGA-based dynamic reconfiguration of sliding mode current controllers for synchronous machines”. In: *IEEE transactions on industrial informatics* 9.3 (2012), pp. 1262–1271.

[42] Acero, Jesus, Navarro, Denis, Barragan, Luis A, Garde, Ignacio, Artigas, Jose I, and Burdio, Jose M. “FPGA-based power measuring for induction heating appliances using sigma–delta A/D conversion”. In: *IEEE Transactions on industrial electronics* 54.4 (2007), pp. 1843–1852.

[43] Baines, Rupert and Pulley, Doug. “A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers”. In: *IEEE Communications Magazine* 41.1 (2003), pp. 105–113.

[44] Kuon, Ian and Rose, Jonathan. “Measuring the gap between FPGAs and ASICs”. In: *IEEE Transactions on computer-aided design of integrated circuits and systems* 26.2 (2007), pp. 203–215.

[45] Zahiri, Behrooz. “Structured ASICs: opportunities and challenges”. In: *Proceedings 21st International Conference on Computer Design*. IEEE. 2003, pp. 404–409.

[46] Drechsler, Rolf, Chevallaz, Christophe, Fummi, Franco, Hu, Alan J, Morad, Ronny, Schirrmeister, Frank, and Goryachev, Alex. “Panel: Future SoC verification methodology: UVM evolution or revolution?” In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–5.

[47] Monmasson, Eric, Idkhajine, Lahoucine, Cirstea, Marcian N, Bahri, Imene, Tisan, Alin, and Naouar, Mohamed Wissem. "FPGAs in industrial control applications". In: *IEEE Transactions on Industrial informatics* 7.2 (2011), pp. 224–243.

[48] Ceesay-Seitz, Katharina. "Automated verification of a System-on-Chip for radiation protection fulfilling Safety Integrity Level 2". PhD thesis. University of Vienna (AT), 2019.

[49] Yun, Young-Nam, Kim, Jae-Beom, Kim, Nam-Do, and Min, Byeong. "Beyond UVM for practical SoC verification". In: *2011 International SoC Design Conference*. IEEE. 2011, pp. 158–162.

[50] *Development of Safety Related Systems: A Lattice Semiconductor White Paper*. Lattice Semiconductor. 2015.

[51] Tallaksen, Espen. "Invited Paper: UVVM – The Fastest Growing FPGA Verification Methodology Worldwide!" In: (2019), pp. 1–6.

[52] Janzen, David and Saiedian, Hossein. "Test-driven development concepts, taxonomy, and future direction". In: *Computer* 38.9 (2005), pp. 43–50.

[53] *Simple Sigma-Delta ADC*. Version 1.5. Lattice Semiconductor. 2018.

[54] Sathyan, Anand, Milivojevic, Nikola, Lee, Young-Joo, Krishnamurthy, Mahesh, and Emadi, Ali. "An FPGA-based novel digital PWM control scheme for BLDC motor drives". In: *IEEE transactions on Industrial Electronics* 56.8 (2009), pp. 3040–3049.

[55] Börcsök, Josef, Hayek, Ali, Machmur, Bashier, and Umar, Muhammad. "Design and Implementation of an IP-core Based Safety-related Communication Architecture on FPGA". In: *2009 XXII International Symposium on Information, Communication and Automation Technologies*. IEEE. 2009, pp. 1–6.

[56] Bitvis. *UVVM - Bitvis AS*. 2020. URL: https://bitvis.no/dev-tools/uvvm/.

[57] Aldec. *Getting started with OSVVM using Riviera-Pro*. 2020. URL: https://www.aldec.com/en/support/resources/documentation/articles/1902.

