

Firmware and Functional Test Platform Developed for a Smart Controller

by

Nicolaas Hendrik Naudé



*Thesis presented in partial fulfilment of the requirements for
the degree of Master of Engineering (Electronic) in the
Faculty of Engineering at Stellenbosch University*

Supervisor: Prof. M.J. Booysen

Co-supervisor: Mnr. A. Barnard

December 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2017

Copyright © 2017 Stellenbosch University
All rights reserved.

Abstract

Firmware and Functional Test Platform Developed for a Smart Controller

N.H. Naude

*Department of Electrical and Electronical Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.*

Thesis: MEng (E & E)

December 2017

Natural resources are important for human existence. Initiatives to manage resources effectively are exercised daily. The consumption of water and electricity increases daily in South Africa, and worldwide, thus the need for exploring new resource savings techniques. The primary electricity supplier of South Africa, Eskom, can not meet the current demand at all times. Not only is South Africa facing a shortage of electricity supply, but, at the time of writing, also a drought that could harm the economy. The Western Cape and the Eastern Cape provinces are especially under pressure by the drought. The Western Cape government implemented water usage limits and it is currently escalated to 87 L per person per day. The Nelson Mandela Bay municipality, in the Eastern Cape, was on the verge of being declared a drought disaster area in March of 2017. The necessity of saving initiatives are thus evident for South Africa.

The Internet of Things is well suited to contribute to these savings initiatives. This thesis forms part of a smart controller (SC) for electric water heaters (EWHs), which allows the user to monitor water usage and set a control schedule to automatically switch the EWH on and off. The SC gathers data from EWHs, allowing research to predict optimal heating schedules. This research can also be used to implement a scheduling technique to switch an EWH on and off, depending on the national electricity grid load during peak consumption times, whilst still providing the EWH user with hot water on demand.

The first development in this thesis is focused on designing and implementing firmware for a new SC hardware design. The SC communicates to a central database, with the use of an equipped cellular modem. The firmware consists of two parts, modem firmware and peripheral firmware. The peripheral firmware is responsible for correct actuator function and measuring the sensors accurately. The measurements are aggregated and concatenated into a single report string, which is sent to a cloud based database every minute.

The SC forms part of a smart electric water heater controller project, which received funding from the Water Research Council to develop and install SCs in eMkhondo municipality district in Mpumalanga, South Africa. The SC used for research purposes is

upgraded with new hardware, containing a new processor, which lead to the requirement of new firmware. The new hardware was tested in-house by a labourer, which required technical skills. This test required physical signal injection and result evaluation by the tester. The need to improve this test procedure lead to the second development of this thesis. An automatic test procedure is designed, which consists of test hardware and test software. The implementation of the complete test system is evaluated and the system efficacy is determined.

The research objective to develop and implement firmware for the new SC hardware is achieved and is implemented on a total of 245 SCs. The data collected, by these SCs, was of such a standard that research could be done on optimisation of heating schedules and provide a means to create awareness of a household's EWH consumption patterns. The second objective to develop and implement a test system was achieved, where the accuracy of the hardware is determined and the test system efficacy showed, during the validation tests, six of the ten tests were successful. The test system would be a benefit to small scale production sectors, where uncertified test equipment suffice and cost effective test solutions are required.

Uittreksel

Harde Programmatuur en Funksionele Toetsplatform Ontwikkel vir 'n Slim Beheerder

(“*Firmware and Functional Test Platform Developed for a Smart Controller*”)

N.H. Naude

*Departement Elektries en Elektronies Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.*

Tesis: MIng (E & E)

Desember 2017

Natuurlike hulpbronne is noodsaaklik vir menslike bestaan. Initiatiewe om hulpbronne effektief te bestuur, word daagliks beoefen. Die verbruik van water en elektrisiteit neem daagliks toe in Suid-Afrika, asook in die wêreld, dus word nuwe maniere ondersoek om besparings moontlik te maak. Suid-Afrika se primêre elektrisiteitsverskaffer, Eskom, se toevoer kapasiteit is onvoldoende vir die verbruikers se aanvraag tot elektrisiteit. Nie net word Suid-Afrika in nood gestel met 'n tekort aan elektrisiteitskapasiteit nie, maar is, tydens finalisering van hierdie manuskrip, in 'n droogtekrisis. Die Wes-Kaap en Oos-Kaap provinsies word veral onder druk geplaas weens die droogte, waar die Wes-Kaap limiete op water gebruik in plek gestel het om waterverbruik te tem, die verbruikslimiet is tans 87 L water per persoon per dag. Die Nelson Mandela Munisipale Metropool, in die Oos-Kaap, was op die drempel van 'n natuur ramp area verklaar, in Maart 2017. Dus is besparings initiatiewe tans noodsaaklik in Suid-Afrika.

Die internet van dinge, *Internet of Things*, is ideaal gepas om by te dra tot besparings initiatiewe. Dié manuskrip vorm deel van 'n slim beheerder (SB) vir elektriese warmwatersilinders (EWs), wat die verbruiker in staat stel om water verbruik te monitor asook 'n skedule te stel om die EW af en aan te skakel. Die SB versamel data van EWs, wat navorsing toelaat om voorspellings te maak vir optimale verhitting skedules. Dié navorsing kan gebruik word om die nasionale elektrisiteitsnetwerk se las te verlig deur gebruik te maak van tyddeel aanvraagsbestuur (TDAB). Hierdie las vermindering tegniek skakel die verbruiker se EW volgens die elektrisiteitsnetwerk se las af en aan. Dit verseker ook dat beide die netwerk en verbruiker positief beïnvloed word deur die netwerk las te verminder gedurende piek verbruikstye asook om te verseker die verbruiker het warmwater tydens aanvraag.

Die eerste ontwikkeling in die manuskrip is gefokus om die SB se hardeware gebaseerde sagteware (harde programmatuur) te ontwikkel en implementeer vir 'n nuwe SB hardeware ontwerp. Die SB kommunikeer na 'n sentrale databasis met gebruik van 'n sellulêre modem. Die harde programmatuur word gevorm deur perifere en modem harde programmatuur. Die perifere harde programmatuur is verantwoordelik om die SB

se aktueerders en sensore te beheer en akurate lesings te neem. Die gemete lesings word saamgestel in 'n enkele boodskap wat per minuut gekommunikeer word na die internet gebaseerde databasis.

Die SB vorm deel van 'n slim elektriese warmwatersilinder beheerder projek, wat befondsing bekom het vanaf die Water Navorsingsraad om SBs te ontwikkel en installeer in eMkhondo munisipaliteits distrik in Mpumalanga, Suid-Afrika. Die SB wat gebruik was vir navorsingsdoeleindes, is opgradeer met nuwe hardeware. Die nuwe hardeware bevat 'n nuwe beheerder wat die vraag vir harde programmatuur laat ontstaan het. Die nuwe hardeware word binnenshuis (inhuis) getoets deur personeel wat tegniese aangelê is. Hierdie toetse benodig fisiese sein generasie en resultaat evaluering deur die toetsers. Die tweede ontwikkeling se nut het ontstaan om hierdie toetsprosedure te verbeter. 'n Outomatiese toetsprosedure is ontwikkel wat bestaan uit toets hardeware en toets sagteware. Die implementering van die volledige toetsstelsel word evalueer om die akkuraatheid van die berekenings-effektiwiteit van die toetsstelsel as geheel te bevestig.

Die navorsingsdoelstelling om nuwe harde programmatuur te ontwikkel en implementeer was bereik en is op 'n totaal van 245 SBs implementeer. Die data, versamel deur die SBs, se gehalte is so van aard dat dit gebruik kan word vir navorsingsdoeleindes. Die tweede navorsingsdoelstelling, om 'n toetsstelsel te ontwikkel en implementeer, was bereik. Die akkuraatheid van die hardeware is bepaal en die berekenings-effektiwiteit van die toetsstelsel het getoon dat ses uit die 10 toetse suksesvol was, tydens die evaluasie toets. Die toetsstelsel kan effektief gebruik word in kleinskaalse produksie van produkte waar die toets voldoende sal wees met koste effektiewe, ongesertifiseerde toetsgereedskap.

Acknowledgements

My sincere appreciation to each and everyone contributing to this research thesis and making use of the work done. I specifically would like to give thanks to the following people for guidance and support:

- Prof. Thinus Booysen for giving me the opportunity to apply my passion for development into research. Also, thank you so much for being a blessing to provide the necessary financial support to complete this research.
- Mr. Arno Barnard for co-supervising me and thank you for challenging me to reach new frontiers.
- My girlfriend, Alexia Maritz, for sticking with me when I am preoccupied with work and providing your excellent editing support. I love you, and forever will.
- My family, thank you for your continual support even when you do not always know what I am doing, thank you for being there, always.
- My co-Mobile Intelligence Lab workers, thank you for the numerous cups of coffee and technical support.
- Thank you MTN for the Mobile Intelligence Lab.
- Thank you to Bridgiot for allowing me to co-develop and work with them.

Contents

Declaration	i
Abstract	ii
Uittreksel	iv
Acknowledgements	vi
Contents	vii
List of Figures	x
List of Tables	xii
Listings	xiii
1 Introduction	1
1.1 Internet of Things Utility Saving Application	1
1.2 Smart Electric Water Heater Controller Project Foundation	2
1.3 Problem Statements	2
1.4 Proposed Solutions	3
1.5 Research Objectives	4
1.6 Contributions	4
1.7 Thesis Structure	4
2 Literature Review	6
2.1 Overview	6
2.2 Smart Electric Water Heater Controller - SC	6
2.2.1 SC Project State	7
2.2.2 Hardware Analysis	8
2.2.3 Legacy Firmware Breakdown	10
2.2.4 Hardware Production Process	10
2.3 Existing Smart Controller Test Procedure	11
2.3.1 Pre-existing Test Procedure	11
2.3.2 Test Procedure Shortcomings	11
2.4 Functional Test System Methodologies	13
2.4.1 Quality Control (QC)	13
2.4.2 Functional Test Procedures	15
2.4.3 Test Philosophy	16

3	Smart Controller Firmware Design	17
3.1	Overview	17
3.2	System Requirements and Specifications	17
3.2.1	Updated System Requirements	17
3.2.2	Updated System Specifications	19
3.3	Firmware Design	20
3.3.1	Firmware Architecture	21
3.3.2	Smart Controller Control Function	22
3.3.3	Temperature Readings	22
3.3.4	Power Measurement	23
3.3.5	Water Flow Measurement	24
3.3.6	Leak Detection	25
3.3.7	Shut Off Valve	25
3.3.8	Latching Relay	26
3.4	Application of Firmware	27
4	Test System	28
4.1	Introduction	28
4.2	Test Procedure Design	29
4.3	Proposed Test System	30
4.4	Test Bench Hardware	33
4.4.1	Proposed Test Bench	33
4.4.2	Test Bench MK 1 - Design and Implementation	34
4.4.3	Test Bench MK 2 - Design and Implementation	38
4.5	Test Manager Software	46
4.5.1	Software Design	46
4.6	Test System Summary	50
5	Experiment and Results	52
5.1	Overview	52
5.2	Smart Controller Firmware	52
5.2.1	Firmware Efficacy	52
5.2.2	Firmware Validation	56
5.2.3	Findings	57
5.3	Test System	57
5.3.1	Test Bench Hardware Efficacy	57
5.3.2	Test Manager Software	61
5.3.3	System Application	67
5.3.4	Validation	68
6	Conclusion	72
6.1	Overview	72
6.2	Evaluation	72
6.2.1	Smart Controller Firmware	72
6.2.2	Test System	72
6.3	Recommendation	73
6.3.1	Smart Controller Firmware	73
6.3.2	Test System	73
6.4	Concluding Remarks	74

<i>CONTENTS</i>	ix
Appendices	75
A Smart Controller Firmware	76
A.1 Firmware Function Diagrams	76
B Test Bench MK 1 and 2 Hardware	79
B.1 Schematic Test Bench MK 1	80
B.2 Schematic Test Bench MK 2	82
B.3 MK 2 Firmware Functions	84
C Test System Software	87
C.1 Test Manager Software Class Diagrams	87
C.2 Test System Handler UML	89
D Results	92
D.1 System Validation Configuration File Setup	92
D.2 Generated PDF Report	95
Bibliography	98

List of Figures

2.1	Smart Controller Sensor and Control Setup on an Electric Water Heater, Adapted from [15].	7
2.2	Risk Mitigation Hardware	8
	(a) Water Shut Off Valve	8
	(b) Leakage Detection Wires	8
2.3	Reed Switch Bounce.	9
2.4	Water Measurement Hardware	10
	(a) Water Flow Meter	10
	(b) Reed Switch Used in the Water Meter	10
2.5	Pre-Existing Test Procedure	12
2.6	Procurement Process Flow for Typical Electronic Products, where P is Pass and F is Fail, Adapted from [44], [45] and [43]	14
3.1	Architecture Summary of the Smart Controller Firmware.	21
3.2	Generic Main Loop of the Smart Controller Firmware.	21
3.3	Controller Function of the Smart Controller Firmware.	23
3.4	Latching Shut Off Valve State Machine used in the Smart Controller Firmware.	26
3.5	Latching Relay State Machine used in the Smart Controller Firmware.	27
4.1	Test System Overview Diagram	28
4.2	Final Test Procedure with Test Software Process Flow	30
4.3	Test Bench Black Box	34
4.4	Test Bench Hardware MK 1	35
4.5	Valve Sense Circuit with Schmitt Trigger from Test Bench MK 1	36
4.6	Leakage Emulation Circuit Diagrams	37
	(a) Designed MK 1 Leakage Emulation Circuit	37
	(b) Leakage Detection Circuit of the Smart Controller	37
4.7	Developed Test Bench MK 1	37
	(a) Top	37
	(b) Bottom	37
4.8	Test Bench Hardware MK 2	39
4.9	Improved Valve Sense Circuit of Test Bench MK 2	41
4.10	Test Bench MK 2 Main Firmware Loop	42
4.11	Developed Test Bench MK 2	45
	(a) Top	45
	(b) Bottom	45
4.12	Test Manager Software Integration In The Test System	46
4.13	Test Software Process Flow	46
4.14	Test System Application	48

4.15	Test Step Handler State Machine	48
4.16	Report Creation Flow Diagram	49
4.17	Micro Controller Application Setup	50
4.18	Modem Setup Flow Diagram	51
5.1	Temperature Measurements Read from a Siglent SDG 1005 Signal Generator, Incrementing in Steps of 10 mV, and the Smart Controller Hardware MK 3 Firmware.	53
5.2	Current Measurements Read from a TopTronic T98T TrueRMS Clampmeter and the Smart Controller Hardware MK 3 Firmware.	54
5.3	Captured Figure from the Web Application of Bridgiot[12]. Total Water Consumed in the Week of Testing. Measured with the Smart Controller Hardware MK 3 Water Measurement Firmware.	54
5.4	Shut Off Valve Control of the Smart Controller Hardware MK 3 Firmware. . .	55
	(a) Valve Latch Closed	55
	(b) Valve Latch Opened	55
5.5	Leak Detection Triggered on the Smart Controller Hardware MK 3 Firmware with a Pulse Waveform.	56
5.6	Frequency Measurements of the Water Flow Pulser.	58
5.7	Water Flow Pulser Producing Two Pulses per Frequency in the Range, One to Five Hertz.	58
5.8	Voltage Measurements from Test Bench MK 2 Digital to Analog Sensor. . . .	59
5.9	Leakage Emulation Triggering Shut Off Valve.	59
5.10	Valve Sense Logic Output from Valve Closing	60
5.11	Serial Output from Test Bench Buttons	60
5.12	Described Test System Setup with Test Bench MK 2 and the Built Test Rig. .	67
5.13	Pogo Pins Used in the Built Test Rig.	67
A.1	Temperature Readings Diagram.	76
A.2	Water Flow Measurement Flow Diagram.	76
A.3	Leakage Detection Flow Diagram.	77
A.4	Power Measurements Service Function Flow Diagram.	77
A.5	Power Measurements RMS Calculation Function Flow Diagram.	78
B.1	Test Bench MK 1 Schematic Diagram.	81
B.2	Test Bench MK 2 Schematic Diagram.	83
B.3	Valve Sense Service Flow Diagram.	84
B.4	Pulse Service Flow Diagram.	84
B.5	Drip Service Flow Diagram.	85
B.6	Control Service Flow Diagram.	85
B.7	Communication Service Parser Flow Diagram.	86
C.1	Test Manager Class Diagrams.	88
C.2	Test System Handler UML Part 1.	90
C.3	Test System Handler Part 2.	91

List of Tables

2.1	Smart Controller Features	8
2.2	Pre-Existing Test Procedure Requirements	11
2.4	In Process Quality Control (IPQC) Test Procedure Comparison, Extracted from [45]	15
3.1	Updated System Requirements, Adapted and Modified from [24]	17
3.1	Updated System Requirements, Adapted and Modified from [24]	18
3.1	Updated System Requirements, Adapted and Modified from [24]	19
3.2	Updated System Specifications, Adapted and Modified from [24]	19
3.2	Updated System Specifications, Adapted and Modified from [24]	20
4.1	Automation Features Required to Achieve Test Procedure Requirements	29
4.2	Compared Single-Board Computer Platforms	32
4.3	Test Bench Requirements and Specifications	34
4.4	Test Bench Hardware Features: MK 1 <i>vs</i> MK 2	39
4.6	Firmware Communication Protocol	43
5.1	Power Measurement Differences Depicting the Over Estimation Factor of the Smart Controller. The Measurements are Between the Smart Controller Hardware MK 3 Firmware and a TopTronic T98T TrueRMS Clampmeter.	53
5.2	A Constant of 10 Litres of Water is Emulated at Different Water Flow Rates and Measured by the Smart Controller Hardware MK 3 Firmware.	55
5.3	The Percentage Difference of the Voltage Dividers used in the Test Bench MK 2.	58
5.4	Definition of the Configuration File, <i>FUNCTION_LIST</i> Section, Functions.	63
5.6	The Default Values Defined in the Configuration File, <i>DEFAULT_JSON</i> Section.	63
5.7	Configuration File, <i>DEFAULT_COMPARE_SETTINGS</i> Section, Used to Bind Received JSON Keys to Functions in the <i>FUNCTION_LIST</i> Section for Comparing Results.	64
5.8	Validation Test Procedure Summary.	69
5.9	Test System Validation with 10 Smart Controllers Tested with the Test Procedure in Appendix D.1.	71

Listings

5.1	Snippet of an Example Test Procedure Configuration File.	65
5.2	Snippet of the Finite State Machine Implemented in a Dictionary of Methods - 'State Methods'	66

Chapter 1

Introduction

South Africa is facing a serious scarcity of resources, motivating water and electricity saving initiatives [1, 2]. The severity of the drought in the Western Cape is forcing authorities to restrict residents to 87 litres of water usage per person, per day [3]. The Nelson Mandela Bay municipality was on the verge of being declared a drought disaster area in March of 2017 [4]. At the time of writing, the capacity of the Eastern Cape dam levels had not been 100% since 2015 [5].

In addition to the water crisis, a national electricity shortage is also present. The electricity demand exceeds the generation capacity, causing the main electricity supplier, Eskom, to reduce the grid load by implementing load shedding, which are set periods of power cuts [6]. Preventative initiatives to reduce load shedding periods, such as "Power Alert" and "Geyser Control", are driven by Eskom and local municipalities [7, 8, 9, 10, 11].

A Smart Controller (SC) for an Electric Water Heater (EWH) was designed to provide insight on the water and electricity consumption of EWHs, where the user has access to water usage patterns and control over the EWH's heating schedule [12, 13]. A study performed on a coffee shop showed a water usage reduction of 67% after the consumption data was revealed to the owner, which proves that behavioural insight could save [14]. Compared to the "Geyser Control" initiative, the SC can similarly be used to reduce the grid load. Instead of pre-defined off-times, the usage patterns of each user can be used to intelligently determine switch-off times of multiple EWHs with the use of Time Division on Demand Side Management, a peak-shaving and valley-filling technique, and still provide the consumer with on-demand hot water [15].

1.1 Internet of Things Utility Saving Application

The Internet of Things (IoT) is defined as "a network of everyday devices, appliances, and other objects equipped with computer chips and sensors that can collect and transmit data through the Internet" [16]. It provides a means to connect everyday household devices to the cloud. One of the first applications of IoT were smart utility meters, used to connect energy consumers to a smart energy grid, which effectively measures and manages energy flow of the consumers [17]. The IoT sphere is vast and capable of countless applications of which the SC for EWHs is one [18]. This application incentivises conscious water use and provides insight to how the heating schedule of the EWH is directly proportional to the hot water usage pattern [14, 19].

The IoT spectrum of applications does not only rely on the infrastructure to connect an object to the cloud, but data storage and management are also key factors. The

aggregated data from IoT applications can potentially be used in big data analysis, where trends, patterns and human behavioural predictions are analysed.

The data gathered from EWHs was analysed and used to validate an EWH simulation model, allowing algorithms to determine an optimal heating schedule according to previous usage events [15].

1.2 Smart Electric Water Heater Controller Project Foundation

This section provides an overview of where the SC project originated from and provides the background to necessitate the choices made and the limitations reached in this thesis.

The origin of the SC for an EWH was designed and developed by an MTN-funded Mobile Intelligence Lab in the Electrical and Electronic Engineering Department of Stellenbosch University [20, 21].

The project gained traction when funding was received from the Water Research Council (WRC) to run a pilot study in the eMkhondo municipality district in Mpumalanga, which mostly contributed to the total 245 SCs installed [22]. The project transitioned from a research project, where only data collection and analysis was done, into a company, named Bridgiot. The company is spun out with Innovus, an industry interaction agent of Stellenbosch University, who assists with investment sourcing and entrepreneurial support [23]. The previously researched data analytics were used to create a promotable user interface to manage an EWH from a web-based application [24, 12].

The WRC funding was used to develop the SC design into a commercial design and gain the required certification according to the South African National Standard (SANS) [25]. The certification was sourced to the South African Bureau of Standards (SABS) certification body, where the SC design was approved for installation onto an EWH.

The commercial design of the SC, namely MK 3, consisted of a completely new hardware layout, designed by Brown as part of his thesis project [24]. The new design required a software program which provides the hardware with all its functionality features, known as firmware. The SC, being an IoT device, requires a maintained internet connection to the cloud while managing and executing control over the EWH. The firmware was divided into two development threads, communication firmware and peripheral firmware. The communication firmware was developed by Cloete as part of his thesis project [26]. The peripheral firmware development is discussed in this thesis project.

The commercial SC, namely MK 3, was rolled out to the eMkhondo municipality district for installation, but only after a manual functionality test was performed on each SC. This ensured that any faulty components were detected in the production process, minimising field repairs or replacements.

1.3 Problem Statements

The rapidly growing IoT sphere creates opportunities to potentially relieve strenuous utility resources with its monitoring and control capabilities. Data capturing IoT devices require accurate sensor measurements and responsive controls to provide valid and usable data collections. The commercialised SC requires firmware to function as an IoT application on EWHs. The SC provides the opportunity to capture EWH temperatures,

measure water and electricity usage; additionally, the SC can implement control on the heating element and the water input [12].

The IoT sphere not only creates opportunities for new applications, but also introduces new test strategies for IoT devices. These strategies consist of standard hardware assembly tests, functional hardware tests and functional system tests allowing the entire system to be tested, from the hardware to the cloud. A traditional way of testing IoT applications is to introduce actual user and environmental data to the test environment to ensure the test data is of a sufficient quality. The quality and reliability of the firmware is validated by introducing simulations to the embedded firmware which emulates typical use case events. The obtained simulation results are compared to the expected outcome indicating whether or not the firmware is of an acceptable quality [27]. The quality assurance on the hardware is still a high priority, because upgrading or exchanging field installed hardware is more cumbersome than issuing a remote firmware update. Firmware updates are issued with the use of Over-The-Air (OTA) programming, readily available in IoT applications [28].

Existing certified and market ready test equipment comes with a hefty price tag starting at $\approx 30,000$ ZAR [29]. A hardware based start-up company either invests capital into hardware or employees. Since Bridgiot is a subsidiary of Stellenbosch University, the tools for building and prototyping hardware is mostly available in-house, therefore capital is rather invested into people than expensive test systems [30]. The need for expensive test systems is also determined by the severity of Failure Mode, Effects, and Criticality Analysis (FMECA). If the product does not threaten an immediate life during failure it does not need certified precision tests [31]. A failure event for the SC can result in producing cold water to the end-user. The test equipment and strategies used are functionality focused to ensure a working product is shipped to the consumer. The test strategy being used consists of manual labour, requiring a low to mid level technically skilled tester. Due to limited available funds, manual tests were devised to test all the functionalities of the SC prior to shipping to eMkhondo, which is time consuming and resulted in a bottleneck. A solution for a fully automated functionality test is required.

1.4 Proposed Solutions

This thesis paper proposes solutions for the identified problems discussed in Section 1.3. Firstly, a solution to the peripheral firmware of the SC is proposed. The research based SC, MK 2, has firmware available, but it is not compatible with the MK 3 SC. Thus, a solution to the incompatibility is to investigate the MK 2 firmware and use the available MK 3 functionality to implement compatible new firmware to the MK 3 hardware, fulfilling the need for SC peripheral firmware to monitor and control an EWH.

Secondly, a proposal to test the MK 3 automatically is developed. The main goal of the proposed test system is to automate the existing manual test procedure. This solution aims to produce a portable and/or easy to setup test platform, by making use of readily available technologies to measure, actuate and emulate the required signals or peripherals. This will effectively emulate an EWH to which a SC can connect. Resultant data will be stored locally and be presentable in a human-readable format.

1.5 Research Objectives

The research objectives for this paper are as follows:

Objective 1 Investigate and design functional peripheral firmware for the new SC MK 3 hardware design.

Objective 2 Design a test system capable of executing a test procedure automatically.

2.1 Identify and assess possible test methods applicable to the SC.

2.2 Design test procedures capable of performing the identified test method/s in Objective 2.1 on the SC.

2.3 Design a cost effective test system for small scale test procedures.

Objective 3 Qualify the designed test system in Objective 2.

Objective 4 Recommend considerations for future contributions towards the designed test system.

1.6 Contributions

The work done in this thesis proved to contribute to the SC for EWHs project by implementing a new firmware design for the MK3 hardware designed by Brown in [24]. The WRC funding resulted in 245 SCs being installed, mostly in eMkhondo municipality district, and EWH data being gathered for future research.

The presented firmware may prove as an example of how to implement peripheral functionality on the specific micro-controller (MCU), ATXmega128A4U from Microchip, previously Atmel [32, 33].

This thesis presents a test system consisting of a hardware test platform and complementing test software designed for testing the MK 3 SC. The test software provides user configurable test procedures, which allows for various test applications within the hardware platform's specifications.

1.7 Thesis Structure

Chapter 2 presents literature on the SC for EWHs project of which the need for the development of new firmware and a test system arised. The project state is explained with an in-depth hardware breakdown. The available legacy firmware is inspected along with the existing hardware production procedure. The existing hardware test procedure is critiqued and an improved test system is proposed. Quality control in hardware production is investigated for applicable test procedures for the project requirements. Finally, the test philosophy used in the proposed test system is defined.

Chapter 3 presents the modified SC system requirements and specifications to which the SC is designed. New firmware for the latest SC hardware design is described along with the application success of the deployed firmware.

Chapter 4 presents the design of proposed test procedure, which requires hardware and managing software. The test system is proposed and the system platform is discussed. The test bench hardware is proposed and went through two iterations of design, producing test bench MK 1 and MK 2. The limitations of the MK 1 design is highlighted. The test manager software is proposed and the design, in light of executing the proposed test procedure, is presented in detail.

Chapter 5 presents the results of testing the firmware design and the test system design. The firmware efficacy is tested with defined experimental setups and the firmware reliability and data validity is validated. The test manager software is implemented and presented in detail. A system application of the test bench hardware along with the test manager software is implemented and validated.

Chapter 6 concludes the work by discussing the findings for each of the designs and implementations. Each of the objectives stipulated in Section 1.5 are assessed and recommendations for future contributions are presented.

Chapter 2

Literature Review

2.1 Overview

The Smart Controller (SC), as depicted in Figure 2.1, for Electric Water Heaters (EWHs) project evolved from various SC related contributions of which Booysen *et al* designed a proof of concept for monitoring and controlling EWHs in a smart grid [34]. Nel implemented an EWH model and a method of detecting hot water usage events [35]. Brown designed SC hardware allowing remote controlling and monitoring of an EWH [24]. He was also responsible for the initial field tests which provided insight with regards to the design and installation of the SC. This led to a new hardware design, namely MK 3. Cloete contributed to the communication layers, allowing users to be connected to their EWH and implement custom heating schedules [26]. As one can see, the research project underhand is an on-going, multi-annual project, based on a SC design.

The project received funding from the Water Research Council (WRC), of which a set amount of SCs needed to be installed in eMkhondo in conjunction with Mkhondo local municipality, situated in Mpumalanga, South Africa [22]. Due to the funding requirements, it was necessary for the MK 3 hardware controller to be functional. This led to objective 1.5 in Chapter 1. The sheer amount of devices being produced required a test platform to detect and rectify production defects. Since the quantity of devices being produced is not large enough for in-line production tests, an alternative solution was required. The need for testing the SCs led to Objective 2.

The research conducted in this project was twofold. Firstly, the required functional firmware for the MK 3 SC had to be completed. Therefore an in-depth investigation was done on the SC. Secondly, production quality control methods were inspected to ensure the SC is tested against the correct and relevant specifications with the most applicable test methods.

This chapter will firstly delve into the specifics of the SC design to provide sufficient background concerning the SC hardware functionality for the development of the firmware for MK 3. Secondly, the different functional test methods are investigated and compared to the functional test requirements ensuring an acceptable level of quality assurance is provided to the consumer.

2.2 Smart Electric Water Heater Controller - SC

The SC was dubbed SC during the roll-out phase of the first installations in eMkhondo. The origin of SC was from the play of words, for making your geyser, South African word

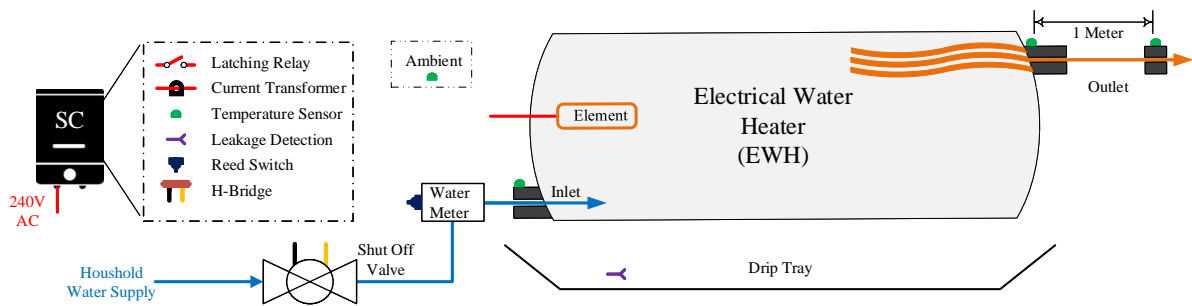


Figure 2.1: Smart Controller Sensor and Control Setup on an Electric Water Heater, Adapted from [15].

for EWH, easy to control [12]. The SC device originated from prior work done by Brown and Booysen [36, 24]. This demand side management solution provides users with the ease and luxury of implementing a water heating schedule remotely. The SC allows users to switch their EWH on and off from a web-based application. Moreover, the SC provides smart water usage data combined with the event duration, the water temperature of the event and with a designed algorithm, the cost of an usage event, combined water and electricity usage, is estimated. The SC also provides risk mitigation with burst detection and a water shut off valve to reduce excess water from incurring additional damages.

2.2.1 SC Project State

The SC project progressed from a working conceptual design, known as MK 2, to a commercial design, which is MK 3. The newly designed MK 3 only consisted of a new hardware design. The hardware design formed part of Brown's research project [24].

In the design of MK 3 the following features are available:

- 4 Temperature Measurements (Inlet, Outlet Near, Outlet Far and Ambient)
- Set Point Control
- Energy Measurement
- Water Measurement
- Water Shut off Control
- Minutely Event Reporting

The SC being an IoT-device consists of two parts, namely communications and peripherals.

Communications The communication side was designed and implemented by Cloete [26]. The protocol used for the communication to the back-end systems are done with Message Queuing Telemetry Transport (MQTT) where the status update report messages are published to a server side MQTT broker [37]. The data is processed via a data translation layer to a Mongo database [38]. The database is where the data is stored, it is accessible by a data processing application to provide the user with the necessary details to manage and control their EWHs.

Since Cloete was responsible for the hardware communications part, the research scope is focused on the peripherals part of the hardware.

Table 2.1: Smart Controller Features

Feature	Hardware	Type
Temperature	Analog Temperature Sensor	Input
Set Point control	Latching Relay	Output Actuator
Energy Measurement	Current Transformer	Input
Shut Off Valve	H-Bridge	Output Actuator
Leakage Detection	Short Circuit Wires	Input
Water Flow Meter	Reed Switch	Input
On-Board Sensor Management	Micro-Controller	Input and Output

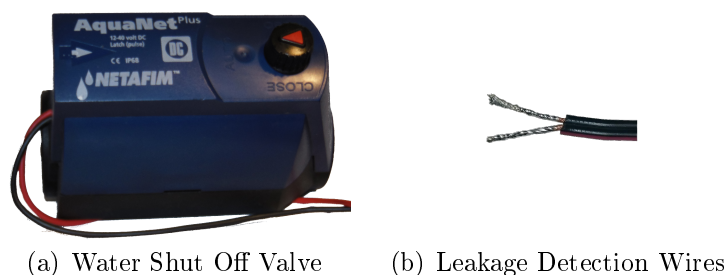
Peripherals The peripheral side consists of various input and output hardware. Each circuit and its corresponding feature is shown in Table 2.1 The peripherals were designed and chosen by Brown in [24], leaving the firmware for future design. The firmware from MK 2 is not compatible with the MK 3 design, therefore a hardware analysis is required. The hardware in Table 2.1 will be inspected and analysed in the next subsection.

2.2.2 Hardware Analysis

Each of the hardware components, listed in Table 2.1, are described in detail as follows:

Analog Temperature Sensor The temperature sensor used to measure the various temperatures around the EWH are low-power linear active thermistors, MCP9700, from Microchip which are powered from 3.3 V. These sensors have a $\pm 2^\circ\text{C}$ accuracy and provides temperature readings in ranges -40°C to $+150^\circ\text{C}$. The conversion used from analog to digital readings is $10 \text{ mV}/^\circ\text{C}$ with a 500 mV offset.

Latching Relay A latching relay, JE10-1/024-HSTL2, from RS-Pro is used to control the power supply to the EWH, allowing the SC to implement scheduling. A latching relay is used since the relay is required to maintain a state for long durations at a time. The latching functionality of the relay requires a set and reset pulse to traverse to the required state. The circuit, designed by Brown, triggers from 3.3 V and uses two transistor switches, one triggers the set pulse and the other the reset pulse. These pulses require a 20 ms pulse duration at 3.3 V to allow sufficient time for the internal coil to charge and be latched. The relay used is rated to switch a 20 A load.

**Figure 2.2:** Risk Mitigation Hardware

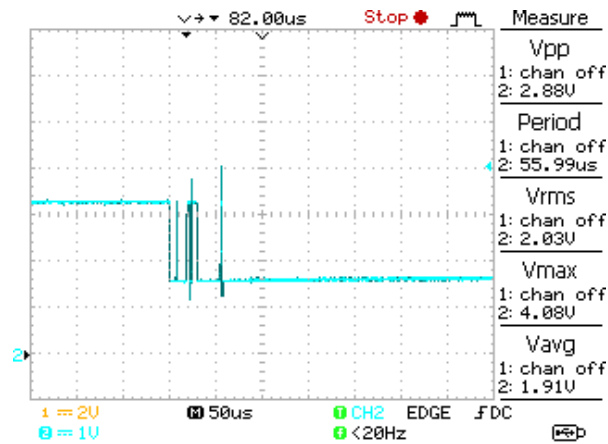


Figure 2.3: Reed Switch Bounce.

Current Transformer The current transformer, AC1005, from Talema is used to measure the amount of energy consumed by the EWH during heating. Since a thermostat is used by default in an EWH, the EWH does not necessarily draw power for the entirety of the allocated heating period when scheduling is implemented. The current transformer circuit is designed to draw a maximum load of 4 kW which equates to an approximated maximum of 17.39 A peak current drawn by the EWH element [24]. The current transformer used in the design has a ratio of 1000:1, which results in producing an output voltage between 0.8 and 2.6 V that oscillates around the reference voltage of 1.65 V, according to [24].

H-Bridge The water shut off valve responsible for risk mitigation is an AquaNet Plus from Netafim as depicted in Figure 2.2(a), is of a latching nature combined with a manual override. The valve requires 18 - 24 V pulses of 80 ms to either open or close the valve. The valve requires bi-polar switching, thus a H-bridge, TA8428K, from Toshiba is required to switch the valve pulse polarities. The H-bridge is capable of producing 3 A pulses for 100 ms and in case of continuous current draw. A maximum of 1.5 A can be drawn.

Short Circuit Wires The short circuit of wires occur when the tips of the two exposed wires, depicted in Figure 2.2(b), conduct due to being submerged in, or connected to a conducting surface or substance. In the SC design the wires will be submerged in water when a EWH leakage event occurs. The wires will be able to pick up a leakage if it is placed within the drip tray of the EWH. The stripped wires will produce a voltage difference depending on the conductivity of the substance it is submerged in, where the voltage difference can be used to determine whether a leakage event has occurred or not.

Reed Switch A reed switch, depicted in Figure 2.4(b), is a switch producing a short-circuit when a magnetic field is applied, which forces ferromagnetic strips to connect for the duration of the magnetic field presence. The water meter, depicted in Figure 2.4(a), provides an interface for a reed switch. The water meter, Elster Kent V100, produces two pulses per one litre of water measured. The water flow meter produce four state changes per one litre, which translates to: for every half litre a falling edge is produced. The reed switch is an mechanical switch, which can cause bouncing and produce false pulses as depicted in Figure 2.3.

Micro-Controller The on-board sensor management describes all the analog measurements, digital signal readings and data processing features required in the SC. The device capable of executing these features is a micro-controller unit (MCU), the MCU chosen in the SC MK 3 design is from Atmel's XMEGA-family, specifically the A4U series with 128 kB of in-system self-programmable flash [33]. The MCU operates at a frequency of 32 MHz at 3.3 V. In the MK 3 design the MCU is responsible for the co-operation of the peripheral-side firmware and the communication-side firmware. All the peripherals of the MK 3 design are measured and controlled at 3.3 V except the cellular modem, which operates at 3.8 V. The cellular modem used is the u-blox LEON G100 Quad-Band GSM/GPRS Module [39], which is used to establish a connection to a cloud server with the use of the transmission control protocol (TCP). The peripheral telemetry of the SC is aggregated by the MCU and sent via the modem to the cloud. The communication infrastructure design and implementation is described in detail by Cloete in [26].

2.2.3 Legacy Firmware Breakdown

The pre-existing firmware was implemented by Brown on a Particle MCU, initially on the discontinued Core and later on the Photon [40], as described in [24]. The firmware consisted of an IDE specific C-based language using platform specific libraries from the Particle IDE. The firmware use decremental time-outs using a hardware timer for time-keeping, allowing multiple processes to be serviced once their decremented timer reaches zero. As soon as the serviced process has completed, its time-out timer is re-set. The design follows a non-blocking process manager, allowing multiple functions to operate in synergy. The documentation of the legacy firmware is depicted in Chapter 3 of [24].

2.2.4 Hardware Production Process

The production process implements a conservative form of quality control by doing voltage checks and functionality checks, the process is as follows.

The SC is populated and assembled by a local technician, responsible for all the soldering and component wiring. Basic voltage checks are done during the assemble phase, ensuring all the voltage sources function properly. The assembled SC is sent to the production test station where a low level technician programs it and do basic functionality tests. After the tested SC has passed the functionality tests, it is put on a burn-in rig where it will communicate to the cloud for 12 hours, if no irregularities were found after burn-in the SC is packaged and marked ready for shipping. The hardware assembling

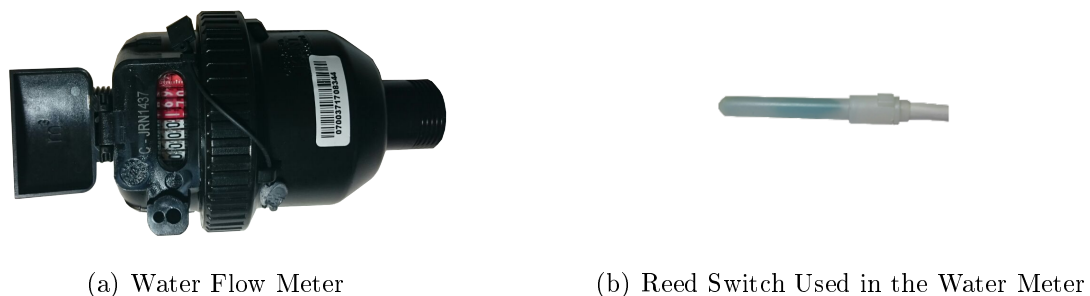


Figure 2.4: Water Measurement Hardware

Table 2.2: Pre-Existing Test Procedure Requirements

ID	Test Requirement
1	Peripherals
1.1	Measured temperatures must correspond to the room temperature when temperature sensors are connected.
1.2	Water valve latches open when SC switch on and latches closed during leakage event.
1.3	Latching relay latches on when SC switch on and latches off during leakage event.
1.4	Leakage event triggers successfully after short circuiting leak detection leads.
1.5	The power measurements reflect the load connected to the SC.
1.6	The SC operates at the correct voltage of 3.3 V.
1.7	The induced water flow pulses correlates to the expected volume of measured water flow.
2	Communication
2.1	The modem of the SC is configured successfully according to the setup script.

local technician is also responsible for repairing any hardware related faults found by the production test station or burn-in rig.

2.3 Existing Smart Controller Test Procedure

The mentioned production test station in the previous section performs a manual test procedure, depicted in Figure 2.5. This test procedure was developed and used prior to the test procedure design of this project. Therefore, the pre-existing test procedure in Figure 2.5 was analysed and used as a basis to build the new test procedure of this project.

2.3.1 Pre-existing Test Procedure

The pre-existing procedure analysis produced the minimum test requirements listed in Table 2.2, which contributed to the proposed test procedure design in Chapter 4. The pre-existing test procedure is an in-house design, aimed at production testing, performed by a low to mid level technician. This technician is responsible for connecting, switching and introducing signals to the SC as depicted in Appendix 2.5.

2.3.2 Test Procedure Shortcomings

Shortcomings were identified in the pre-existing test procedure, in Figure 2.5, which are as follows:

1. Signal injections are prone to human error.
2. No voltage measurements are taken during the test.
3. The test requires listening and observing of latching components.

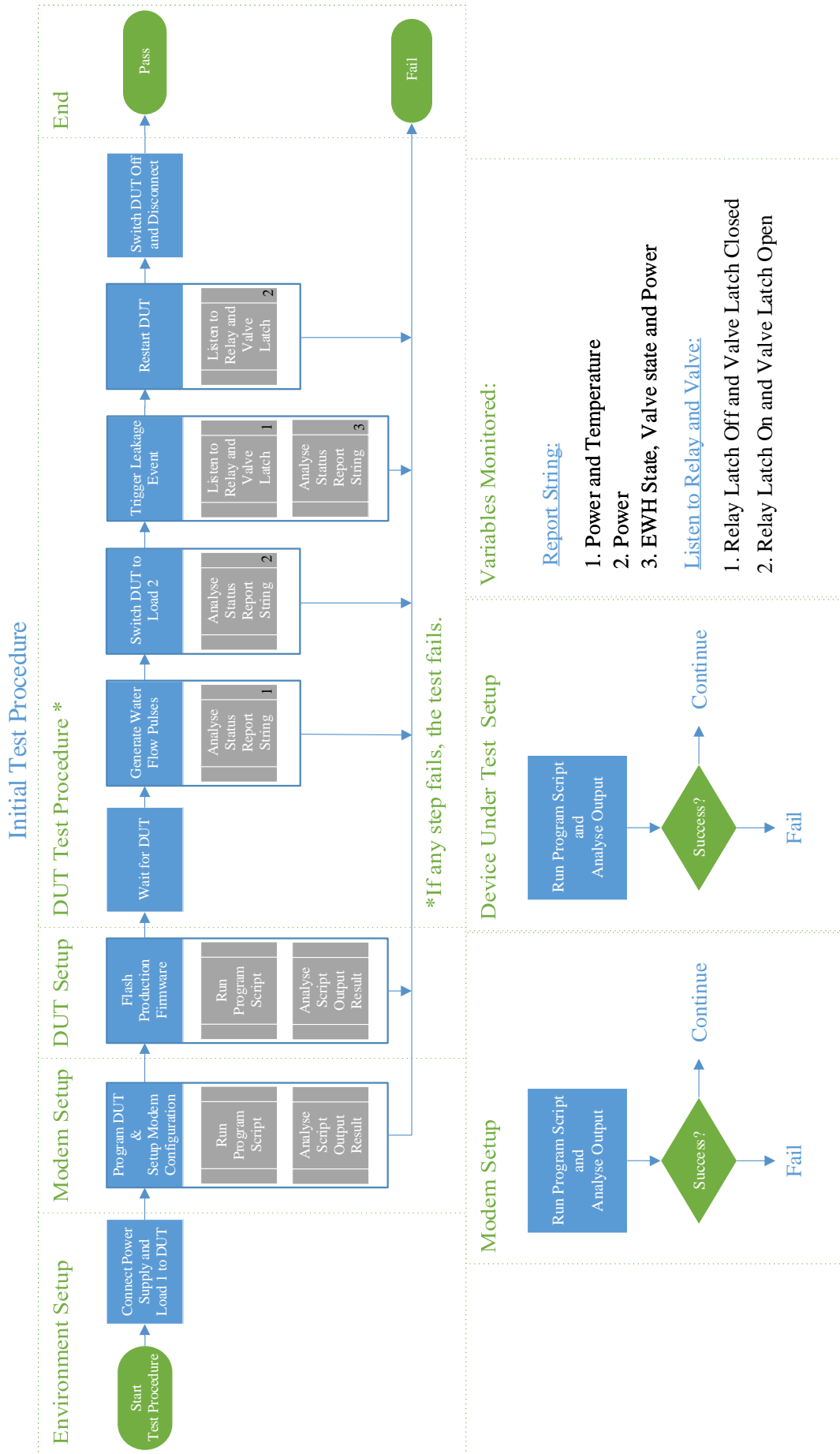


Figure 2.5: Pre-Existing Test Procedure

4. The report strings are verified from the SC hardware and not from the cloud.

The areas of focus for improvement are reliable signal injection for all the possible signals that can be emulated, the SC voltages will be measured, the valve latching will be detected without the need to listen whether the valve has latched. The reported strings from the SC will be monitored from the cloud, ensuring a closed-loop test is done. These improvements will lead to simplification of the test procedure such that a non-technically inclined person can run the test successfully, another area subject to improve is test times. The design choices are made with the goal to reduce design and production costs. Due to the nature of the SC, test procedures do not require exceptional accuracy and any national certifications other than an environmental stress test accreditation, which is outsourced to an accredited test facility.