A | Appendix Items

A.1 Input Position Translation Code

Implementation code example showing the input position translation from the button inputs into an understood position format

```
input_multiplexing : process(rst_i, clk_i)
begin
  if (rst_i = '0') then
    position_in_s     <= POSITION_0;
    buttons_read_in_s <= (others => '0');
  elsif rising_edge(clk_i) then
    buttons_read_in_s <= ca_3_s & not n_ca_3_s & ca_4_s & not n_ca_4_s
        & ca_5_s & not n_ca_5_s & ca_6_s & not n_ca_6_s;
    if (buttons_read_in_s = BUTTONS_POS_0) then
      position_in_s <= POSITION_0;
    elsif (buttons_read_in_s = BUTTONS_POS_1) then
      position_in_s <= POSITION_1;
    elsif (buttons_read_in_s = BUTTONS_POS_2) then
      position_in_s <= POSITION_2;
    elsif (buttons_read_in_s = BUTTONS_POS_1_0) then
      position_in_s <= POSITION_1_0;
    elsif (buttons_read_in_s = BUTTONS_POS_2_0) then
      position_in_s <= POSITION_2_0;
    elsif (buttons_read_in_s = BUTTONS_POS_0_FROM_1) then
      position_in_s <= POSITION_0_FROM_1;
    elsif (buttons_read_in_s = BUTTONS_POS_0_FROM_2) then
      position_in_s <= POSITION_0_FROM_2;
    else
      position_in_s <= INVALID_POSITION;
    end if;
  end if;
end process input_multiplexing;
```

A.2 Input Processing Code

Implementation code example showing the launch control process and the direction and movement determination

```
input_processing : case Launch_State is
        when WAIT_STATE =>
          if (cmd_ready_i = '1') then
            command_in_s <= command_i;
            Launch_State <= INPUT_ACQUISITION;
          else
            Launch_State <= WAIT_STATE;
          end if;
        when INPUT_ACQUISITION =>
          if (command_in_s = RETURN_TO_0) then
            return_to_zero_flag_s <= '1';
          end if;
          current_position_s <=
                  position_i(3 downto 2);
          position_desired_s <=
                  command_in_s(POSITION_ENCODING_WIDTH - 1 downto 0);
          relative_position_s <=
                  position_i(POSITION_ENCODING_WIDTH - 1 downto 0);
          Launch_State <= CALCULATE_MOVEMENT;
        when CALCULATE_MOVEMENT =>
          if (command_in_s = RETURN_TO_0) then
            return_to_zero_flag_s <= '1';
          end if;
          if (position_desired_s < current_position_s)
              or (position_desired_s = current_position_s
              and relative_position_s = RIGHT_OF_POSITION) then
            direction_o       <= RIGHT_MOVE;
            next_position_i_s <= position_desired_s & ACTUAL_POSITION;
            Launch_State      <= MOVE;
          elsif (position_desired_s > current_position_s)
              or (position_desired_s = current_position_s
              and relative_position_s = LEFT_OF_POSITION) then
            direction_o       <= LEFT_MOVE;
            next_position_i_s <= position_desired_s & ACTUAL_POSITION;
            Launch_State      <= MOVE;
          else
            Launch_State <= RETURN_TO_ZERO_CHECK;
          end if;
        when MOVE =>
          move_required_i_s <= '1';
```

```
          if (error_o_s = '1') then
            Launch_State <= ERROR;
          elsif (position_reached_o_s = '1') then
            move_required_i_s <= '0';
            Launch_State      <= INPUT_ACQUISITION;
          end if;
        when RETURN_TO_ZERO_CHECK =>
          if (return_to_zero_flag_s = '1') then
            Launch_State <= ERROR;
          else
            Launch_State <= WAIT_STATE;
          end if;
        when ERROR =>
          Launch_State <= ERROR;
          error_o <= '1';
      end case state_transitions;
```