2.4 Functional Test System Methodologies

The process from product development to product production consists largely of prototyping, prototype refining and finally ensuring the final product functions as it was intended to for the designed lifetime duration [41]. The reason for achieving Objective 2 and Objective 3 in this research project is to provide a quality assurance to the consumer by ensuring the used test methods provide confidence in the product production and assembly stages. The expected level of quality assurance can be achieved by employing proper quality control (QC) methods in the manufacturing process. The QC methods are explained and investigated in this section along with the possible test procedures. The various test procedures are inspected and considered for the test system design in Chapter 4.

2.4.1 Quality Control (QC)

Quality control is the process of inspecting a product against the product designed specifications [42]. Quality control conducted on a typical product procurement assembly line, depicted in Figure 2.6, consists of the following three sub-processes according to Wu in [43].

Incoming Quality Control (IQC) The process of sampling and inspecting of the sourced raw materials used in the product design prior to manufacturing.

In Process Quality Control (IPQC) The process of testing and inspecting of the designed sub-circuits and product features during manufacturing and assembly.

Outgoing Quality Control (OQC) Products are re-inspected visually as well as functionally before it is shipped out to the consumer.

Quality control sub-processes consist of various types of tests conducted during the procurement process of the product. The procurement process researched is depicted in Figure 2.6 which outlines the various stages of the product assembly line and QC sub-processes. The QC sub-processes are responsible for, IQC, ensuring the materials used in the product assembly comply to the material specifications. The product assembly stages commence once IQC is approved, the IPQC process consists of three assembly stages where the tests conducted in the initial assembly stage verifies the correct positioning of

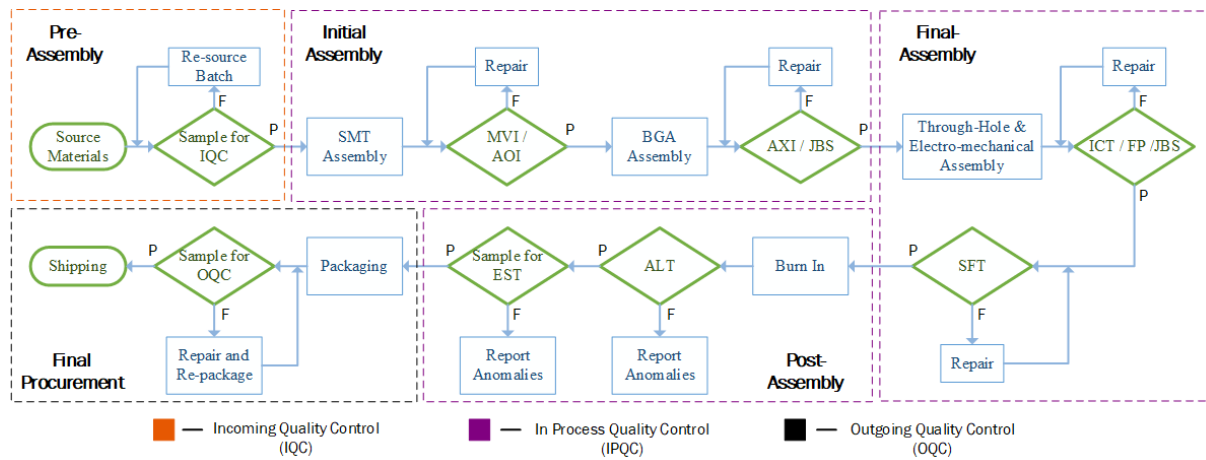


Figure 2.6: Procurement Process Flow for Typical Electronic Products, where P is Pass and F is Fail, Adapted from [44], [45] and [43]

components, non-dry solder joints and PCBA voltages are correct. The final assembly stage tests are used for sub-circuit functionality testing and ensuring any defective PCBAs are detected and sent for repairs. During the post assembly stage products require environmental stress tests and accelerated life tests depending on the industry requirements of the product. During the post assembly tests, all anomalies are sent for failure analysis to determine whether the product design requires adjustments.

The sub-process focused on is IPQC since the SC components are sourced from reliable suppliers and the environmental stress tests are an industry requirement. The environmental stress tests are done on sampled units by a national recognised test facility following the standards set by the South African National Standards (SANS) body. The accredited certification facilities available in South Africa are the South African Technical Auditing Service (SATAS) and the South African Bureau of Standards (SABS) providing SANS certification, which is recognised by all institutions, in South Africa, for product certification. The SANS specifications are developed nationally or are adopted from the International Organisation for Standardisation (ISO) standards according to [25].

In Process Quality Control The proposed test system forms part of the IPQC where functional tests are used to ensure the SC hardware performs as expected. During this process any problems arising from assembling should be caught and rectified, minimising any after sales product returns. Since the IPQC process is mainly responsible for ensuring the correct assembly of the SC hardware, the test procedures considered are limited to the procedures used within the IPQC process.

Typical IPQC Test Procedures The IPQC process, as depicted in Figure 2.6, encapsulates various stages when specific tests are done during the assembly process. These tests are responsible for minimising defects that might occur during the each stage the assembly process goes through. The possible tests that can be performed, detailed by Eastman in [45], in the IPQC process can be described as follows.

Manual Visual Inspection (MVI) Inspecting the printed circuit board assembly (PCBA) and solder joints manually.

Table 2.4: In Process Quality Control (IPQC) Test Procedure Comparison, Extracted from [45]

IPQC Test	IPQC Stage	Setup Difficulty	Cost	Applicable
MVI	Initial Assembly	Simple/Easy	Low Cost	Yes
AOI	Initial Assembly	Moderate	Moderate	No
AXI	Initial Assembly	Moderate	Expensive	No
ICT	Final assembly	Difficult	Expensive	Yes
FP	Final Assembly	Simple	Moderate	No
JBS	Initial/Final Assembly	Moderate	Low Cost	No
SFT	Final Assembly	Difficult	Low Cost	Yes
ALT	Post Assembly	Difficult	Moderate	No
EST	Post Assembly	Difficult	Expensive	Yes

Automated Optical Inspection (AOI) Solder joints and PCBA are examined with the use of digital imagery processed by a computer.

Automated X-Ray Inspection (AXI) An x-ray is taken of the PCBA allowing inspection of ball grid array (BGA) type of surface mount solder joints.

In-Circuit Test (ICT) Probes, from a PCBA specific test rig, connect to the test points on the PCBA, measuring analog behaviour as well as injecting test signals.

Flying Probe (FP) Tests similar to ICT, but the probe is moving with the use of a robotic arm to connect to the test points on the PCBA.

JTAG Boundary Scan (JBS) Tests the digital integrated circuits (ICs) on the PCBA with the use of a test access port it allows testing of multiple ICs on the PCBA.

System Functional Test (SFT) Application specific tests, ensuring all the functionality of the PCBA works as intended.

Accelerated Life Test (ALT) Application specific tests, inducing multiple product use cases under normal operating conditions to ensure product reliability.

Environmental Stress Test (EST) Tests used to achieve product reliability requirements, which involve testing of possible extremities a product needs to endure.

The design scope can be directed to the four applicable test procedures listed as applicable in Table 2.4, namely MVI, ICT, SFT and EST. Although, EST is outsourced to SABS to receive national certification. The most applicable test for MK 3, of the SC, is the SFT procedure. Further inspection into functional test procedures are done in the following section.

2.4.2 Functional Test Procedures

The functional test procedures used in industry are rolled out with high precision, high accuracy and certified measuring instruments such as National Instruments' Peripheral Component Interconnect (PCI) Extension for Instrumentation (PXI) used in [46, 47]. These instruments are used by leading embedded companies capable of funding the required equipment and design processes. Objective 1.5 requires the designed measuring and

signal generation hardware to be verified and "calibrate[d]... against certified equipment having a known valid relationship to internationally or nationally recognised standards" as stated in the ISO 9001 quality management standards [48]. The SC is analysed to determine the required functionality tests needed to complete objective 1.5 in Chapter 1.

Functional test procedures include physically connecting to the SC and stimulating expected inputs and reading the response outputs. The strategy used in the existing test procedure is known as black box testing, where the SC is probed with inputs and the outputs are verified [49]. Although black box testing is known for a software testing method, the firmware on the SC produces output data when specific test inputs are induced.

The black box testing method involves functional software tests which needs to know only what is passed on to the system input and what is expected on the output. Based on the specifications of the Device Under Test (DUT), the test engineer creates a test procedure to ensure that the planned functionality tests are executed properly. The designed test procedure requires a test philosophy, since the outcome of a test is reliant of a balance between test time, test expenditure, the allowable failure rate and ultimately, the failure mode and effects analysis (FMEA) of the product. In the case of the SC, imminent failure of the device will lead to a power failure of the SC and inherently causing the EWH to not heat up.

2.4.3 Test Philosophy

A test system philosophy applicable to the SC for EWHs project would rely on efficient, cost effective and reliable testing methods. The number of devices requiring testing create the need for an automated test. The existing test procedure makes use of functional test methods, which motivates the use of a SFT. The automated SFT is required to detect all the possible failures during the test procedure. The FMEA of the SC relaxes the expected fail rate from extensive up to zero fail rates to a more relaxed fail rate, although the fail rate is minimised to be as close to zero as possible. The in-house production of the SCs can lead to an increase of the fail rate since the assembly of the SC is done by hand. The test procedure must be easily changeable to allow the test engineer to modify the test procedure when necessary. The test system will record each test procedure setup along with the test results. The test system design is discussed in Chapter 4 with the results in Chapter 5.

Chapter 3

Smart Controller Firmware Design

3.1 Overview

The hardware of the SC has undergone numerous developments which had an impact on the functionality and compatibility of the firmware. With the implementation of a new processor, the requirements and specifications of the firmware had changed, as it implicated the change in peripheral circuitry implemented in MK 3. In this section, these new requirements and specifications are tabulated and defined. The development of the firmware that is used to execute the features of the SC is described. Ultimately, the newly designed control firmware is explained and validated.

3.2 System Requirements and Specifications

The original firmware is incompatible with the latest hardware design, MK 3. Therefore, the firmware had to be modified to meet the specified requirements that pertain to the MK 2 hardware model, described by Brown in [24]. These updated system requirements and specifications were extracted, adapted and re-organised from [24] and are listed in Table 3.1.

3.2.1 Updated System Requirements

The system requirements determine the boundaries in which the firmware needs to operate. Therefore, the functionality that is expected from the device will operate within the stipulated boundaries. The previously established system requirements are no longer valid, and had to be changed due to the change in the hardware design of the SC. The changes in the system requirements are listed in Table 3.1.

Table 3.1: Updated System Requirements, Adapted and Modified from [24]

ID	Requirement
1	Peripherals
1.1	The system will measure inlet temperature.
1.2	The system will measure outlet temperature.
1.3	The system will measure ambient temperature.

Table 3.1: Updated System Requirements, Adapted and Modified from [24]

ID	Requirement
1.4	The system will measure outlet far temperature.
1.5	The system will measure hot water consumption.
1.6	The system will measure cold water consumption.
1.7	The system will measure power consumption.
1.8	The system will measure water leakage.
1.9	The system will measure the latching relay state.
1.10	The system will control the power supply via a latching relay.
1.11	The system will control the water flow via a latching valve.
2	Communication
2.1	The system will connect to a cellular service provider tower.
2.2	The system will connect to a central server via IP address or valid domain name.
2.3	The system will make use of a TCP connection.
2.4	The system will report the device readings minutely.
2.5	The system will limit mobile data usage.
2.6	The system will respond to remote operation commands.
2.7	The system will notify the central server of a possible leakage event.
2.8	The system will use a human-readable communication protocol.
2.9	The system will respond to a server side temperature set point controller.
3	Control
3.1	The system will implement an online and offline temperature set point controller.
3.2	The system will default to a fail-safe set point control when the device lost connection to the server.
3.3	The system will shut off the water supply when a water leak is detected.
3.4	The system will switch off the power supply when a water leak is detected.
3.5	The system will provide minutely accurate measurements and system states.
3.6	The system will aggregate all the measurements and states into a single status packet to send to the server.
3.7	The system will actuate on remote operation commands.
3.8	The system will control the element of the EWH via the power supply control.
3.9	The system will maintain a safe state whilst the leakage event is occurring.
3.10	The system will implement system control minutely.
3.11	The system will take the power supply default as <i>On</i> and the water supply default as <i>Open</i> .
3.12	The system will always report the temperatures, the non-default values and non-zero readings.
4	Risk Mitigation
4.1	The system will detect structural failure.
4.2	The system will minimize damage when a leak is detected.

Table 3.1: Updated System Requirements, Adapted and Modified from [24]

ID	Requirement
4.3	The system will have precautionary control of the water supply to the EWH.
4.4	The system will have precautionary control of the power supply to the EWH.
4.5	The system will detect water in the EWH drip tray.
4.6	The system will check for water in the drip tray after a power cycle.
4.7	The system will persist in a "Burst protection" state until power cycled.
4.8	The system will ignore water supply activation commands during the "Burst protection" state.
4.9	The system will ignore power supply activation commands during the "Burst protection" state.
4.10	The system sensors will be protected from excess current draw caused by short circuit.
4.11	The system MCU will be protected from short circuits on the sensors.
4.12	The system will cause little to no discomfort to the end-user.

3.2.2 Updated System Specifications

System specifications, listed in Table 3.2, were put in place to ensure the hardware operates within the stipulated boundaries and simultaneously provide the firmware design with its required design specifications.

Table 3.2: Updated System Specifications, Adapted and Modified from [24]

ID	Specification	Derived from
	Hardware	
HS[1]	Temperature sensors measurement range from -20°C to 150°C .	1.1 - 1.4
HS[2]	The power supply control will make use of an energy efficient latching relay.	1.10, 3.8, 4.4
HS[3]	The water supply control will make use of an energy efficient latching valve.	1.11, 3.3, 3.9, 4.3
HS[4]	Electrical isolation will be maintained in power measurements.	1.7, 3.9, 4.10
HS[5]	The leak detection sensors will function in water temperatures from 0°C to 100°C .	1.8, 3.9, 4.1, 4.5
HS[6]	The leak detection sensors will detect tap water with an atypical low conductivity of 1mS .	1.8, 3.9, 4.1, 4.5
HS[7]	All the sensors will have a measurement error of no more than 10%.	1
	Firmware	

Table 3.2: Updated System Specifications, Adapted and Modified from [24]

ID	Specification	Derived from
FS[1]	Asynchronous hot and cold water flow measurements will be taken.	1.5, 1.6, 3.5
FS[2]	Water flow measurements will range from 0 to 150 l/min.	1.5, 1.6, 3.5
FS[2]	Power consumption measurements range from 0 kW to 4 kW.	1.7
FS[3]	Energy calculation integration steps will be less than one minute.	1.7, 3.5
FS[4]	An average of continual samples for a minute's instantaneous power consumption will be calculated.	1.7, 3.5
FS[4]	The drip tray will be checked for water every 100 ms.	1.8, 3.3, 3.5, 3.9, 4.1, 4.5, 4.6
FS[5]	The system will respond within two seconds to the detection of water in the drip tray.	1.8, 3.3, 3.5, 3.9, 4.1, 4.5, 4.6
FS[6]	The system will enable a "Burst Protection" state if water is detected for more than two seconds in the drip tray.	1.8, 2.7, 3.3, 3.4, 3.5, 3.9, 4.1, 4.5, 4.6
FS[7]	The "Burst Protection" state can only be reset with a system reboot.	3.9, 3.11, 4.6, 4.7
FS[7]	The system will check for a leakage event after a system reboot.	3.3, 3.9, 4.1, 4.6, 4.7
FS[8]	The system will use a $\pm 1^\circ\text{C}$ hysteresis in the offline temperature set point control.	3.1, 3.2, 3.8, 3.10, 4.12
FS[9]	System response to remote operation prompts will be within two minutes.	2.4, 2.6, 2.8, 2.9, 3.1, 3.2, 3.5, 3.7, 3.10, 3.12
FS[10]	System will make use of the JavaScript Object Notation (JSON).	2.4, 2.5, 2.8, 2.9, 3.1, 3.5, 3.6, 3.10, 3.12

3.3 Firmware Design

One of the previous contributions to the SC for EWHs project was a new hardware design, however new firmware was not part of that research's scope. A complete redesign of the firmware was needed considering that the new MCU is more complex and does not make use of any third-party applications other than the Atmel Studio 7, a Visual Studio based software tool provided by Atmel [50].

The overall design of the firmware is a classic embedded infinite loop which is obstructed or put on hold as little as possible. This creates space for accurate and precise execution on asynchronous events, allowing robust event detection and efficient system-time management. The peripherals in use can either be asynchronous events that occur, such as a water usage event, or synchronous events, such as reading from the temperature sensors periodically. Thus, it is important to implement a stable and reliable system

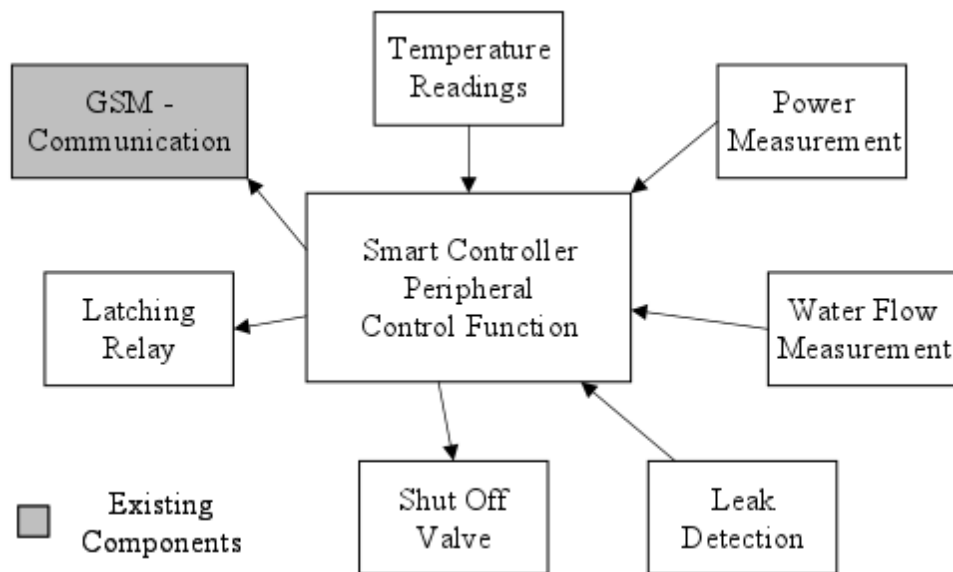


Figure 3.1: Architecture Summary of the Smart Controller Firmware.

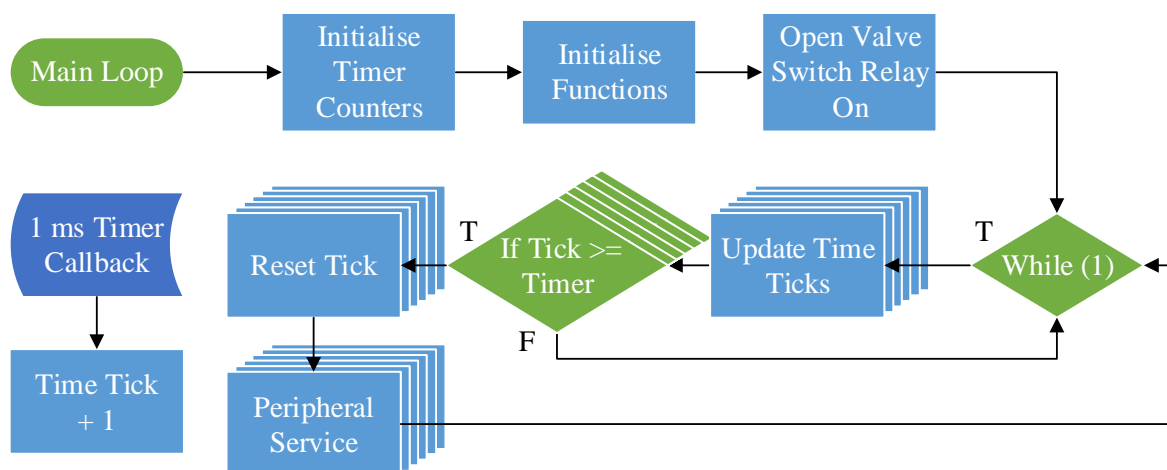


Figure 3.2: Generic Main Loop of the Smart Controller Firmware.

clock. The firmware of each peripheral is discussed extensively in the following section and a detailed software diagrams can be found in Appendix A.1.

3.3.1 Firmware Architecture

Alongside the main loop, a central control function was implemented as depicted in Figure 3.1. The control function allows the design to follow a modular design architecture, where each dedicated function of the firmware is managed accordingly. The modularity enables designing of each peripheral function in such a way to allow an external entity to request either feedback or actuation from the function. Each of the firmware design blocks is discussed in detail in the following sections.

Main Loop The firmware main loop, Figure 3.2, initialises timer counters for each peripheral, the counter values depend on the required execute rate of each peripheral. The peripheral execution rate is managed and controlled from the SC's control function. The

next process is to initialise each peripheral functionality, where each function's dedicated pins are initialised and variables are allocated. The main loop then proceeds to ensure the water shut off valve is opened and the latching relay is switched on, allowing a state reset when the device is rebooted. The infinite loop is created in which all the timer counters are updated as soon as the 1 ms interrupt callback counter is incremented. Once a peripheral's time tick reaches the predefined timer counter it will execute the peripheral service.

System Clock The MCU requires an accurate time keeping clock to ensure the MCU has a free main loop for the majority of the time and executes instructions in a nearly perfectly timed fashion. The MCU provides a 32 MHz internal oscillator used with a phase lock loop (PLL) to ensure a stable clock frequency is achieved. The MCU also provides run-time calibration by enabling a digital frequency locked loop (DFLL) from a 32 kHz crystal oscillator, allowing automatic calibration of the oscillator and optimising the accuracy according to the potential temperature and voltage drift [33].

3.3.2 Smart Controller Control Function

The SC control function utilises all the peripherals, depicted in Figure 3.1, to form a smart controller. It also makes use of the modem-side firmware (GSM-Communication) to post status updates to the cloud along with receiving commands from the cloud. The control function is responsible for the managing the set point control and the data aggregation of all the peripherals.

The control function, depicted in Figure 3.3, implements a state-machine for managing the set point controller where the controller is either controlled via the cloud or internally. The state-machine also implements the leak detected "Burst Protection" state. The control function continuously checks the modem connection to the cloud server. The safety check is primarily to ensure hot water to the user in case of connectivity issues, whether it is data depletion or reception issues.

When the device is under cloud control, the SC only awaits set point commands from the cloud where the set point is configured with a *'setter'* function. In the case of internal control, the device implements a DEFAULT set point of 60°C. The set point set by either cloud control or internal control is regulated with a set point controller. The outlet near temperature is measured and compared to the configured set point. The power supply is adjusted accordingly, whether the output temperature is higher or lower than the set point temperature. A hysteresis check of $\pm 1^\circ\text{C}$ is implemented to prevent constant boundary switching of the power supply. Once the device reconnects to the cloud, the cloud reconfigures the set point to the user defined temperature.

In the case where a leak was detected the "Burst Protection" state is activated, the control function simply switches of the water and electricity supplies to the EWH. This state is persistent until the SC has been reset at the main household distribution board.

The data aggregation is done sequentially every minute, reading all the telemetry data of the peripherals which is concatenated into a single JSON formatted report string [51]. This string is sent to the cloud minutely when the SC is powered on and connected.

3.3.3 Temperature Readings

The temperature sensors used are analog sensors which output a voltage level according to the temperature being measured. Since the sensors can measure below zero degree

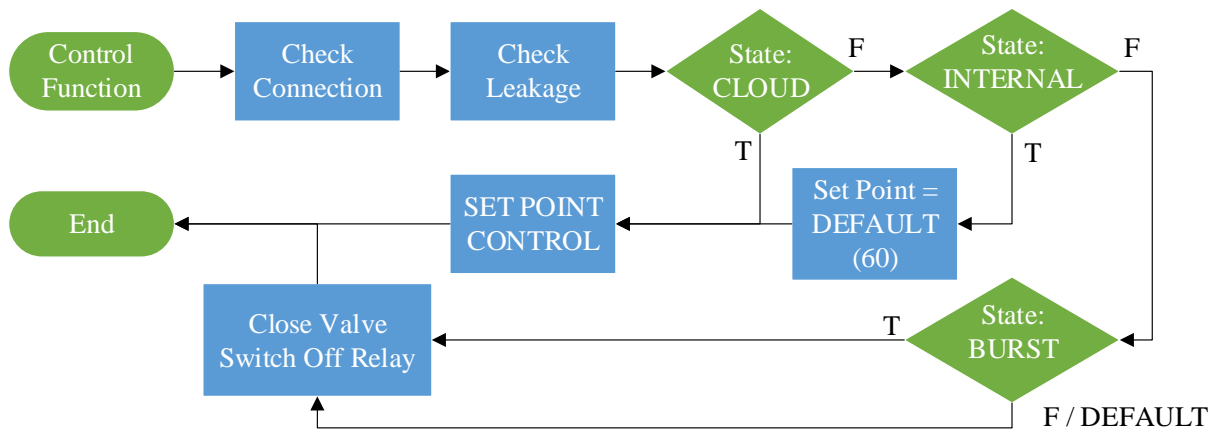


Figure 3.3: Controller Function of the Smart Controller Firmware.

Celsius, a voltage offset is implemented by the sensor. The temperature readings taken are queried minutely for the controller state report to the central cloud database. These readings are taken around the EWH body from temperature sensors extended with shielded wiring. These readings consist of the water inlet and outlet temperatures, where the outlet temperature consists of an outlet near and an outlet far sensor, the ambient temperature and an optional solar temperature sensor.

The temperature sensors output an analog voltage that ranges from 0 to 3.3 V. Before any voltage readings can be measured by the MCU, the ADC pins need to be configured accordingly. An analog read function is needed to retrieve the measured data from the MCU's registers and simplify future implementations for ADC readings.

A helper function was designed to read the specified MCU port and the specified port pin. The readings are sampled and filtered, by discarding the first 10 readings to allow the ADC to settle and produce accurate readings.

The analog read function can then be implemented into a temperature read function. The temperature read function implements an array read. This sets the analog read function in a loop to read all the specified temperature sensor pins. Since the analog read already allows for the ADC to settle, accurate readings are received. The sensors are rated to $\pm 2^\circ\text{C}$ sensor accuracy, to ensure a smooth output a simple digital low pass filter (DLPF) is implemented.

The function is designed to be called periodically so that the average temperature of each sensor can be maintained and be accurate. The temperature readings function use a synchronous timer of 250 ms to measure the analog temperature sensors. The retrieval of the data of each sensor can be achieved by using the 'getter'-function. The getter function allows for convenient retrieval of sensor data. The 'getter'-function also converts the ADC value to degree Celsius. This is achieved by converting the ADC value to the equivalent voltage and subtracting the offset voltage of the temperature sensor.

3.3.4 Power Measurement

The calculation for instantaneous power usually needs to take the active and reactive parts into consideration for accurate power measurements. The active part is what the consumer is billed by when using an EWH, thus the focus is mainly the active part. Since the EWH's element is purely resistive the active power part is equal to the apparent

power. The apparent power is equal to the product of Root-Mean-Square (RMS) current (I_{RMS}) and RMS voltage (V_{RMS}). Since the SC only supply a current measuring circuit the V_{RMS} will be taken as a constant, which is in South Africa 230 V.

The SC is equipped with a current transformer to measure the instantaneous load (EWH element) current. The current transformer produces an output signal directly proportional to the current being drawn through the wire wound through the current transformer with a primary to secondary turns ratio of 1 : 1000. The burden resistor (33 k Ω) and an additional offset resistor (1 k Ω) on the secondary side of the circuit needs to be taken into consideration when determining the calculation ratio for the current measurement.

The current transformer circuit produces a voltage at an offset which needs to be taken into consideration in the firmware design.

The EWH draws AC where the current transformer produce an identical waveform with a smaller amplitude, readable by the MCU. Ideally, the exact waveform must be captured and converted to the I_{RMS} to determine the instantaneous power drawn. Since the report rate for the SC is one minute, capturing the entire minute's waveform will cause a main loop lock and deny the other peripherals to function properly. Thus, to prevent the loop lock, periodic readings need to be taken. These periodic readings need to be sampled faster or equal to the Nyquist frequency of the utility voltage's frequency, which fluctuates around 50 Hz in South Africa, to prevent aliasing.

The analog read function was used to read the current transformer output voltage. The voltage readings are taken periodically to ensure periodic readings coincide with a periodic measured load signal. To prevent the main-lock loop, a timer is used to take a reading every two milliseconds, which is stored until the sample size of 80 is sampled. The sample size of 80 was used to capture enough samples to reconstruct the 50 Hz sine wave. The sampled values are then filtered by removing the offset voltage. The filtered samples are then converted to a RMS value by squaring each sample and add it to a sum total. The I_{RMS} is determined by multiplying the mean of the squared current with a current transformer current ratio. The current transformer current ratio is the size of the burden resistor divided by the offset resistor and timed by the ADC voltage conversion value.

This will result in the I_{RMS} which is multiplied with the constant 230 V_{RMS} to obtain the instantaneous power for the sample set. Since the report rate of the SC is set to one minute, an average of the instantaneous power is taken over 10 instances to ensure a smooth and accurate reading.

This function also allows data logging, where the total energy is added up and zeroed via the available 'getter'- and 'setter'-functions.

3.3.5 Water Flow Measurement

The water meter used to measure water consumption is an Elster Kent V100, formerly PSM 190, Utility meter [52]. The measuring method produces a pulse generated by the meter via a reed switch. The pulses generated are produced at a rate of two pulses per litre.

Since water usage events are asynchronous, the data capture needs to be able to capture the events regardless of the current state of the main loop. Therefore it is required to use the interrupt service routine (ISR) to ensure the main loop is put on hold while the water event pulses are being detected.

Specification FS[2] states a maximum of 150 litres per minute (L/min) can be expected, which will cause an expected maximum of five main loop interrupts per second.

The interrupt also only increments a counter which is read every minute by the control function to report the water usage.

Ensuring that false reads do not happen, a de-bounce time-out timer is implemented. This time-out is based on specification FS[2], this means that pulses occurring within the minimum time between pulses must be picked up but anything higher than that can be filtered out. Since the maximum pulse rate is 5 Hz, a period of 200 ms per pulse, a conservative de-bounce time-out of 100 ms is chosen to prevent false pulse readings and prevent aliasing from occurring. The reed switch produced bounce in a period of 56 μ s as depicted in Figure 2.3, showing 100 ms is long enough to miss the mechanical bounce pulses, but short enough to ensure maximum water flow rates can be measured.

3.3.6 Leak Detection

Leakage detection is reliant on the effect of solute molecules in the water supply. These molecules allow water to conduct and with this a simple stripped two-wire probe is used to measure the voltage difference between the two wires.

The measurement is done via the ADC of the MCU, thus the analog read function is used. The readings are considered to be floating voltage values, thus a persistence check is built in.

Persistence is ensured by implementing a threshold boundary. The leakage detection uses a synchronous timer of 100 ms to read the analog values. A leak is detected when the voltage measured on the leak detection pin falls below one volt. A detected leak will increment a threshold counter, otherwise the counter will be decremented. When the threshold counter has incremented to the threshold boundary, a leak detected state is broadcast to the control function, enabling the "Burst Protection" state.

The "Burst Protection" state includes, switching off the power supply to the EWH and closing the water supply to the EWH. This will reduce additional damages caused by excess water and potential electricity hazards.

Measuring a leak in a roof or outside can lead to multiple false reads due to natural interferences. These are out of the SC's control and might not give the user accurate results about a EWH leakage, but other results can be derived from it, for instance; a leakage is detected and no leakage is found, instead a rodent in the roof urinated on the sensor. With the EWH outside, it is not advised to implement the leakage detection sensor due to the nature of the sensor.

3.3.7 Shut Off Valve

The shut off valve controls the water supply to the EWH. This is done with a 12 V latching water valve. The valve requires swapped polarities to latch open and close. To allow swapping of polarities, an H-bridge is used to allow switching of the valve.

Swapping the polarities is possible by changing the input pin states. To open the valve pin A must be high and pin B must be low, to close the valve the inverse states for each pin will achieve it. Caution must be taken to prevent short-circuiting the valve pins when both pins A and B are pulled high. The valve latch from a 200 ms pulse generated from the toggled pins. Caution must also be taken to not over drive the latching valve's coil, this can be prevented with a current state memory flag.

The prevention of the above mentioned risks can be solved by implementing a simple state machine, as depicted in Figure 3.4. The state machine checks to see if the current state, which is predefined at program start to be *Open*, needs to be changed to the newly

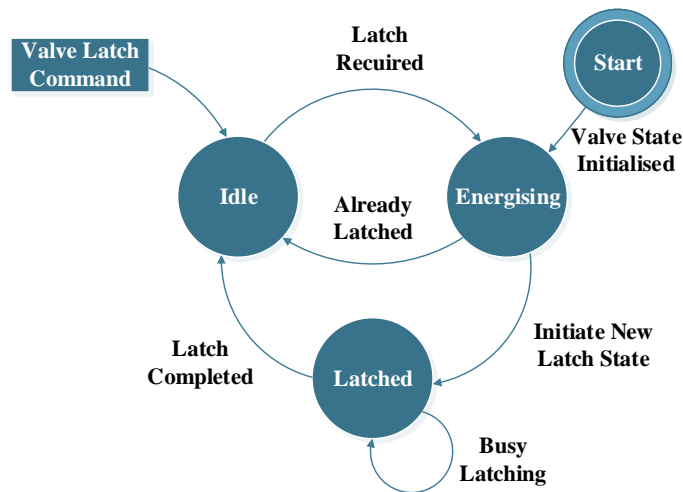


Figure 3.4: Latching Shut Off Valve State Machine used in the Smart Controller Firmware.

requested state. The valve latching state needs to be carefully monitored since there is no other way to tell in which state the valve currently resides in. The state machine will also check to see whether the correct pins are toggled to the correct states for the required amount of actuation time. Thus, the state machine will provide an efficient latching mechanism and prevent short-circuiting in the valve's H-bridge circuit.

The state-machine, depicted in Figure 3.4, consists of three states, namely: *Idle*, *Energising* and *Latched*. The initial state of the valve is forced, by starting the state-machine in the *Energising* state, to obtain the initial valve state. The state-machine will reside in *Idle* until a latching command is requested. The command will be compared with the current valve state, only if the command's state differs from the current state will the state-machine go to the next state which is *Energising*, this state will persist until the actuation time expires, once the required state pulse, *Open* or *Close*, was generated the state-machine will move to the *Latched* state where the pins will be pulled low and ensure the valve is in an idle state again. The state-machine then moves back to *Idle* and wait for the next command.

3.3.8 Latching Relay

The latching relay consists of two sub-circuits. The first being the actuation circuit and the other the safety check circuit. The safety check circuit is an opto-coupler that provides a digital high or low when the relay is either latched or unlatched, respectively. Other than the shut off valve, the need to remember the relay state is not necessary since the state can be read on demand via the safety check circuit.

The relay is controlled from two transistor driven circuits, similar to the H-bridge but a *set* and *reset* pin is designated to the relay. If the *set* pin is driven low and the *reset* pin is driven high, the relay is latched into the *On* state. The *Off* state can be acquired when the opposite is done with the pins. The relay requires a 200 ms pulse to energise the coil and latch the relay. Caution must be taken when both pins are driven high since there is no current limiting diode to prevent short-circuiting. A state-machine, depicted in Figure 3.5, almost similar to the water shut off valve state-machine needs to be implemented to ensure the actuation time and pin directions are safely managed to prevent possible coil burn out.

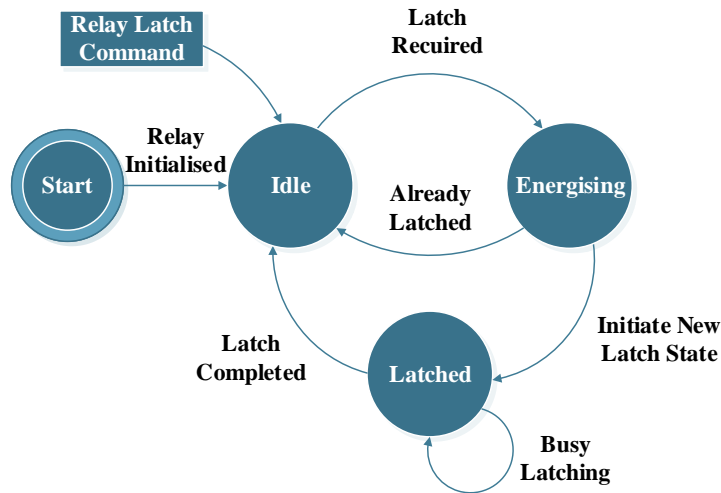


Figure 3.5: Latching Relay State Machine used in the Smart Controller Firmware.

Both the power supply relay and the valve use latching circuitry which is distinct of each other, but in firmware the state-machines are similar. The latching relay's state-machine is able to determine whether the relay has to latch or whether it is already in the desired state by reading an analog pin state. The pin state is acquired from an opto-coupler connected, via a voltage divider, to the high voltage relay circuit, measuring whether the relay is providing power to the circuit or not.

3.4 Application of Firmware

This section describes how the firmware was used in the SC eco-system. Despite the project pressures and strenuous timing, the firmware was deployed to a total of 245 devices of which 40 are derivative devices, Smart Water Meters. This particular firmware version set the foundation for future features and improvements, also allowing the existence of multiple child-devices. The Smart Water Meter from Bridgiot is one of the many derived devices [12], [53] and [54]. The final design implementation results will be discussed in Chapter 5 in further detail.

Challenges The firmware implementation was met with production time constraints and limited in-depth code testing prior to deployment, thus the need for fast paced development. The IDE required to develop the firmware proved to have a rather steep learning curve. Atmel do however provide some form of support with their Atmel Software Framework (ASF), although with limited examples explaining the use of the ASF provided functions. The application notes, provided by Atmel, provided insight on how to use the registers and pre-scalers to configure a system clock with minimal frequency jitter and configuring the ADC.

Chapter 4

Test System

"but test them all; hold on to what is good,"

— 1 Thessalonians 5:21

4.1 Introduction

The test system, shown in Figure 4.1, designed in this chapter originated from the test procedure requirements set by the existing test platform discussed in Chapter 2.3. The test system specifications are defined by taking the SC hardware specifications, discussed in Chapter 2.2, into consideration to design hardware capable of measuring and emulating signals for a test procedure to ensure the Device Under Test (DUT), the SC, is tested on the same or higher level set by the existing test platform.

The proposed test system consists of test bench hardware combined with a computing platform, specifically a Raspberry Pi, capable of running the test manager software. A system overview is depicted in Figure 4.1, showcasing the two main parts of the test system. These parts consists of the test bench hardware, interfacing with the SC via a peripheral harness. Secondly, the test manager software, interfacing with the test bench hardware via a command protocol.

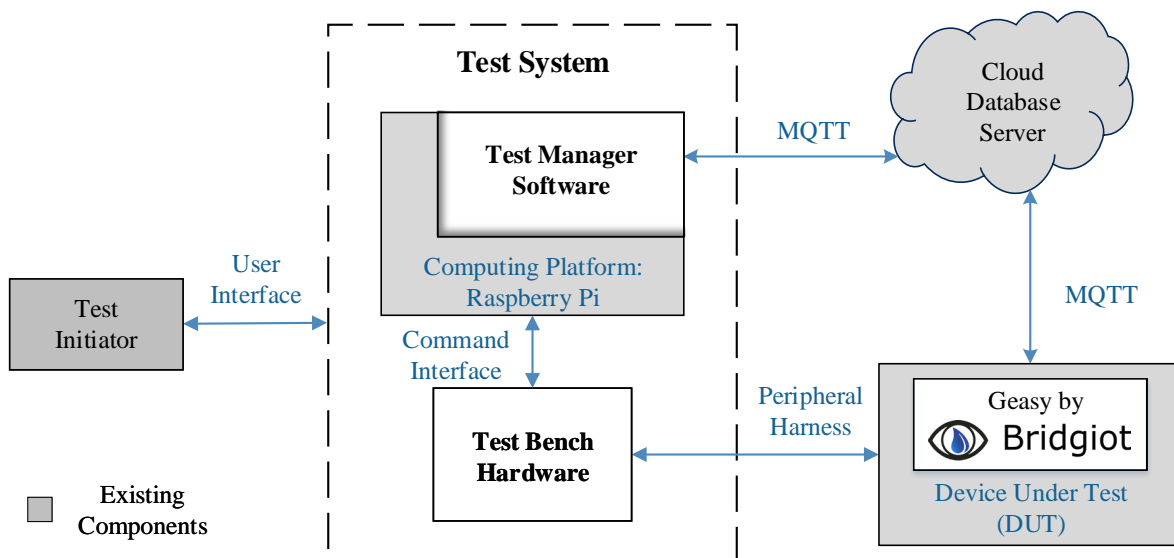


Figure 4.1: Test System Overview Diagram

Table 4.1: Automation Features Required to Achieve Test Procedure Requirements

ID	Feature Description
1	Peripherals
1.1	Change temperature measurements with voltage adjustments on the input pins.
1.2	Detect the latching pulse when the water shut off valve open or closes.
1.3	Determine latching relay state when it switches on or off.
1.4	Induce a leakage event by emulating detection leads dipped in water.
1.5	Extract power usage from SC status report strings.
1.6	Measure the SC voltages when it is switched on.
1.7	Induce a water usage event by emulating the pulses received from the water meter.
2	Communication
2.1	Configure the modem with the provided setup script.
2.2	Connect to the cloud to evaluate the report strings sent.

In this section the proposed test procedure is described and motivated, a new test procedure is designed and a proposed test bench is described to interact with the SC, being the DUT. The test bench hardware underwent a second iteration of design, improving on the initial hardware design and system integration. A design overview is given before delving into the specifics of each test bench design and how each design was implemented. The test manager software is designed to implement and manage a predefined test sequence. Finally, the proposed test procedure implementation is described by integrating the test manager software with the final test bench hardware design.

4.2 Test Procedure Design

The test procedure design is based on the analysed pre-existing test procedure requirements listed in Table 2.2 in Chapter 2.3.

Test Procedure Design Considering the existing test procedure in Figure 2.5, the tester is required to listen and observe whether the correct responses were generated by the DUT via the report string or the latching components. The test procedure can be improved by automating the observing functions by means of analog signals read by an application specific designed hardware. The newly designed test system requires the features listed in Table 4.1 for it to be able to achieve automation of the existing test platform. Additionally, the programming of the DUT can be automated since the DUT requires different versions of firmware during the test procedure to perform the modem setup, functionality tests of the test procedure and program the official release firmware. Finally, the test results can be generated into a human-readable format as well as indicating on the test bench hardware the test status via light emitting diodes (LEDs).

Implementation of Test Procedure The new test procedure, depicted in Figure 4.2, was designed to not only increase productivity of the initial test procedure by automating the test procedure, but also to ensure accurate readings and measurements were recorded and used to determine whether a test has passed or failed. The final test procedure

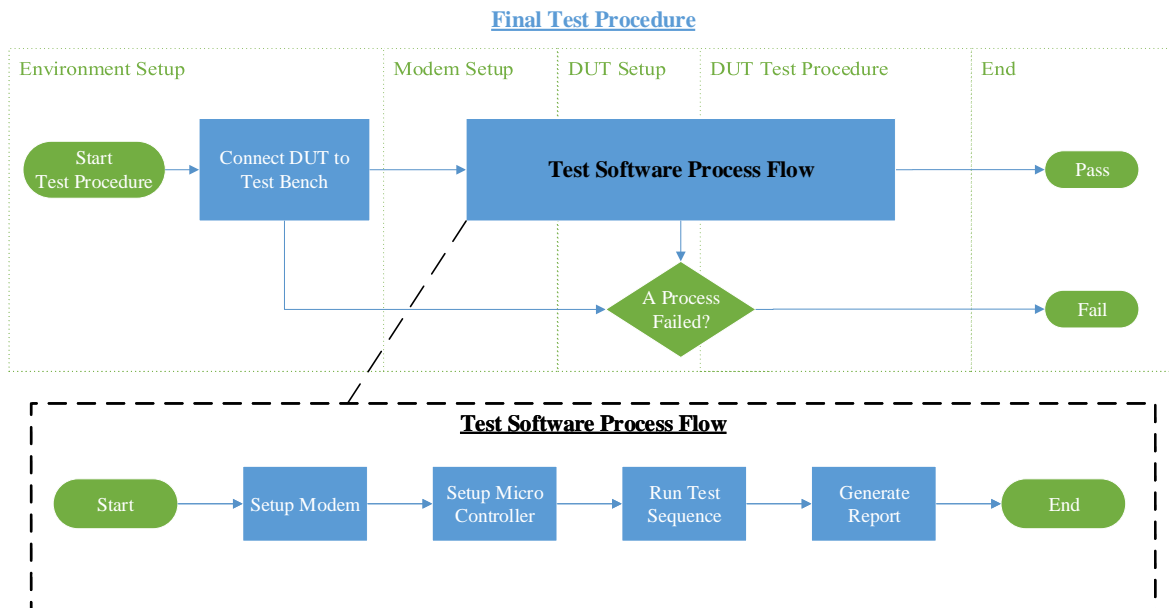


Figure 4.2: Final Test Procedure with Test Software Process Flow

has similar processes compared to the initial test procedure in Figure 2.5. Although, the processes are combined and automated in the test software process flow which only requires a tester to connect the DUT, start the test procedure and evaluate the test report.