A.3 ADC Multiplexor Harness Code

Implementation code example showing the multiplexor harness used to control the ADC measurements

```
case Adc_State is
        -- measuring current
        when CURRENT_CALC1 =>
          rst_internal        <= '0';
          delta_i             <= analog_current_i;
          Adc_State <= CURRENT_CALC2;
        when CURRENT_CALC2 =>
          rst_internal <= '1';
          delta_i      <= analog_current_i;
          if (stop_mc_i = '1') then
            Adc_State <= VOLTAGE_CALC1;
          elsif (sample_rdy_s = '1') then
            motor_current_o     <= digital_out_i; -- input topology 0
          end if;
        -- measuring voltage
        when VOLTAGE_CALC1 =>
          rst_internal        <= '0';
          delta_i             <= analog_voltage_i;
          Adc_State <= VOLTAGE_CALC2;
        when VOLTAGE_CALC2 =>
          rst_internal <= '1';
```

```
22          delta_i      <= analog_voltage_i;
23          if (start_mc_i = '1') then
24            Adc_State <= CURRENT_CALC1;
25          elsif (sample_rdy_s = '1') then
26            supply_voltage_o    <= not digital_out_i; -- input topology
27          end if;
28      end case;
```

A.4 PWM Regulation Code

Implementation code example showing the PWM regulation process

```
1       if (en_regulation_i = '1') then
2         current_error_s    := to_integer(unsigned(NOMINAL_CURRENT))
3             - to_integer(unsigned(current_hold_i));
4         duty_cycle_error_s := current_error_s / 2; -- K gain
5         duty_cycle_o_s := to_integer(set_pwm_i) /2 + duty_cycle_error_s;
6         if (duty_cycle_o_s < to_integer(PWM_MIN_VALUE)) then
7           duty_cycle_o <= PWM_MIN_VALUE;
8         elsif (remove_limit_i = '1') then
9           if (duty_cycle_o_s > to_integer(PWM_MAX)) then
10            duty_cycle_o <= PWM_MAX;
11          else
12            duty_cycle_o <= to_std_logic_vector(duty_cycle_o_s,
13                    PWM_RESOLUTION);
14          end if;
15        elsif (duty_cycle_o_s > to_integer(set_pwm_i)) then
16          duty_cycle_o <= set_pwm_i;
17        else
18          duty_cycle_o <= to_std_logic_vector(duty_cycle_o_s,
19                  PWM_RESOLUTION);
20        end if;
21      else
22        duty_cycle_o <= set_pwm_i;
23      end if;
```

A.5 IGBT Driving Code

Implementation code example showing the IGBT driving process of the motor driving module

```
igbt_driving : process (rst_i, clk_i)
    begin
    if (rst_i = '0') then
      pwm_db_o <= '0';
      en_db_o <= '0';
      pwm_gb_o <= '0';
      en_gb_o <= '0';
    elsif rising_edge(clk_i) then
      if (brake_i = '1') then
        pwm_db_o <= '1';
        en_db_o <= '1';
        pwm_gb_o <= '1';
        en_gb_o <= '1';
      elsif (pwm_enable_i = '1' and direction_i = LEFT_MOVE) then
        pwm_db_o <= '0';
        en_db_o <= '0';
        pwm_gb_o <= pwm_i;
        en_gb_o <= '1';
      elsif (pwm_enable_i = '1' and direction_i = RIGHT_MOVE) then
        pwm_db_o <= pwm_i;
        en_db_o <= '1';
        pwm_gb_o <= '0';
        en_gb_o <= '0';
      elsif (pwm_full_i = '1' and direction_i = LEFT_MOVE) then
        pwm_db_o <= '0';
        en_db_o <= '0';
        pwm_gb_o <= '1';
        en_gb_o <= '1';
      elsif (pwm_full_i = '1' and direction_i = RIGHT_MOVE) then
        pwm_db_o <= '1';
        en_db_o <= '1';
        pwm_gb_o <= '0';
        en_gb_o <= '0';
      else
        pwm_db_o <= '0';
        en_db_o <= '0';
        pwm_gb_o <= '0';
        en_gb_o <= '0';
      end if;
    end if;
  end process igbt_driving;
```