The test software process flow encapsulates the setup processes and processes executed by the *Test Manager Software* to complete a successful test sequence. Each process is explained in Section 4.5. The final test procedure, depicted in Figure 4.2, implements a modified version of the existing modem setup script, which makes the details extracted from the DUT available to the subsequent processes. The test procedure will automatically use the obtained information to initiate the next process. The DUT will be connected to a Raspberry Pi (RP) capable of programming AVR MCU chips, thus the programming section will be done automatically. Once the DUT has been programmed, the test sequence will be executed. The test sequence will be customisable by the test engineer, allowing custom test cases for different test scenarios. Each scenario will be implemented with the use of a configuration file accompanying the test manager software. This software is responsible for managing the specified test case and comparing the test results with the specified expected results or values. Finally, the test data and results are combined into a human-readable format.

4.3 Proposed Test System

A test system is proposed, which will be able to execute the improved test procedure discussed in Section 4.2. The test system will require an interface to the SC to be able to produce emulated signals and take measurements. The proposed choice will be a harness consisting of shielded wiring to minimise possible noise on the emulated and measured signals. The test procedure will be able to decrease test times if an automated test procedure is executed, thus a processing platform will be required. A software program will be able to manage and control the test procedure according to a pre-defined test sequence. The software will be required to manage the emulation and control of the hardware, the

cloud communication to and from the SC whilst maintaining the test sequence. Thus a multi-threaded processing platform will be required and a programming language optimised for multi-threading. Finally, all the design choices have to minimise costs to achieve Objective 1.5.

Processing Platform The processing platforms considered, in Table 4.2, are single-board computers, which is readily available, have an operating system and is cost efficient. The platforms considered are the Raspberry Pi (RP) platforms, where the RP 2 is cost wise attractive, it is limited to only an ethernet port for connectivity and the power supply capability may be a potential limit. The platform will have to be able to supply enough current to activate relays, produce emulation signals and measure signals. With proper circuit designs and energy management, the RP 2 might be a viable option, but for the initial design of the test system, energy efficiency is not an objective. The RP 3 provides on-board wireless connectivity, which makes remote management and development a viable option, but the main feature to consider is the 2.5 A current limit on the RP 3. The RP 3 is also considered above the other platforms since it is a successor to the RP 2, which has an well established developers community.

The RP 3 provides 40 General Purpose Input and Output (GPIO) pins, these pins provide an Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI) bus with two slave select pins, Inter-Integrated Circuit (I2C) with a 5 V and two 3.3 V power supply voltage pins.

The SPI can be used to configure two slave devices, the slave devices applicable to the test system would be Analog to Digital Converters (ADCs) or Digital to Analog Converters (DACs). These slave devices are managed from the SPI bus via the master, which is the RP, with the use of three SPI pins and an slave select pin, each dedicated to a specific slave device. The three SPI pins consist of a Master-In-Slave-Out (MISO), Master-Out-Slave-In (MOSI) and a Serial Clock (SCLK) pins, these pins are one-directional [55].

The I2C uses two communication wires to connect to the slave devices, although with I2C the slave devices can be more than two, each wire represents a Serial Clock (SCL) and a Serial Data (SDA) line on the master device, the RP, and slave devices [56]. Each slave requires an unique address on the communication bus, so long as the device addresses are not the same, slave devices can be added. The RP can either use SPI or I2C to communicate to serial peripherals such as the ADC or DAC suggestions.

The UART is a widely used serial communication scheme. Unlike in the cases of SPI and I2C, the UART does not need a physical clock line, instead a transmission rate is defined in each device. This transmission rate is known as the *baud rate* at which data are transmitted or received. The baud rate is an approximate bit frequency at which data is transmitted [57]. The UART uses the baud rate to configure its internal clock to sample the incoming data correctly as well as transmitting the data accordingly, for another devices to communicate with each other via UART, their baud rates must be set the same. Additional configuration is also available, such as defining the parity, a rudimental error checking technique, the amount of stop bits, normally set to one, and defining the amount of data bits send, which is normally set to eight bits.

Software Tussle The software program required to maintain and manage the test system, requires features for running multiple code sequences simultaneously and provide a means of aggregating relatively large resultant data. These features are supported in Object-Orientated Programming (OOP) languages, which means objects can be created

Table 4.2: Compared Single-Board Computer Platforms

Platform	Features	Cost (ZAR)
Raspberry Pi 2 Model B	900MHz quad-core ARM Cortex-A7 CPU 1GB RAM 4 USB ports 40 GPIO pins Full HDMI port Ethernet port Combined 3.5mm audio jack and composite video Camera interface (CSI) Display interface (DSI) Micro SD card slot VideoCore IV 3D graphics core Micro USB power source up to 1.8A	647.19
Raspberry Pi 3 Model B	1.2GHz quad-core Broadcom BCM2837 CPU 1GB RAM BCM43438 wireless LAN and Bluetooth 40-pin extended GPIO4 USB 2 ports 4 Pole stereo output and composite video port Full size HDMI Camera interface (CSI) Display interface (DSI) Micro SD card slot Micro USB power source up to 2.5A	886,83
Beagle Bone Black	AM335x 1GHz ARM Cortex-A8 512MB DDR3 RAM 4GB 8-bit eMMC on-board flash storage 3D graphics accelerator NEON floating-point accelerator 2x PRU 32-bit microcontrollers USB client for power & communications USB host Ethernet HDMI 2x 46 pin headers 2.1mm 5V power source up to 1A	1 134,27

with its own data properties and functionalities in a class. An object can be used in conjunction with other objects to create a symphony of various objects interacting and manipulating each other. Objects are the programming variable to which a class instance is linked, a class is the definer of the data properties and available functions. The OOP languages provide class specific data containers, the objects, which is only aware of the pre-defined data properties in its class thus, an object manages its own memory allocations which reduce the risk of memory corruption.

The tussle is in choosing the right programming language for the application. The application at hand requires peripheral control, UART communication, cloud communication and some degree of data processing. According to an analysis by Stackoverflow, the most renown OOP language is Java, but the fastest growing OOP language is Python [58]. The previous contributors also used Python for managing and controlling the existing test procedures, the setup script of the SC and the modem configuration is all done in Python. The only drawback found with choosing development with Python is that Java leads with computational efficiency and performance although neither Java nor Python is optimized for high-performance computing [59].

The choice was made to develop in Python 3, since this is the language most of the contributors are familiar with and high-performance processing is not required in this application [59]. The available open-source support is vast and Python is an agile developing language which reduces development and implementation time.

4.4 Test Bench Hardware

4.4.1 Proposed Test Bench

The test procedure requires a physical interface to the DUT, enabling the test procedure to emulate and/or take measurements from the DUT. This section will describe a proposed test bench capable of interfacing with all the peripherals of the SC. By either emulating or simulating each peripheral input to the SC the test bench will form a virtual EWH connected to the SC. The test bench will be required to measure or emulate at a resolution higher or equal to the SC hardware specifications. Ensuring to reach the required resolution with the proposed test bench, the hardware of the latest SC design in [24] is used in conjunction with the updated specifications listed in Table 3.2.

The proposed test bench requires a set of requirements and specifications that need to be fulfilled to ensure the test functionality covers all the required areas and the specifications specify how the requirements must be implemented. The determined requirements and specifications are listed in Table 4.3, each specification follows the respective requirement.

Test Bench Features A proposal of the test bench design is depicted in Figure 4.3 as a black box device, laying emphasis on the desired signals being measured or emulated. The test bench will be designed to connect onto a RP 3. The RP 3 is chosen since it is capable of supplying 2.5 A, of which 1.2 A can be supplied through the USB peripheral port. It also has built-in Wi-Fi capabilities, is within budget, and allows multi-threading. It is also capable of flashing AVR firmware via its GPIO pins, ideal for programming the SC firmware. The test procedure proposed in Section 4.2 will require a test bench with at least 12-bit resolution for the voltage measurements and emulation. Constant water flow pulses and time dependent leakage events will be produced by the test bench. The

Table 4.3: Test Bench Requirements and Specifications

ID	Requirement
1	Requirements
1.1	Measure the operating voltages of the DUT.
1.2	Determine when the water shut off valve changes state.
1.3	Manage the power supply to the DUT.
1.4	Manage at least two loads connected to the DUT.
1.5	Emulate water flow events.
1.6	Emulate a leakage event.
1.7	Emulate at least 4 temperature sensors.
1.8	Provide serial communication functionality with the DUT.
2	Specifications
2.1	Measure at least 3 analog voltage levels ranging between 3.3 and 12 V within 10% accuracy.
2.2	Emulate the maximum current drawn of 1 A by the shut off valve.
2.3	The live and neutral connections of the DUT must be controlled.
2.4	The load switching relay must withstand at least 10 A being drawn.
2.5	The water flow emulation must emulate the internal resistance, 10 Ω , of the reed switch used with the Elster Kent V110 water meter.
2.6	The leakage detection emulation must emulate the resistance of drinking water, 0.0005 - 0.05 S/m (2000 - 20 Ω m) [60].
2.7	Temperature emulation requires at least a 10 mV resolution to emulate the MCP9700 temperature sensor.
2.8	The DUT use UART, the test bench must be able to send and receive messages via UART at 115200 baud.

**Figure 4.3:** Test Bench Black Box

triggering of the water shut off valve will be detected by using an additional valve sense circuit on the test bench. The latching relay will be detected by inspecting the report strings and evaluating the SC responses.

The RP UART serial communication port will be used to set up the modem of the SC and receive debugging messages combined with the report strings of the SC. The voltage emulator will be used to emulate the temperature sensor voltage levels, where each degree Celsius requires 10 mV increments. The voltage measurements are used to check operating voltages of the SC on various locations. The water flow pulser and leakage emulation will provide accurate water flow pulses and short circuit the SC leakage detection pins accordingly. The valve sense circuit will allow the test bench to determine a latching pulse on either positive or negative pin. The relays will be used to manage the power supply to the SC and the loads it is connected to.

4.4.2 Test Bench MK 1 - Design and Implementation

The initial test bench design, depicted in Figure 4.4, was focused on creating an add-on hardware module for a Raspberry Pi by using the Hardware Attached on Top (HAT) standard. The first design, namely MK 1, enabled the RP to control the test bench peripherals directly from the GPIO pins managed by the *Peripheral Manager* (PM). The MK 1 design was a trial phase determining plausible solutions for the final test procedure implementation. Various circuit design implementations was used to determine the extent

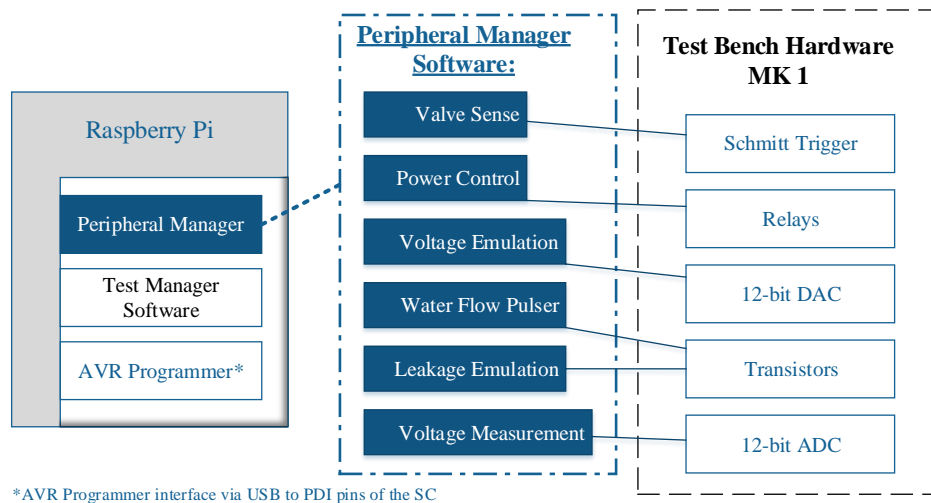


Figure 4.4: Test Bench Hardware MK 1

of automating the test procedure. The design and the implementations are described in the following sections.

4.4.2.1 Hardware Design

The hardware design consists of multiple circuits implemented on a printed circuit board (PCB) able to fit onto the RP extended header pins. Each peripheral, depicted in Figure 4.4, was designed according to the specifications listed in Table 3.2 and the schematic sheet is attached in Appendix B.1.

Voltage Measurement The voltage measurement circuit was needed since the RP does not have an internal voltage measuring capability. The chosen option was a high-resolution 12 bit, single channel ADC, MCP3201, in conjunction with a multiplexer, 74HC4051, to enable multiple measuring points to be read.

The RP will interface via the available Serial Peripheral Interface (SPI) with the ADC chip and manage the voltage measuring point with the multiplexer.

The voltage measurement circuits provide voltage dividers to allow measurements of up to 12 V.

Valve Sensing The circuit, depicted in Figure 4.5, is designed in to enable detection of polarity changes caused by the water shut off valve. The latching water shut off valve receives a bi-polar 12 V pulse from the SC. The polarity of the pulse determines whether the valve is latched opened or closed. Due to this characteristic, the designed valve sensing circuit uses a diode bridge rectifying circuit that provides two paths for the current to flow, by directing the current with the use of the diodes. The pulse detection was designed with the use of a Schmitt trigger, 74LVC2G14, combined with a diode rectifying bridge to allow measuring polarity changes from the shut off valve circuit of the SC. The circuit also emulated the current draw of the shut off valve, with the use of high wattage resistors, allowing one Ampère.

Power Control Four relays are used to control the power supply to the DUT as well as managing which load is connected to the DUT. Two relays are dedicated to controlling

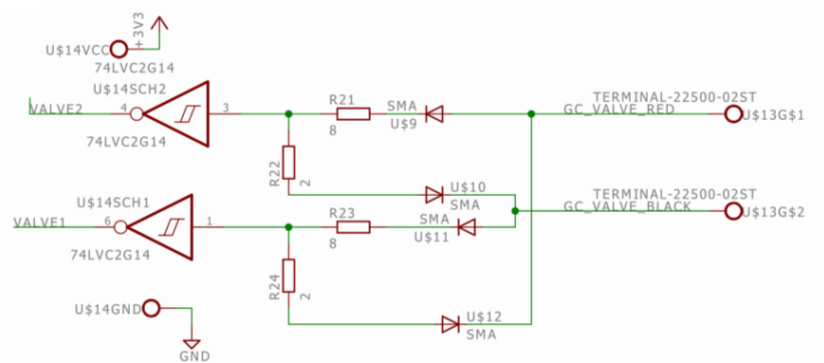


Figure 4.5: Valve Sense Circuit with Schmitt Trigger from Test Bench MK 1

the power supply to the DUT, one for live and the other for neutral. The other two relays are dedicated to the load controlled by the DUT, allowing the test bench to switch to two different loads during the test procedure. The power control circuit is designed with the HF102F non-latching relay. Since the test bench does not need to maintain a certain power state for extended periods of time it was decided to use the non-latching relay type. All four relays use the same control circuit, which is a simple transistor switch with a diode to prevent the inductance discharge from the relay to cause potential damage to the transistor. The RP can easily control the power supply and the loads by pulling the transistor base pin high to activate the respective relay.

Voltage Emulation The RP is capable of producing Pulse With Modulated (PWM) signals from its GPIO pins, but due to the temperature sensors being used in the SC design the resolution of the generated voltage signals need to be within $10 \text{ mV}/^\circ\text{C}$.

Ensuring an accurate and stable output voltage, a single channel 12 bit resolution DAC, MPC4921, was chosen to emulate the temperature sensor voltage levels. The DAC is also connected to a 74HC4051 multiplexer to provide multiple voltage emulation points to the DUT.

Water Flow Pulser The water flow meter produces 2 pulses per litre via a reed switch, the pulses are 3.3V and the pulse frequency increases as the water flow rate increases. A transistor switch was designed to ensure fast enough switching to allow for a wide range of pulse frequencies. According to the specifications in Table 3.2 the maximum pulse frequency expected is 300 pulses per minute, which is a maximum pulse frequency 5 Hz. This is well in range of the switching frequency of a 2N2222 bipolar junction transistor.

Leakage Emulation The leakage emulation is also done with a 2N2222 transistor switch with a load resistance that is similar to the resistance of water which is roughly 100Ω . A leakage event can be emulated by pulling the activation pin high from the RP. Leakage event duration is determined from the duration of the pin being pulled high.

Changes were required to ensure the transistor saturates and the output pulse is pulled to the correct voltage level. The *DRIP_LOW* pin, depicted in Figure 4.6, requires to be pulled directly to ground, but in the SC circuit the pin connected to *DRIP_LOW* is pin 2 of *P14*, and its connected with a $1 \text{ k}\Omega$ resistor to ground. The *DRIP_LOW* pin was changed in hardware to connect directly to the test bench ground.

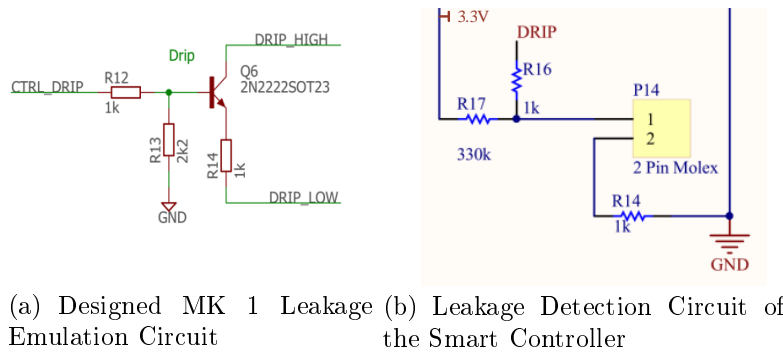


Figure 4.6: Leakage Emulation Circuit Diagrams

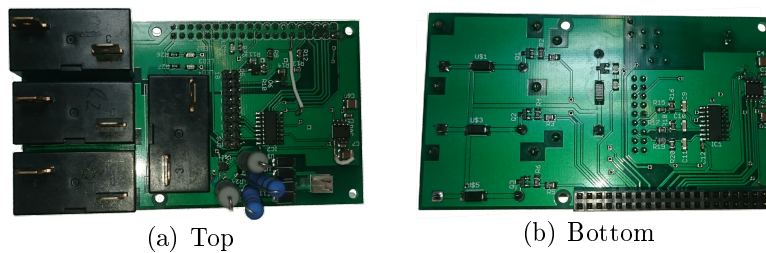


Figure 4.7: Developed Test Bench MK 1

4.4.2.2 Implementation

The test bench, depicted in Figure 4.7, was extensively tested to ensure all the designed peripherals and features work as intended. As could have been expected from a trial phase, the MK 1 had a few design flaws which led to the decision not to implement the test bench into the entire test procedure, but rather improve on the design first.

Peripherals The test bench, MK 1, comes with a 12 bit ADC and a 12 bit DAC controlled with SPI. The ADC pins allow 3 voltage inputs, maximum voltages are 3.3 V, a 3.8V and a 12V input. The DAC provides 5 output pins interchangeable via the multiplexer. The water flow emulator provides a two-pin connector with a resistance, 10 Ω , equal to the internal resistance of the reed switch, used by the SC, making the connection identical as to the actual connection from the installed water meter. The water flow emulator can produce pulses up to 20 Hz . The leakage emulator design is similar to the water flow emulator, only with a larger load resistance of 100 Ω to simulate the water's actual resistance. Four relays are provided to manage the power supplied to the SC as well as control the loads dedicated to the SC. The test bench fits onto a RP, where the communication pins of the RP is extended to allow UART communication to and from the SC.

Test Bench MK 1 Setup Requirements The MK 1 design requires additional control logic to allow the test bench to operate within the ranges specified in Table 4.3. Since a RP is used to control the test bench peripherals, a default peripheral state is required during device start up. The default peripheral state will allow safe connecting of the test bench onto the RP. The RP also needs to be enabled to control the test bench via SPI

and UART communication via the header pins needs to be enabled as well. Finally, the test bench can be used to implement the final design of the test procedure.

Raspberry Pi Setup The RP is capable of using SPI for communicating to SPI enabled components, the DAC and ADC, this needs to be explicitly switched on in the Raspberry Pi Configuration Tool (raspi-config). A Python package Python Spidev [61], was used to interact with the SPI via Python.

The UART communication between the SC and the RP via the header pins also needs to be enabled explicitly, this is done through the raspi-config terminal by enabling the GPIO serial port communication.

The operating system used on the RP is Raspbian, a free Debian based operating system used to run a peripheral control program to control the test bench. The peripheral control program was used to initialise the test bench to a default off state once the RP powers on. This was achieved by pulling the extended header pins to their respective off states as soon as the RP switches on. A Raspbian scheduling package was used to initiate the test bench default start up sequence, allowing safe connecting and disconnecting of the physical test bench when the RP was initially switched on.

4.4.2.3 Test Bench MK 1 Limitations

The test bench connected to the RP can send and receive messages from the UART, control the DAC and ADC via SPI, switch relays on and off, emulate the water flow and the leakage events. Test procedures can be run and the measured data or commands issued to the test bench can be controlled from a software program launched from the RP.

The usage integration with the test procedure was not implemented fully due to the problem areas mentioned. The peripherals were all tested and an informal implementation of the test procedure was used to test the functionality of the test bench, which proved useful in identifying possible hardware improvements.

Problem Areas The valve sensing circuit of the shut off valve failed to produce the expected output. The Schmitt trigger chip was unable to produce a stable output signal when the valve latched which was mainly due to the high current drawn by the circuit since the chip has a low operating current rating. The output signal received after the rectifying circuit produced a voltage offset on the opposite measuring input pin, producing false detections.

A minor modification was required to ensure proper transistor saturation occurs in the leakage emulation circuit, Figure 4.6.

The transistor switching circuits for the relays proved to work, although the design can be improved by reducing the base resistor 1 k Ω , value to increase the holding current. The holding current will allow a more stable connection since the current design caused the relays to loose contact when the test bench takes light impact.

The test bench does not provide any user interface to allow the tester to manage the test procedure from the hardware itself.

4.4.3 Test Bench MK 2 - Design and Implementation

The MK 2 design depicted in Figure 4.8 encases the *Peripheral Manager* software into the physical add-on RP HAT board by incorporating a MCU directly onto the test bench.

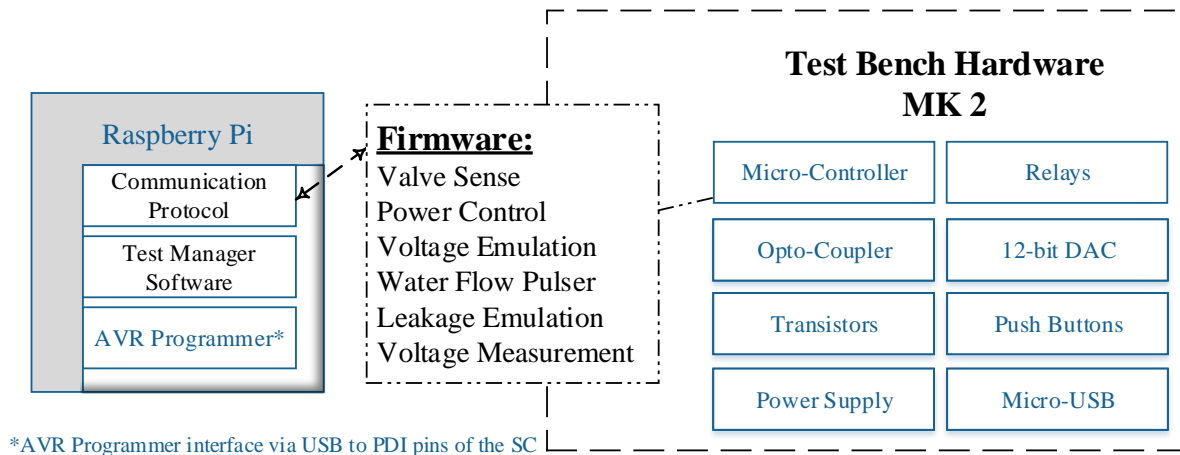


Figure 4.8: Test Bench Hardware MK 2

Table 4.4: Test Bench Hardware Features: MK 1 *vs* MK 2

Feature	MK 1	MK 2
Voltage Measurement	Single Channel 12-bit ADC.	12 Channel 12-bit ADC.
Water Flow Pulsar	Transistor Switch.	Transistor Switch.
Voltage Emulation	Single Channel 12-bit DAC.	4 Channel 12-bit DAC with EEPROM.
Leakage Emulation	Transistor Switch.	Transistor Switch.
Valve Sense	Schmitt Trigger Circuit.	Opto-Coupler Circuit.
User Interface	N/A	4 Push Buttons.
Power Supply	N/A	Micro-USB Combined with a 5 V to 3.3 V Regulator and ESD Protection.
Micro Controller	N/A	Atmel ATXmega128A4U.

The intended design was to improve the usability of the test bench by giving it a thread of its own. This enabled a design feature to control the test bench peripherals over UART or USB instead of having multiple pins directly connected to the RP and relieves the RP from having to maintain an additional thread. The MK 2 design also improved design flaws found in MK 1, namely the valve sense circuit and the relay control circuit. The improved design relieves the pin usage on the RP, providing capacity for additional future features to be easily implemented onto the test bench. The design of MK 2 is described in this section, and the detailed design schematic can be found in Appendix B.2

4.4.3.1 Hardware Design

The hardware changes in MK 2 was focused on simplifying the peripheral control software with firmware onto a MCU and reducing the PCB footprint. The main changes were to add a MCU onto the HAT and changing the DAC to an I2C controlled 4 channel DAC with a built-in Electrically Erasable Programmable Read-Only Memory (EEPROM), MCP4728, removing both the multiplexers from the ADC and DAC designs used in MK 1. The valve sensing circuit was improved and the transistor base resistors of the relay control circuits were changed. Added features for MK 2 include push buttons, programmed to request system, MK 2, settings or a status update from all the measuring peripherals, a separate communication protocol controlling all the peripherals, and a micro-USB connection which allows stand-alone functional operation of the test bench. The hardware changes from MK 1 to MK 2 are listed in Table 4.4 and described in this section.

Voltage Measurement The voltage measurements are done with the on-board ADC of the ATXmega128A4U chip. The voltage measurements are mainly to ensure the SC voltage regulators, transformers or switch-mode power supplies produce the correct voltage to the SC. Voltage dividers are used for all the input voltages since the operating voltage is 3.3 V and the ADC reference voltage is V_{CC} 1.6, thus all the measured voltages must range from zero to 2.0625 V. The voltage dividers are for a 12 V, 3.8 V and 3 V input.

Voltage Emulation The voltage emulation functionality makes use of an I2C DAC, MCP4728-E/UN, with a four channel output and a built-in EEPROM. Allowing voltages to be set and the peripheral controller does not need to maintain the set voltage, such as the case in MK 1 where the multiplexer required a specific controller to maintain the voltages on a single channel. The four channel output allows multiple voltage outputs simultaneously.

Valve Sensing The valve latch sense circuit was redesigned from the original MK 1 design and is depicted in Figure 4.9. A similar rectifying circuit was used, although pull-down resistors was implemented to remove false positives, also an opto-coupler, LTV-826S, was used to replace the Schmitt trigger in the circuit. The MCU simply triggers once a signal is measured on either the positive or the negative input pin. The opto-coupler isolates the high amperage and voltage spike received once the valve latches open or closed, protecting the test bench from possible over-current damage.

The valve sense circuit depicts 100 Ω pull-down resistors, although it was designed for 300 k Ω resistors and were populated with it.

Power Supply The board power supply is either received from a micro-USB connection, with ElectroStatic Discharge (ESD) protection with the use of a protection chip, IP4234CZ6, and a combined, 5 V to 3.3 V linear voltage regulator and 300 mA linear current regulator, or directly from the connected RP. The chosen regulator, MCP1824, provides a 500 mA maximum pulsed output for a period less than 10 ms, ideal for when all the relay transistors are switched on. The regulator provides 3.3 V to all the peripherals except to the relays which acquire 5 V directly from the micro-USB or RP connection. The entire board can also receive power, 5 V and 3.3 V, as in the MK 1 design from the RP when it is attached via the header pins.

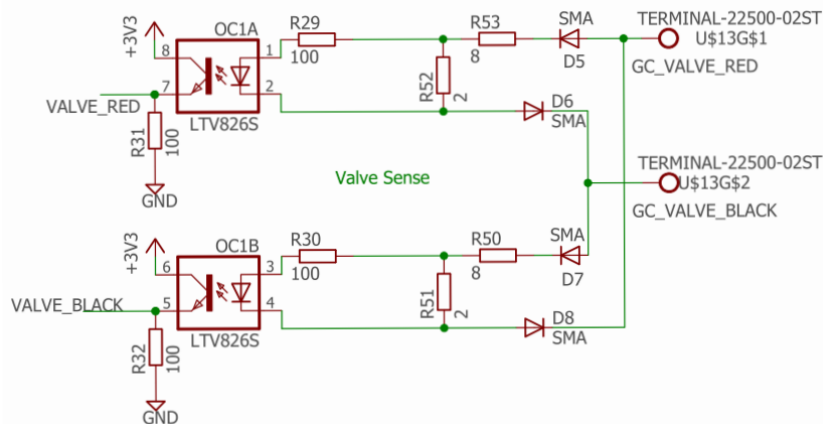


Figure 4.9: Improved Valve Sense Circuit of Test Bench MK 2

Micro Controller A micro controller was chosen to simplify the overall system control and improve reliability. The main reasons for choosing the ATXmega128a4u as a MCU was firstly, the confidence instilled in the MCU, since it is used in the SC itself and secondly, the experience of firmware design for the SC enabled fast and efficient firmware development for the MK 2 test bench. The MCU also allows the test system as a whole to be managed from any platform capable of serial communication either directly from the RP or via the micro-USB connection.

A layer of complexity in the test system is removed by implementing the peripheral manager, previously managed from the RP, directly onto the test bench MCU firmware. The firmware implements a communication protocol designed to control all the peripherals of MK 2 and provide a programmable periodic output of all the measurements.

The MCU provides an 12-bit ADC, which is used to replace the previous voltage measurement design in MK 1. The same designed function, analog read, used in the SC firmware was used to measure the voltage levels. The ADC was used to read three pins, two of the pins are capable of reading 3.3 V and the other pin is capable of reading 12 V. These readings are taken only to ensure the DUT operates at the correct fixed voltages, thus no transient voltages need to be measured.

4.4.3.2 Firmware Design

The firmware design for each peripheral function listed in the main firmware loop, Figure 4.10, will be explained in this section and more in depth function diagrams can be found in Appendix B.3. The firmware implements a designed communication protocol to manage the on-board test bench peripherals. Both the peripheral control functions, and the communication protocol, is designed to ensure a stable and reliable test bench and provides an interface to manage and control the peripherals from an external computing platform. The firmware was developed in the C-language with the use of the Atmel Studio 7 IDE [50].

USB Connection Service The connection service could be easily implemented from the available Atmel Software Framework (ASF). The USB device interface (UDI) was configured to the communications device class (CDC) which provides an interface to managing serial communications with the USB connection. The communication port defined by the CDC is a low rate port, typically for communications with a baud rate of

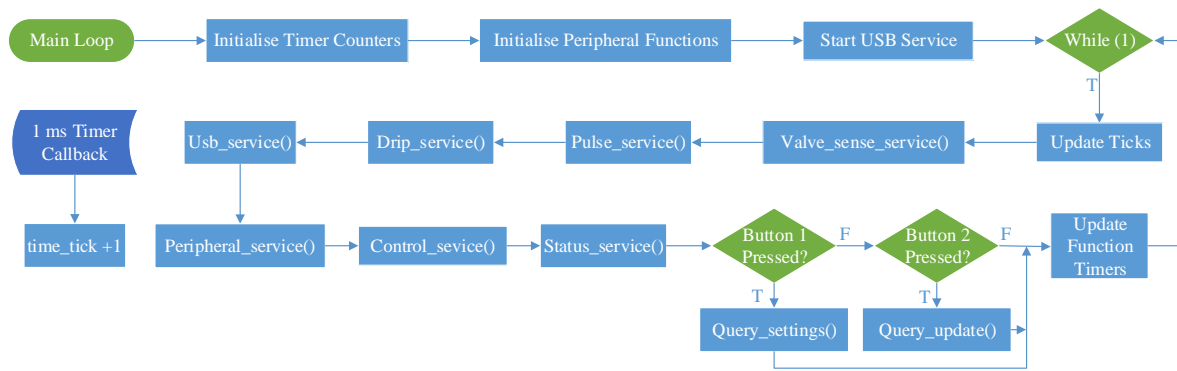


Figure 4.10: Test Bench MK 2 Main Firmware Loop

less than 512000. The internal 32 MHz RC oscillator needed to be up-scaled with the provided internal clock pre-scalers to provide a 48 MHz clock for the USB communications. The system is unable to use the USB clock as a clock source in the main loop, thus the 32 MHz clock was pre-scaled down to a valid frequency of 24 MHz, which is sufficient for the peripheral management firmware.

Valve Sense Service Since the input to the MCU from the valve sense circuit is an opto-coupler, the two input pin states are read for a digital high or -low state. The valve sense service, in Appendix B.3, is continuously reading each pin state, a notification flag is set allowing the pin state to be prompted from a ‘*getter*’ function when the latching pulse is read.

Pulse Service The pulse service, in Appendix B.4, is responsible for the pulse generator of the water flow emulation. The pulse generator use a start flag to initiate a water flow event, this event is defined by a ‘*setter*’ function. The emulated water flow is defined by implementing two parameters, the pulse frequency and the amount of pulses. The maximum amount of pulses is limited to 1000 pulses, and the frequency of these pulses, a maximum of 20 Hz can be specified. The boundaries prevent possible buffer overflows but still allows an implementation of a larger spectrum of frequency pulses. The maximum expected water flow rate on the SC is 5 Hz, thus any frequency higher is only for research purposes and limit testing.

Drip Service The leakage emulation was implemented similar to the water flow emulation, with the pulsing feature being a static state change instead. The drip service, in Appendix B.5, requires an activation flag to be set before the leakage event is generated. The leakage event is defined with a ‘*setter*’ function which requires two parameters, the leakage state and the leakage duration. The leakage state needs to be enabled to allow the drip service to execute the leakage event, and the leakage duration starts a countdown timer counting down the duration the leakage event must be maintained. The leakage emulation was designed with a maximum duration of 60 seconds to allow for longer durations since the SC must respond within two seconds and mainly to prevent buffer overflows in the firmware.

Communication Services The communication services comprises of two main loop functions, the USB service and the peripheral service. Each service is responsible for

Table 4.6: Firmware Communication Protocol

Function	Command	Parameters
Relays	#R, <i>x</i> , <i>x</i> , <i>x</i> , <i>x</i> \$	$x = 1 : on, x = 0 : off$
Water Flow Emulation	#W, <i>x</i> , <i>y</i> \$	$0 \leq x \leq 20$ Hz, $1 \leq y \leq 1000$ pulses.
Leakage Emulation	#D, <i>x</i> , <i>y</i> \$	$x = 1 : on, x = 0 : off$, $1 \leq y \leq 60$ seconds.
Set Voltage Levels	#V, <i>x</i> , <i>x</i> , <i>x</i> , <i>x</i> \$	$0 \leq x \leq 3300$ mV
Status Report Period	#S, <i>x</i> , <i>y</i> \$	$x = 1 : on, x = 0 : off$, $1 \leq y \leq 120$ seconds
Query Settings	#QS\$	N/A
Query Update	#QU\$	N/A

managing the respective communication channel for in-bound messages as well as the sending of out-bound messages. The peripheral service requires additional UART setup configuration, where as the USB service implements a USB manager provided by the ASF, which is well documented. The communication services provide the latest in-bound message to the message parser, in Appendix B.7, which is responsible for decoding any commands received. The commands are decoded into a peripheral structure. The peripheral structure is updated with each new received message, the peripheral structure deems as a command list that needs to be executed. The messages containing the commands need to comply to the designed communication protocol formatting, otherwise the message will be rejected.

The communication protocol is the gateway to the designed firmware. The design is of such a nature that the test implementer can easily set test conditions without the need to fully understand the inner workings of the firmware itself. A custom designed protocol is used where the input message have to start with a '#' character and end with a '\$' character. Each peripheral and system query have a specified identifier which selects the designated peripheral. The protocol is designed to send the peripheral identifier followed with its arguments, delimited with a ',' character. The complete set of commands are listed in Table 4.6. Precautions were taken to ensure only valid messages are executed. The first is to identify the start and end characters of messages, to ensure complete and non-corrupt messages are decoded before the commands are executed. Another precaution is taken to ensure the arguments received is within the boundary specification of each peripheral before it is passed to the destined function. Two UART ports, routed to the USB and the RP extended header, on the MCU can be used to initiate the protocol for peripheral control or system queries. The out-bound communication relies on two JSON formatted strings. The first being the settings string, which is the expected reply when the '*Query Settings*' command is sent to the test bench. The settings string contains a summary of the last set operational command settings. The second out-bound message is the status update string, which comprises of the valve sense results and the voltage measurements. The valve sense results depict which connection of the valve received the triggering pulse, where "Valve Red" ("VR") is set to '1' when the valve opened and "Valve Black" ("VB")

is set to '1' when the valve closed, simultaneously setting "VR" to '0' again. The status update string can either be set to be received periodically or it can be queried. Each string is provided with a time tick ("T"), provided by the MCU system timer.

The default JSON strings for each of the query functions are as follow:

```
Query Update: {"VR":0,"VB":0,"GV3V3":0,"GV3V4":0,"GV12V":0,"T":67}
Query Settings: {"mV":[0,0,0,0],"RS":[0,0,0,0],"D":[0,0],"WF":[0,0],"S":[0,0],"T":69}
```

Control Service The peripheral structure used to store the command list is used to implement the control service, in Appendix B.6. The control service brings all the peripheral functionality to life by obtaining each the arguments of the peripherals from the peripheral structure and issues the function call. The peripheral functions without a main loop service are also implemented in the control service. The relay control function is a dual state, switch-case which implements the relay number and the relay operating state. The default state of the switch-case will switch all the relays off. The temperatures are emulated with the use of an external DAC chip, MCP4728, controlled via I2C. The chip also provides an on-board EEPROM, allowing set temperatures to be maintained after it has been specified. The I2C protocol implemented was a third party contribution dedicated to the specific chip, although some modifications were made to allow full functionality on the MCU. A voltage 'setter' function is implemented to allow the control service to easily set the specified voltage levels. Since only pin states are changed and maintained, a main loop service is not necessary to be implemented for the relay control and temperature emulation functions.

4.4.3.3 Firmware Implementation

The test bench hardware was tested while implementing the firmware design, the firmware peripheral functions listed in Appendix B.3 were tested individually to ensure the requested output was received. The test bench completed trial test procedures successfully on a DUT as indicated in Figure 4.8. Therefore, the design was approved for implementation of the complete test procedure. The test bench provides a simple command line type of interface, with the custom communication protocol, allowing quick integration to external interfaces. A test rig can easily be connected to the connection header brought out on the PCB.

Peripherals Test bench, MK 2, complies to the HAT standard for a RP, allowing direct integration with the RP via the extended header pins. These pins provide the required 3.3 V and 5 V operating voltages, a communication port to the test bench operation protocol, the required pins to program the DUT with the required firmware and two push buttons available for possible test execution or interruption functionality. Additionally, the test bench comes with an external micro-USB connection, providing external power, 5 V regulated to 3.3 V, and proving as a communication port. The MCU on-board the test bench provides a 12 bit ADC and the required pins for the pulse emulation and relay control. The ADC pins allow 3 voltage inputs, voltages are 3.3 V, 3.8V and 12V input. An external 12 bit DAC, controlled with I2C from the MCU, is used to emulate the DUT temperature sensors. The water flow emulator provides a two-pin connector with a resistance, 10 Ω , equal to the internal resistance of the reed switch, used by the SC, making the connection identical as to the actual connection from the installed water meter. The water flow emulator can produce pulses up to 20 Hz . The leakage emulator

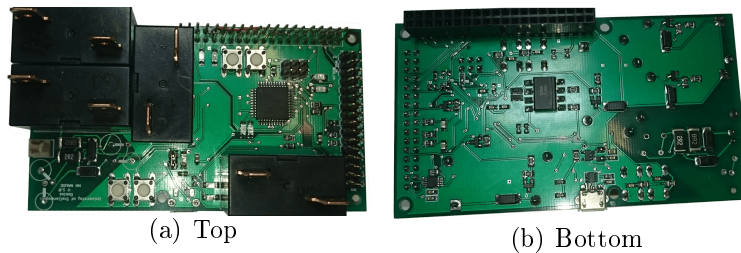


Figure 4.11: Developed Test Bench MK 2

design is similar to the water flow emulator, only with a larger load resistance of $100\ \Omega$ to simulate the water's actual resistance. Four relays are provided to manage the power supplied to the DUT as well as control the loads dedicated to the DUT. The valve sense circuit provides an additional connection point, since the circuit is isolated from the rest of the low voltage circuits. The test bench provides feedback, *Status Update*, every 15 seconds at default, the feedback contains the 3 measured voltages from the ADC pins and the valve sense readings. The test bench is fully configurable from either UART connection and the feedback response rate is configurable, the *System Settings* provides the configured peripheral settings.

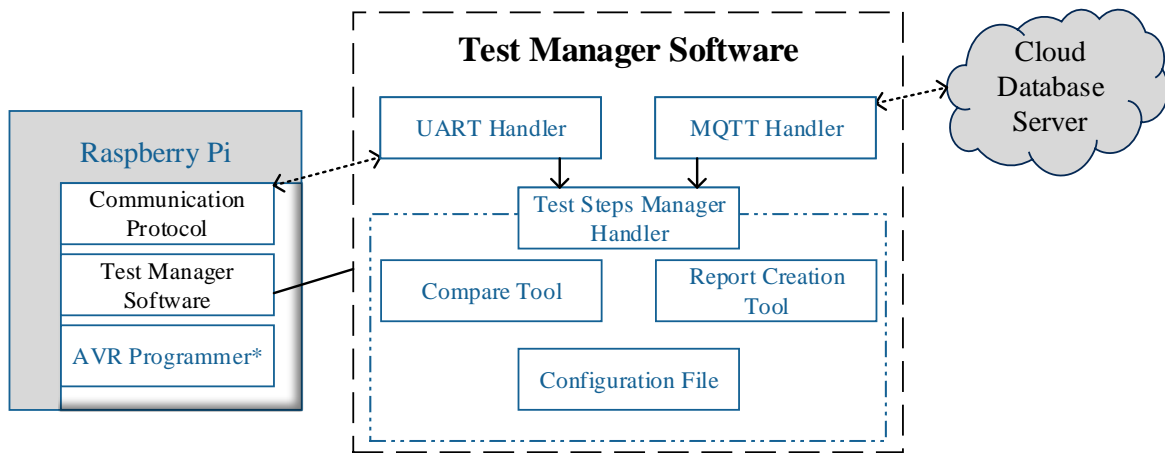
Test Bench MK 2 Integration Requirements The MK 2 design requires a UART or USB connection for communication to the device, at a required baud rate of 115200. The communication protocol is case-sensitive and requires a newline character, '\n', for instantaneous communication response. The test bench requires 5 V to use the relays, the rest of the board can run on 3.3 V.

Raspberry Pi Setup The UART communication needs to be enabled on the RP from the rasp-config panel, allowing sending and receiving of communication via the extended header pins. The RP also requires installation of a programming tool, AVRDUDE, to enable programming of the AVR MCU via the RP extended header pins [62]. AVRDUDE requires a pin selection setup, where the dedicated pins on the test bench need to be selected in the configuration file of AVRDUDE. Once AVRDUDE is configured, the programming tool can be initiated from the command line on the RP.

4.4.3.4 Test Bench MK 2 Summary

A hardware solution, depicted in Figure 4.11, capable of measuring and emulating all the required signals stipulated in Table 4.3 was designed and implemented. The test bench is capable of integrating with external systems, allowing the test system to implement a test procedure. A user interface is provided to allow the user to interact with the test system to initiate and manage the test procedure with the implementation of push buttons.

The RP connected to the test bench can manage all the peripherals with the designed serial communication protocol, also the system update and -settings can be queried at any time. The test bench can be implemented with the use of a micro-USB, allowing more computing platforms to interact with the test bench which will lead to improved development times and quicker set up times for new test procedures.



*AVR Programmer interface via USB to PDI pins of the SC

Figure 4.12: Test Manager Software Integration In The Test System

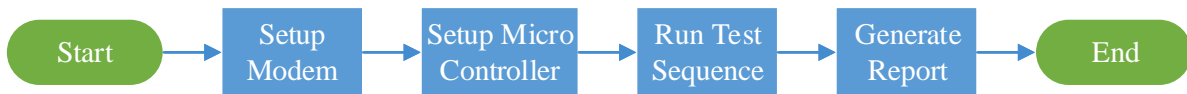


Figure 4.13: Test Software Process Flow

4.5 Test Manager Software

The software used for implementing the test system consists of various processes, as depicted in Figure 4.12, happening simultaneously allowing the software to consist of modular handlers used to maintain specific processes whilst a main test handler controls the test procedure. The two prominent handlers maintain communication via a *UART handler* and the other communicates subscribe and publish commands via the *MQTT handler*. The handler dependent of the two mentioned communication handlers are the *Test Steps Manager Handler*, the main purpose of this handler is to bring together the test configuration, the communication and the data management. The test manager software was designed with the purpose to allow the test bench to simply co-operate with the defined test procedure by either actuating on demand or providing the necessary measurement at the appropriate time.

4.5.1 Software Design

The software used in the test software process flow, depicted in Figure 4.13, was designed and written in the programming language Python 3 which is an OOP language. The design choice was made since Python is a dynamic scripting language, allowing the user to integrate with multiple platforms with ease. This makes Python ideal for prototyping and implementing the minimal viable product required to complete the task at hand. Due to the fact that Python is very versatile, it comes with the draw back of not being a high performance processing language. In the scope of this thesis project, high performance processing power and time critical tasks are not a priority since response times of the SC is relatively long compared to computing time.

Each handler, in Figure 4.12, was created as an class, allowing creation of class instances capable of running in separate threads. By running each handler in a separate thread, a harmony is reached in the test system by allowing each handler to independently handle their own set of tasks without interrupting each other, a UML class diagram can be found in Appendix C.1.

UART Handler - Communication The UART handler is responsible for handling any communication received or sent via the serial communication channel (UART) connected to the DUT. The handler requires a device name to connect to, as well as the set baud rate at which the device communicates. The handler ensures complete data packets are received by inspecting the incoming messages for either a newline (`\n`) or carriage return (`\r`) character. The serial read and transmit functions can be thread blocking, but it is prevented by using read and write time-outs with the serial connection. The message handling allows the UART handler to organise the received messages into two groups, the JSON structured messages are sorted into a status reports group and all the other incoming messages are sorted into a debugging messages group. The messages are stored in a First-In-Last-Out (FILO) stack allowing the handler to provide the latest received message. The latest message is requested via the respective message request methods for each stack. The UART handler makes primarily use of the PySerial package [63].