A.6 PWM Control State Machine Code

Implementation code example showing the PWM control state machine phases and transition conditions

```
pwm_state_machine : case PWM_State is
        when PWM_MIN =>
          set_pwm_o_s <= PWM_MIN_VALUE;
          en_regulation_o <= '0';
          if (pwm_increase_i = '1') then
            pwm_max_value_s <= top_pwm_i;
            PWM_State       <= PWM_INCREASE;
          else
            PWM_State <= PWM_MIN;
          end if;
        when PWM_INCREASE =>
          en_regulation_o <= '0';
          if (brake_i = '1') then
            PWM_State <= PWM_MIN;
          elsif (set_pwm_o_s >= pwm_max_value_s) then
            set_pwm_o_s <= pwm_max_value_s;
            PWM_State   <= PWM_REGULATION;
          else
            if (increase_flag = '1') then
              set_pwm_o_s <= set_pwm_o_s + PWM_INCREASE_CONSTANT;
            end if;
            PWM_State <= PWM_INCREASE;
          end if;
        when PWM_REGULATION =>
          en_regulation_o <= '1';
          if (brake_i = '1') then
            PWM_State <= PWM_MIN;
          end if;
      end case pwm_state_machine;
```

A.7 Position Module UVVM Verification Code

Verification code example showing the application of UVVM to simulate, verify and test the Position sub-module

```
position_inputs_s <= in_position_1;
await_value(output_position_s, POSITION_1,
            0 * C_CLK_PERIOD, 10 * C_CLK_PERIOD,
```

```
            WARNING, "Waiting for position update");
wait for (15 * C_CLK_PERIOD);
check_value(output_position_s, POSITION_1,
            WARNING, "Check output is position 1");
check_stable(output_position_s,
            10 * C_CLK_PERIOD,
            WARNING, "Checking output is stable");
```

A.8 Motor Control Module OSVVM State-Transitionin Code

Verification code example showing the mapping of state transitions for automatic OSVVM simulation coverage

```
test_FSM_CP.AddCross("Transition from Wait State to Thyristor Startup",
                     GenBin(Motor_Control_State'POS(WAIT_STATE)),
                     GenBin(Motor_Control_State'POS(THYRISTOR_STARTUP)));
test_FSM_CP.AddCross("Transition from Thyristor Startup to PWM Startup",
                     GenBin(Motor_Control_State'POS(THYRISTOR_STARTUP)),
                     GenBin(Motor_Control_State'POS(PWM_STARTUP)));
test_FSM_CP.AddCross("Transition from PWM Startup to PWM Increase",
                     GenBin(Motor_Control_State'POS(PWM_STARTUP)),
                     GenBin(Motor_Control_State'POS(PWM_INCREASE)));
test_FSM_CP.AddCross("Transition from PWM Startup to Brake",
                     GenBin(Motor_Control_State'POS(PWM_STARTUP)),
                     GenBin(Motor_Control_State'POS(BRAKE)));
test_FSM_CP.AddCross("Transition from PWM Increase to PWM Regulation",
                     GenBin(Motor_Control_State'POS(PWM_INCREASE)),
                     GenBin(Motor_Control_State'POS(PWM_REGULATION)));
test_FSM_CP.AddCross("Transition from PWM Increase to Brake",
                     GenBin(Motor_Control_State'POS(PWM_INCREASE)),
                     GenBin(Motor_Control_State'POS(BRAKE)));
test_FSM_CP.AddCross("Transition from PWM Regulation to Brake",
                     GenBin(Motor_Control_State'POS(PWM_REGULATION)),
                     GenBin(Motor_Control_State'POS(BRAKE)));
test_FSM_CP.AddCross("Transition from Brake to No Command",
                     GenBin(Motor_Control_State'POS(BRAKE)),
                     GenBin(Motor_Control_State'POS(NO_CMD)));
test_FSM_CP.AddCross("Transition from No Command to Wait State",
                     GenBin(Motor_Control_State'POS(NO_CMD)),
                     GenBin(Motor_Control_State'POS(WAIT_STATE)));
```