MQTT Handler - Communication All the cloud communication is done through a MQTT broker, to which receiving and sending messages are done by subscribing and publishing messages to specified broker topics, respectively. The MQTT handler connects to the broker address and provides the broker with the necessary details to be able to connect. Once connection is made to the broker, the handler subscribes to the respective topics to be able receive the messages sent from the DUT. The handler inspects each received message and adds it into a JSON format to ensure the handler adds the received message to the FILO stack. Since the MQTT communication is cellular (mobile) data dependent, no debugging messages are to be expected from the subscribed topics. The latest received message can be retrieved from the handler via the provided request message method in the MQTT handler class. Sending messages to the DUT is easily managed from the handler by issuing a publish message command when the message issue method is called in the MQTT handler class. The MQTT handler makes primarily use of the paho-mqtt package [64].

Test Steps Manager Handler The test steps manager handler comprises of four parts, the test steps manager handler class itself, a configuration file, a comparing tool and finally a report creation tool for the test results. The test steps manager handler class is responsible for initiating the test procedure by creating the UART- and MQTT handler objects, opening up the communication ports to the DUT, also it is responsible for initiating the procedural test steps defined in the configuration file. These test steps defined in the configuration file are predefined by the test engineer.

A test step is defined as a step in the test procedure where a single or multiple procedural commands can be issued. The test procedure is comprised of multiple test steps. The procedural commands reflect to the available functionality functions presented by the test system. The test step setup in the configuration file allows custom test scenarios to be implemented with the use of the configuration file. The configuration file makes use of *step* and *compare* sections. The *step* section provides a means of command

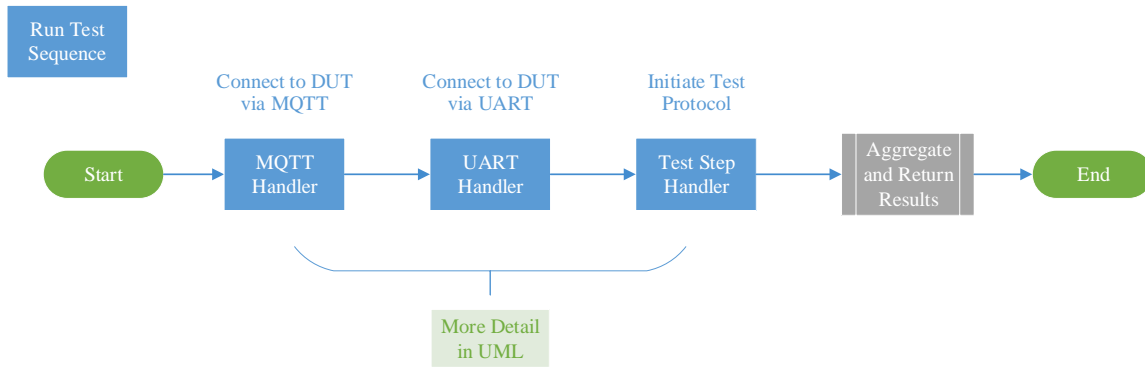


Figure 4.14: Test System Application

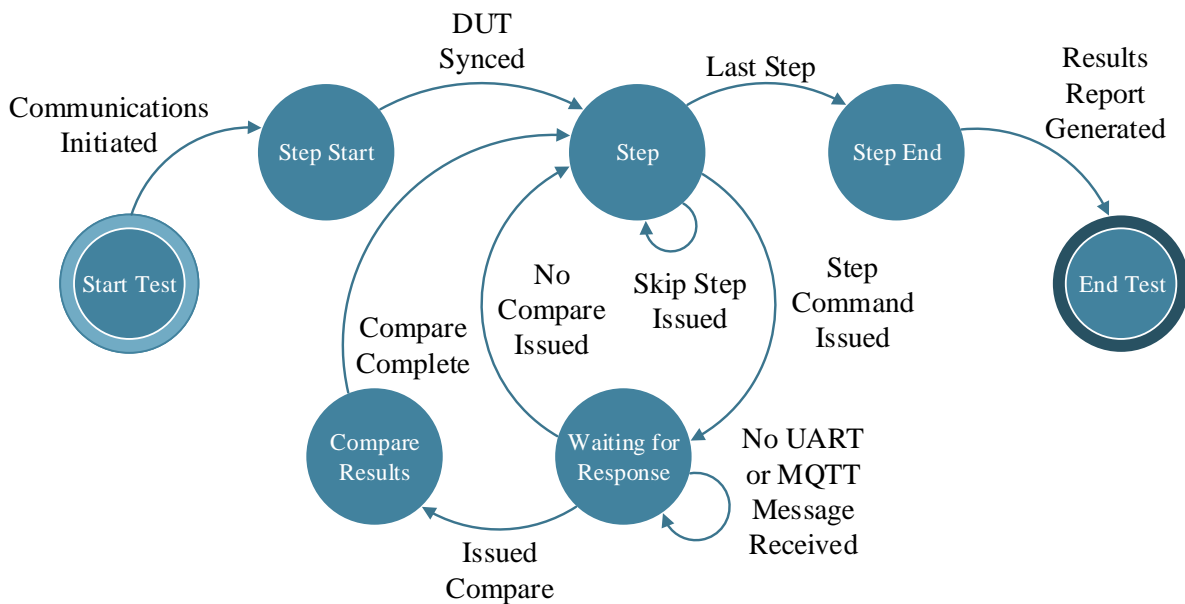


Figure 4.15: Test Step Handler State Machine

issuing, in a step wise manner, while the *compare* section provide a *step* command the option to compare and validate the expected result from the issued command. The compare options provide value comparison and threshold comparison, which are defined in a corresponding *compare* section.

The test steps manager handler implements the state machine depicted in Figure 4.15, where it cycles through the step commands specified by the *step* and *compare* sections in the configuration file, until the final step is reached and the state transitions to the *Step End* state in the state machine. The *Step* state issues the current test step's predefined command list sequentially. Once all the commands have been issued, the state transitions to the *Waiting for Response* state where it waits for a report string either from the UART, MQTT or both. The received report string transitions the state to the *Compare Results* state where the corresponding *compare* command is executed on the received string via the compare tool. The comparison results are stored in a results dictionary in the compare tool object and is retrievable at the end of the test procedure. The *Compare Results* state transitions to the *Step* state where the next *step* commands are issued. Once all the *step*

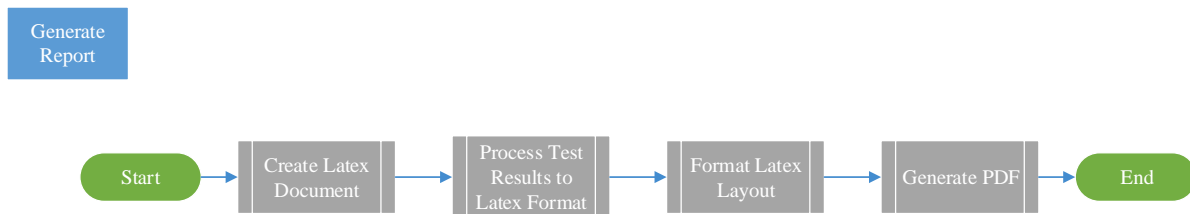


Figure 4.16: Report Creation Flow Diagram

sections are executed and the *step end* section is reached, the test steps manager handler will continue to the *Step End* state where the results dictionary is retrieved from the compare tool and passed to the report creation tool. All the results and DUT details obtained during the test are combined and compiled in a human-readable document by the report creation tool. Finally, the test comes to an end and all the communication ports are safely closed.

Defining A Test Procedure The process of defining a test procedure requires in-depth knowledge of how the DUT functions and how it will react under certain conditions. The test procedure is defined in such a manner to ensure all the required test points and peripherals are tested. There are various methods of testing a specific peripheral or test point, as discussed in Chapter 2. The defined test procedure used for validating and implementing the designed test system is defined as a proof of concept, to show the designed hardware and software functions as intended. The test procedure covers all the SC's peripherals and implements a possible emulation state of an EWH. The configuration file setup provides the functionality to be able to implement various test procedures. Each test procedure makes use of the test software process flow in Figure 4.13, and the configuration file only defines the test sequence in the *Run Test Sequence* process.

Compare And Report Creation Tools The compare tool and the report creation tools provide auxiliary data processing functionality to the designed test system. The compare tool use the configuration file to determine which compare command is executed during a specific test step. The compare tool receives the latest report string along with the current test step and executes the defined comparison on the report string. The comparison indicates with a True or False whether the comparison has passed or failed at the specific step. During the comparison the input data and resultant data is combined to form a results dictionary, saving the entire configuration file along with the results of the data compare.

The report creation tool makes primarily use of the PyLatex package [65]. The input data is wrangled into a table with the use of the Pandas package [66]. The Latex document is created, as depicted in 4.16, where the input data, the results dictionary, is formatted into a presentable format, easy to follow and provides a quick summary of the test results. The results of each compared step is inspected and if any step has failed it is indicated on the summary and the test is classified as failed, as depicted in Appendix D.2. The Latex document can be generated from the report creation tool, if the required Latex compiler and dependences are available on the test platform.

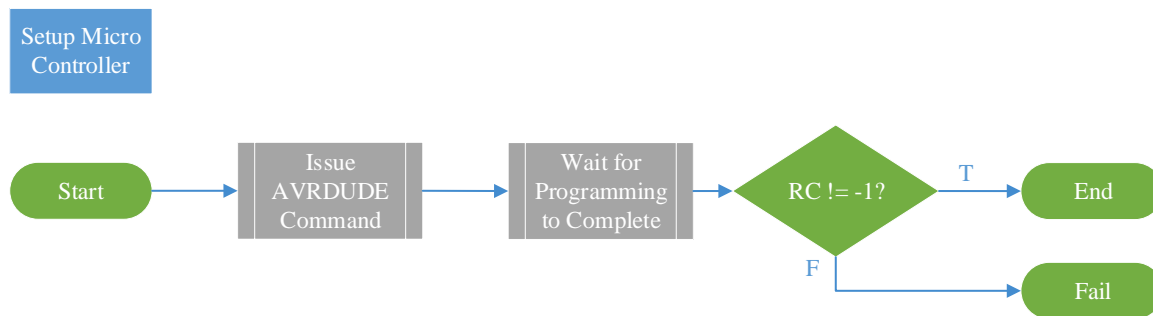


Figure 4.17: Micro Controller Application Setup

Firmware Programming The Raspberry Pi is capable of programming firmware onto the MCU with the use of a Linux package, AVRDUDE [62]. This package allows programming of most Atmel AVR chips, of which the SC is using a compatible chip, without the need of an additional chip programmer. The MCU is directly programmed from the RP header pins connected to the MCU SPI. The SC unfortunately does not provide an interface to the SPI of the MCU, but since Atmel-ICE Debugger Program and Debug Interface (PDI) programmers are available, the RP use AVRDUDE along with the USB connected PDI programmer to program the required firmware onto the SC via its PDI port [67, 68].

This enabled the RP to program the MCU with the appropriate firmware. The RP can then program the required firmware from a programming script as depicted in Figure 4.17.

Modem Setup The modem used in the DUT needs to be set up accordingly, allowing communication between the cloud servers and DUT. This is the first process in the test software process flow, which requires a custom firmware programmed onto the MCU allowing direct communication to the modem chip via the UART of the MCU. The modem responds to AT commands, a specific set of commands are required to set the modem into the correct functioning state ensuring the modem responds correctly to the official SC release firmware [26]. The commands are fed to the modem via serial from the RP with a error checking Python script, known as modem setup, to ensure each command has been received by the modem.

Micro-Controller Setup The MCU setup programs the MCU with the test sequence firmware, once the modem has been successfully set up with the modem setup. The test sequence firmware is an edited version of the official SC release firmware. The changes made in the firmware were to reduce the reporting period from one minute to 10 seconds, allowing a faster response rate from the DUT when the *test step handler* injects signals and state changes into the DUT. Whilst the changes reduce the idle time, it provides sufficient responding time for the DUT modem.

4.6 Test System Summary

The designed test system was implemented with software as depicted in Figure 4.12 and with hardware as depicted in Figure 4.8

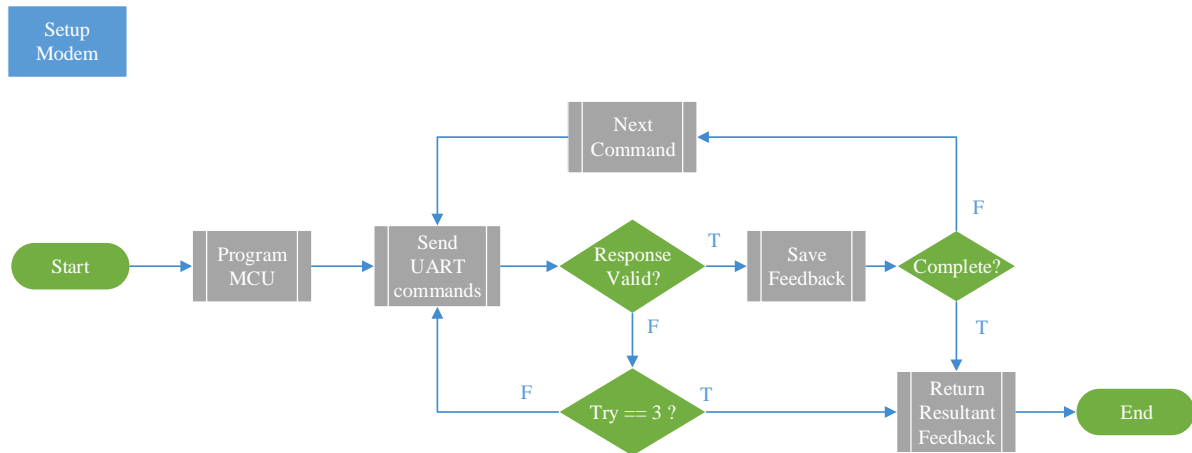


Figure 4.18: Modem Setup Flow Diagram

A complete hardware to software test system was designed, able to communicate to the cloud via MQTT and to the hardware, the DUT and test bench, via UART. The test system provides a test platform, specifically for the SC, where functional test procedures can be combined with in-circuit test point measurements.

The designed software can initiate MQTT queries to the DUT and receive report strings from the cloud based MQTT broker, thus closing the system loop and allowing a complete system test. Automation of the test procedure allows accurate measurements to be compared to the received feedback of the DUT and producing a detailed test report. The test report depicts not only the received test results but also the test environment and test conditions in which the DUT was tested.

The final test procedure depicted in Figure 4.13 ensures a valid quality assurance test can be initiated, provided the test procedure is properly defined. The DUT can undergo various test procedures with the designed test system, but for all intents and purposes of the research design a set of standard tests were specifically chosen to ensure the product meets a certain level of an acceptance level. The defined test procedure used in the research project are orientated to test the functionality of the product as previously covered by the original, manual test procedure.

Chapter 5

Experiment and Results

5.1 Overview

The designed SC firmware is implemented and tested with regards to its accuracy, reliability and how it fared as a deployed firmware in the field.

The test system is implemented and each peripheral circuit is validated against its respective design and SC counter part. The boundaries of the test system test bench is tested and its accuracy is evaluated against the requirements set in Chapter 4. The software manager is implemented and used to test the test system and obtain the results used in this chapter.

The tests carried out in this chapter were done on either a SC MK 3 device, for tests related to the SC firmware, or the test bench MK 2 was used for tests related to the test system.

5.2 Smart Controller Firmware

The firmware functions responsible for measuring, actuating and implementing control are each individually used in various experimental setups to determine each of the firmware function's efficacy.

Design Implementation During the design implementation, the firmware was implemented on a test EWH, remotely actuating cyclic water usage events of different sizes. The firmware was deemed stable after numerous days of running in the test environment, allowing field implementations and retrieving real time data feedback. Needless to say, a couple of iterations were required to produce a stable and reliable firmware version. The firmware reliability mainly depends on a stable connection to the cloud to ensure real-time measurements were made available to the end-user. Factors causing devices to lose connection include data depletion, poor network connectivity and power loss to the SC, detailed communication results can be found in [26].

5.2.1 Firmware Efficacy

Each test experiment setup is described following with a discussion of the obtained results. Any firmware adjustments made for specific experiments will be stated, and is only applicable to the specific experiment.

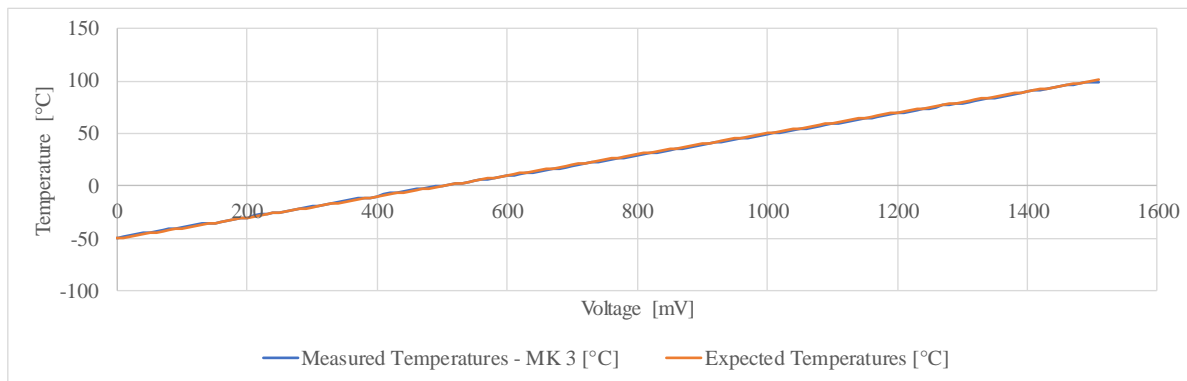


Figure 5.1: Temperature Measurements Read from a Siglent SDG 1005 Signal Generator, Incrementing in Steps of 10 mV, and the Smart Controller Hardware MK 3 Firmware.

Table 5.1: Power Measurement Differences Depicting the Over Estimation Factor of the Smart Controller. The Measurements are Between the Smart Controller Hardware MK 3 Firmware and a TopTronic T98T TrueRMS Clampmeter.

Rated Wattage	% Difference
800	+6.495%
1600	+3.004%
3600	+5.267%

Temperature Readings The temperature readings were taken by inducing 10 mV incremental voltage levels with a Siglent SDG 1005 signal generator. The incremental steps started at zero volts and the voltage was gradually increased to the maximum temperature reading of 99°C at 1.49 V. The report timings were adjusted by a factor of 10, to receive the temperature reading every six seconds.

The measured temperature compared to the expected temperature at the set voltage level is depicted in Figure 5.1. The measured temperatures were well within the 10% accuracy specification, HS[7], listed in 3.2. Minor deviations of one degree Celsius are visible in the measured temperature levels.

Power Measurement The experimental setup for the power measurements consisted of a convection heater with a two-level, 800 W or 1600 W rated, heating setting and a household water boiling kettle, rated 2000 W. These resistive loads were used to emulate the heating element in the EWH. The current was measured with the use of a TopTronic T98T TrueRMS Clampmeter. The current was measured since the voltage is taken as a constant of 230 V in the firmware to calculate the current energy draw of the EWH. The report timings were left unaltered since a running average is taken for each current measurement in the firmware. The measurements are taken over a 10 minute period, producing 10 measurement readings which were averaged and depicted in Figure 5.2.

The measurements show that the measured differences between the MK 3 firmware and the clampmeter are well below the 10% specification. The percentage differences are tabularised in Table 5.1.

Water Flow Measurement The water flow meter has two aspects that need to be considered for testing, firstly the accuracy and secondly the de-bounce design. The ac-

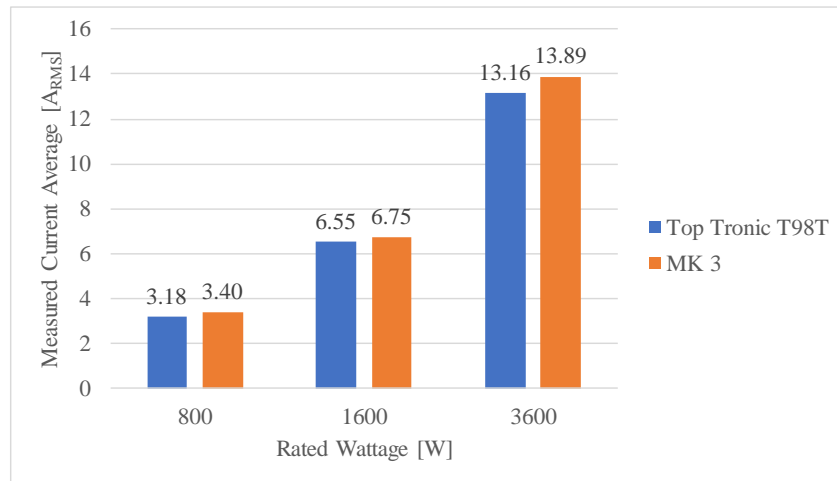


Figure 5.2: Current Measurements Read from a TopTronic T98T TrueRMS Clampmeter and the Smart Controller Hardware MK 3 Firmware.

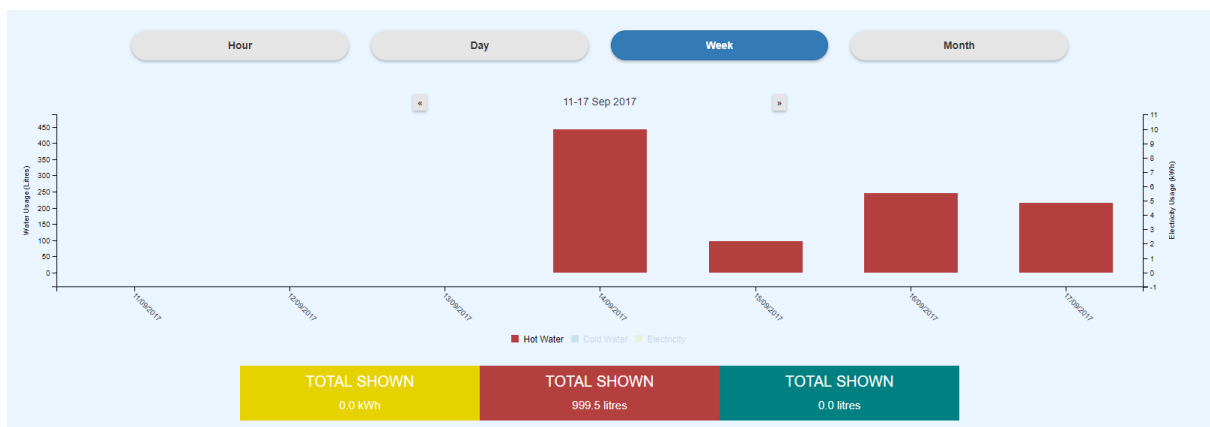


Figure 5.3: Captured Figure from the Web Application of Bridgiot[12]. Total Water Consumed in the Week of Testing. Measured with the Smart Controller Hardware MK 3 Water Measurement Firmware.

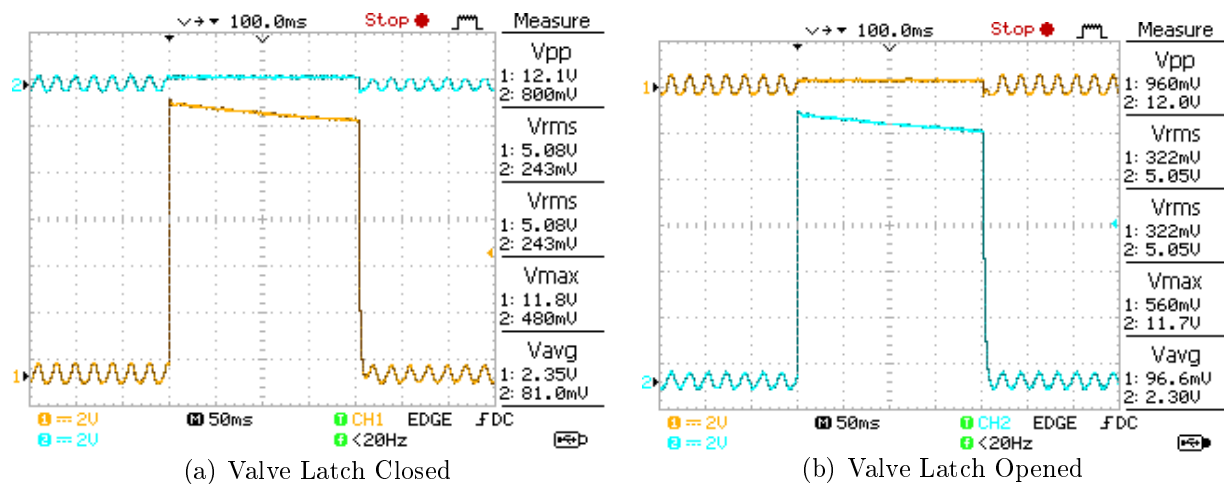
curacy was tested by taking the physical readings on the Elster Kent V110 water meter connected to the SC prior to water events. The test ran for usage events until a total of 1000 litres were measured by the physical water meter. The online web application of the SC was used to calculate the total amount of water consumed, as depicted in Figure 5.3. The designed water flow measurement firmware was edited into the latest version of the SC firmware to allow online monitoring of the total water consumption.

The results were within the 10% specification. The readings on the web application depicted a total of 999.5 litres were consumed. Due to the fact that the water meter has a resolution of 500 ml, only producing two pulses per litre, it is possible that the water meter is on the boundary or close to producing the next pulse.

The water flow measurement firmware makes use of a 100 ms de-bounce timer, which can be seen as a low pass filter designed at a cut-off frequency of 10 Hz. The SC was connected to a Siglent SDG 1005 signal generator, set to produce 20 pulses at set frequencies. The pulses were 3 V with a 50% duty-cycle. The designed firmware was used with the report timer set to report every six seconds. The output of the SC was considered and resulted in Table 5.2. It can clearly be seen that the cut-off frequency is

Table 5.2: A Constant of 10 Litres of Water is Emulated at Different Water Flow Rates and Measured by the Smart Controller Hardware MK 3 Firmware.

Water Flow Rate (l/min)	Measured Litres (l)
30	10
60	10
90	10
120	10
150	10
180	10
210	10
240	10
270	10
300	3.5
330	5

**Figure 5.4:** Shut Off Valve Control of the Smart Controller Hardware MK 3 Firmware.

set at 10 Hz, where some pulses are being ignored by the SC and aliasing is occurring at the 300 l/min rate. The maximum flow rate the firmware can detect reliable is 270 l/min.

Shut Off Valve The shut off valve control, as depicted in Figure 5.4, was measured by inducing a leakage event. The leak detection wires were short-circuited to induce a leak to measure the close pulse required to switch the valve to the closed state. The valve open pulse was measured when the SC was switched off and on again. The expected pulse durations were measured and are depicted in Figures 5.4(b) and 5.4(a), respectively.

Leak Detection The leak detection relies on a threshold calculation adding and subtracting the threshold value depending on whether the threshold has been reached. The threshold functionality is tested by inducing an inverted pulse signal waveform to the leak detection wires of the SC. The pulse waveform is set to $3 V_{\text{peak-peak}}$ with a duty-cycle of 30% at an offset of $1.6 V_{\text{DC}}$. The designed firmware sets the threshold voltage of the leak detection at one volt and the threshold count to 20. The leak detection is serviced every 100 ms, which means if the voltage on leak detection wires falls below the one volt

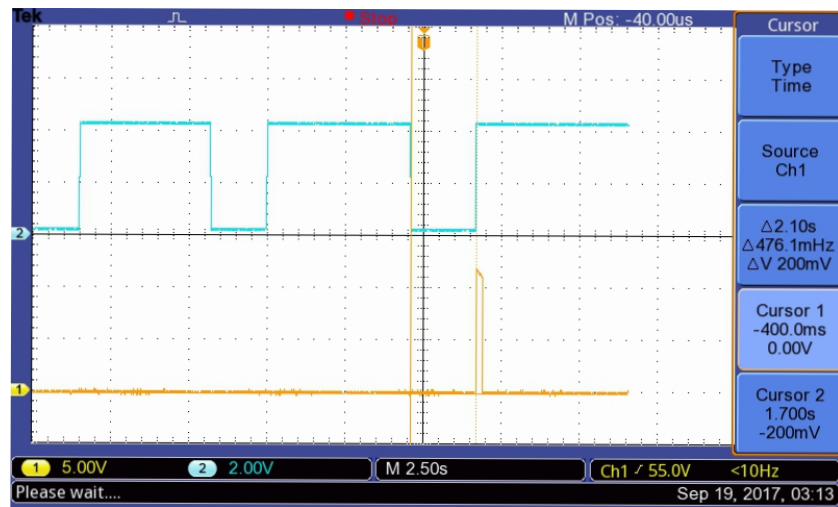


Figure 5.5: Leak Detection Triggered on the Smart Controller Hardware MK 3 Firmware with a Pulse Waveform.

threshold for two seconds or longer, the "Burst Protection" state is activated which is when the water shut off valve is closed.

The leakage detection triggered the valve to latch when the pulse waveform stayed low for 2.1 seconds, as depicted in Figure 5.5.

5.2.2 Firmware Validation

The designed firmware ensured readily available and accurate data to be gathered for research projects based on EWH data. The data have already been used in multiple completed or ongoing projects [15], [19] and [26].

Field Deployment Concerns The majority of SC devices were deployed in the eMkhondo district in Mpumalanga, South Africa. The firmware deemed to be reliable hence the installation, region and operator difficulties. The installation difficulties implies that most installations were done on houses not complying to the SANS EWH installation standards, which lead to extra installation requirements to ensure the device is installed on a SANS approved EWH installation. Majority of the EWHs were not earthed properly, putting the SC functional operation at possible risks.

The region proved to be tough to do installations since the region's network infrastructure was at first not fully equipped to handle the data transmission load of the installed devices, which led to multiple packet losses and overall poor network connectivity [26]. Since the hardware and firmware development is done in the Western Cape, Stellenbosch, the servicing of installed devices were logistically impractical and fault finding lagged with the customer complaint. Additionally, a major problem with most of the installations was not the installation itself, but rather the client receiving the SC. The client was not technically inclined to log into a web-based application and configure the schedule for the EWH.

Data Analysis The data received from the installed devices allowed modelling and analysis software to experiment with different scheduling techniques. Although the data

was not perfectly complete, data cleaning and filling techniques proved to recover most of the missing data.

The SC was used to validate an EWH simulation model used in [15]. The model provided a window into how water events effect the energy input in the EWH and provides insight to determine energy efficient water heating schedules.

Another research paper used the data gathered, by the SC, to receive insights on how EWH water usage patterns differ over weekdays and weekends and how it compares different geographical regions, namely Cape Town, Western Cape, and Mkhondo, Mpumalanga [19].

5.2.3 Findings

The firmware proved to be reliable, together with the communication firmware, and provide accurate measurements. During the development of the SC firmware, problems with memory corruption and clashing interrupt service routines were dealt with. The memory corruption was due to buffer overflows in the status report string generation. The clashing interrupts were caused by a analog sweep on all the analog input pins, and when water usage events occurred the interrupt priorities were the same. The analog sweep method of reading the analog input pins was replaced by the analog read function, reading a single analog pin when the function is executed.

The SC proved to provide access to one's EWH with the ease of signing in on a web application from any smart communication device. The data gathered was of such a standard that research could be done on optimisation of heating schedules and provide a means to create awareness of a household's EWH consumption patterns [15, 19].

5.3 Test System

The test system results comprise three main aspects, the test bench hardware, the managing software and an application of the designed hardware and software as a test system. Each aspect is examined and verified to the intended design, respectively.

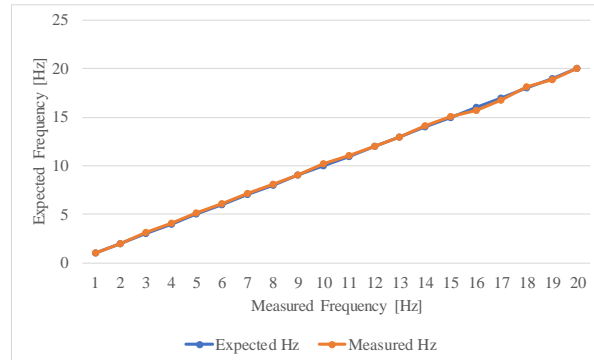
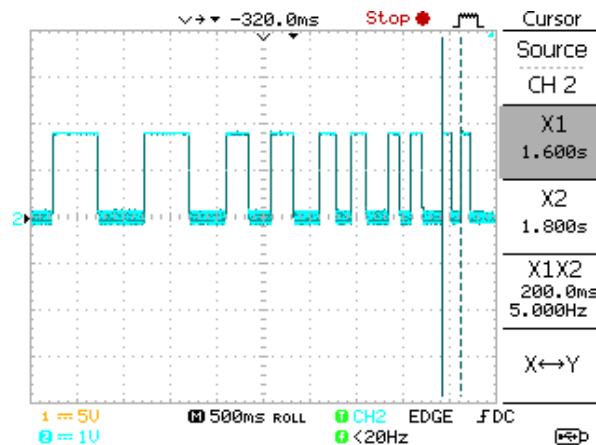
5.3.1 Test Bench Hardware Efficacy

An experimental setup for each of the hardware peripherals, test bench MK 2, is explained followed by the results and findings of each experiment. The experimental setups are used to measure and validate the accuracy of each peripheral.

Voltage Measurements The analog voltage measurement functionality of the test bench was measured against an ISO-Tech oscilloscope, model ID 6052-U. The voltage dividers scaled the measured voltages, therefore the input voltage was used to determine the expected measured voltage. The measurements were taken on the input of the two different voltage divider circuits, where the input voltage was incrementally increased from zero to 3.3 V and 12 V respectively. The test bench update report string was considered, where the measured voltages were reported in millivolts. The percentage difference, depicted in Table 5.3, of all the measured voltages proved to be accurate enough for the intended application of the voltage measurements peripherals. The 3.3 V measurements achieved a +0.53% difference from the calculated expected voltages, where the 12 V measurements achieved a -1.23% difference.

Table 5.3: The Percentage Difference of the Voltage Dividers used in the Test Bench MK 2.

Dividers	% Difference
12 V Input	-1.23%
3.3 V Input	+0.53%

**Figure 5.6:** Frequency Measurements of the Water Flow Pulsar.**Figure 5.7:** Water Flow Pulsar Producing Two Pulses per Frequency in the Range, One to Five Hertz.

Water Flow Pulsar The water flow pulser was connected to the ISO-Tech oscilloscope, to measure the expected frequency set with the communication protocol of the test bench. The water flow emulation function in Table 4.6 was used where the frequency is incremented gradually until the test bench reached its pulse producing limit. The amount of pulses were kept to two.

The frequency ranges considered ranged from zero to 20, these frequencies were achieved with minor deviations from the expected frequencies depicted in Figure 5.6. A finer inspection of the typical operating frequencies of the water flow meter is depicted in Figure 5.7, where a frequency sweep is done with the pulser, from zero to five Hertz for two pulses per set frequency.

Voltage Emulation The temperature sensors used by the SC increments the measured temperatures in steps of 10 mV/°C. Therefore the voltage levels were incremented every second with 10 mV, ranging from zero to 3.3 V. The set voltage levels function in Table

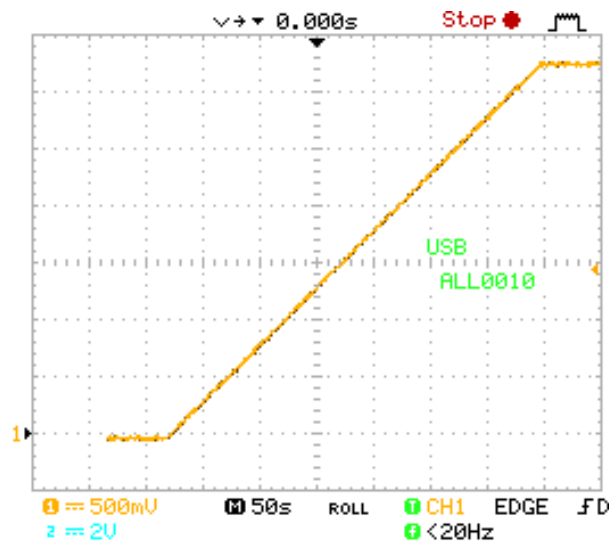


Figure 5.8: Voltage Measurements from Test Bench MK 2 Digital to Analog Sensor.

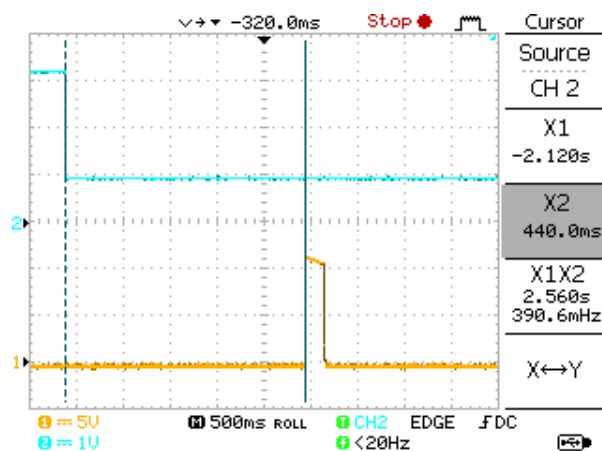


Figure 5.9: Leakage Emulation Triggering Shut Off Valve.

4.6 was used to adjust the DAC to the specified voltage level. The output voltages, depicted in Figure 5.8 produced a linear increase of voltage as expected.

The linear outcome of the voltage emulation, allows the SC to read the temperatures, as indicated in Figure 5.1, proving the voltage emulation suffice in providing valid temperature sensor output voltages for the SC to measure.

Leakage Emulation The leakage emulation are used to trigger the shut off valve, thus a short-circuit is induced on the leakage detection wires. The leakage event is triggered with the use of the leakage emulation function in the communication protocol listed in Table 4.6. The duration was set to maximum and the elapsed time to trigger was measured. The triggered valve can be seen in Figure 5.9, where the time to trigger is 2.560 ms.

Valve Sense The valve sense circuit in MK 2 captured the valve open and close states and produces a logic high for each state dedicated pin of the MCU. The result is clearly depicted in Figure 5.10, where the valve close state is captured, the same result occurs for the valve open state.

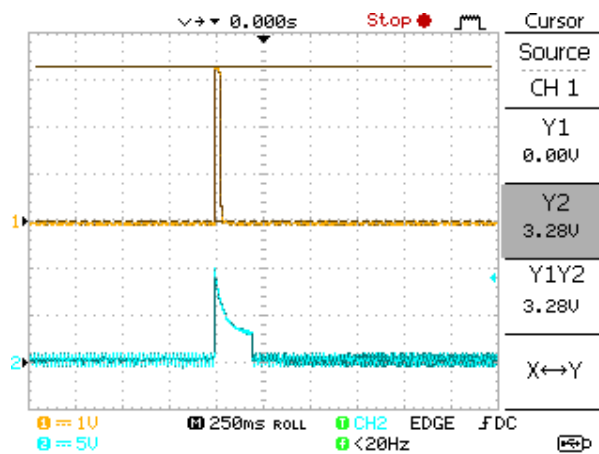


Figure 5.10: Valve Sense Logic Output from Valve Closing

```

pi@raspberrypi:~$ python3 serial_read.py /dev/serial0 115200
b'{"VR":0,"VB":0,"GV3V3":0,"GV3V4":0,"GV12V":0,"T":67}\r\n'
b'{"mV":[0,0,0,0],"RS":[0,0,0,0],"D":[0,0],"WF":[0,0],"S":[0,0],"T":69}\r\n'

```

Figure 5.11: Serial Output from Test Bench Buttons

The logic high is measured by the MCU and the status string is updated. The status string provides which state was triggered most recently, this can then be interpreted from any of the communication channels, UART or USB.

User Interface The user interface to the test bench consists of four buttons, split into two pairs. The first pair is dedicated to the MCU, where any of the firmware communication protocol functions can be dedicated to a button. The protocol functions used during developing were the status- and update query functions. The second pair is made available to the connected RP to initiate or stop test procedure scripts.

Summary The peripherals provided adequate accuracy and resolution for interacting with the SC. The peripherals successfully emulate an EWH by providing the SC with the expected signals at each of the SC's input peripherals. The test bench also provides communication via UART or USB of all the measurable peripherals, which are the valve sense and voltage measurement functionalities. The test bench provides an interactive request and respond platform with the communication protocol, allowing external managing software to easily interface with each of the peripherals.

5.3.2 Test Manager Software

The manager software is responsible for interpreting the test procedure, set by the user, from the configuration file and executing the user defined test procedure. The test procedure is broken up into test steps, allowing the software to compare each step to the pre-defined expected result. Finally, the test results are aggregated and presented in a human-readable format. The test manager software comprise software handlers, a configuration file and helper methods. The manager software is dependent on communications to either the SC's cloud or the test bench.

Software Handlers The software handlers consists of three handlers, namely; *UART Communication*, *MQTT Communication* and *Test Steps Manager*. The communication handlers are each a class of their own, independent of any other handlers. Even though the test steps manager is a class of its own, it makes use of the communication handlers since it is the main class managing the test procedure execution, result comparison and -generation. The integration of the three classes explained in this section is depicted in Appendix C.1.

UART Communication The UART communication class provides a user with communication setup via a serial connection. The messages received are verified whether it is of a JSON format since it is the chosen notation for communication. The class also provides a send command method, where the user can prompt for a provided message to be sent to the connected serial device.

The UART communication class requires the serial device port name and baud rate when the class instance is created and appended to an object. The created object has a subclass of threading, which allows the object to run in its own thread. The threading is started with its 'start()' method. The object continuously monitors the in-bound communications and add the valid JSON strings to a Last-In-First-Out (LIFO) queue, always providing the latest message first. An out-bound message prompt raises a notification flag, notifying the in-bound message checker to pause and write the output to the serial port before it continues to check again for in-bound messages.

The message queue is accessible via 'getter' methods, which return the latest available message in the queue. The out-bound message can be sent via the 'setter' method by providing the message in string format as argument to the send command method.

MQTT Communication Similar to the UART communications, a class for the MQTT communications was created with the subclass of threading. The MQTT handler makes use of a Python MQTT package, paho-mqtt. This package provides a connect, subscribe, message loop checker and a message publish methods. The MQTT protocol requires a client to subscribe to specific topics, which are defined in the MQTT broker. These topics are predefined by the SC's communication protocol design. The MQTT handler connects to the broker and subscribes to all the required topics, these topics need to be serviced by the message loop checker to be able to receive the latest message published to the topic. The message publish method is used to publish queries to the specific SC subscribed to. Each SC has a unique identifier linked to the specific topic name when it subscribes to the MQTT broker.

The unique identifier used by the SC communication protocol is each SC's International Mobile Equipment Identity (IMEI) number, thus providing a method of ensuring connection to a specific topic is device specific and connection to the wrong device is

minimised. The IMEI of each SC is obtained by prompting the SC's modem via UART communication, each modem is also required to be setup by issuing AT-commands to it. The process of gaining the IMEI and setting the modem to the correct operating settings are done from a helper function provided by [12]. This particular helper function is referred to the modem setup script. The script was modified to retrieve the SC's IMEI.

The MQTT communication class requires the SC's IMEI to be able to subscribe and publish to the correct SC during the test procedure. Once the instance is created and appended to an object and the threading is started, the object firstly connects to the broker and once it is connected it subscribes to all the topics. The message loop checker runs continuously, checking the JSON string and adding it to the LIFO queue. The messages not adhering to the JSON format are considered and added to the JSON dictionary if it was a reply from the SC regarding the set-point, airtime balance, firmware version, signal strength or SC uptime. The publish command is used to query these possible replies from the SC, the publish command notifies the message loop checker to pause and publish the specified command to the related topic before it continuous to check the subscribed topics for new messages.

Due to the nature of specific topics for specific publishable commands, methods were implemented for each of the available queries to the SC. The latest received message can be retrieved from a *'getter'* function, which returns a JSON dictionary.

Configuration File A configuration file, using the INI format, is used to define a test procedure, which consists of test steps and compare steps. Each of the steps are defined as sections, where each of the step sections contains a function which are defined by the user [69].

The functions used in the configuration file are defined in a section called, *FUNCTION_LIST*, which is defined in Table 5.4. These functions serve two purposes, firstly, it links the required function to be executed to a specific test step. Providing a means of setting a test procedure in place by adding functions to the sections, *step* and *compare*. In essence, an user friendly way was found to set up a test procedure without having to alter the source code. The second purpose for the functions are to create an interface between the user defined test procedure and the test procedure handler, namely the test steps manager.

The compare step has two compare types, *VALS* and *THRES* as listed in Table 5.4. These compare types refer to a value comparison to see if the received result is equal to the set value or a threshold range is provided wherein the resultant value is still valid if it is within the set threshold. Since the INI format nor the Python language does not make use of type definers, the *VALS* compare type can consist of numerical values or a character string. The *THRES* compare type is only valid when the *VALS* compare type is numerical, the *THRES* compare type will compare the resultant measured output against the set values defined in the related *step* section.

The configuration file also provides a means to edit the expected JSON format key-value pairs listed in Table 5.6. This is achieved by editing the *DEFAULT_JSON* section, this section also provides default values to the reported string from the SC. In this system, the key-value pairs not being reported is assumed to be either zero, open or off, as listed in Table 5.6.

Combined with the *DEFAULT_JSON* section, is the *DEFAULT_COMPARE_SETTINGS* section, tabularised in Table 5.7. This section links the compare function listed in the *compare* step to a JSON key, to ensure the same keys are compared to one

Table 5.4: Definition of the Configuration File, *FUNCTION_LIST* Section, Functions.

<i>FUNCTION_LIST</i>	Definition
SET_TEMP = [x,x,x,x]	Set each voltage emulation channel, where x is the temperature value with the configuration [T1, T3, T2, T4]
SET_RELAYS = [x,x,x,x]	Switch the relays, where x is enable, 0 is off and 1 is on with the configuration [Load 1, Load 2, Live, Neutral]
SET_DRIP = [x,y]	Sets the leakage emulator, where x is enable, 0 is off and 1 is on and y is the duration in seconds.
SET_WF = [x,y]	Sets the water flow pulser, where x the frequency and y is the amount of pulses.
SET_RT = [x,y]	Sets the update report time of the test bench, where x is enable, 0 is off and 1 is on and y is the period in seconds.
M_NUM = [x]	Requests the SC's contact number via the MQTT communications, where x is a place holder.
M_RSSI = [x]	Requests the SC's signal strength via the MQTT communications, where x is a place holder.
M_BAL = [x]	Requests the SC's remaining data balance via the MQTT communications, where x is a place holder.
M_VER = [x]	Requests the firmware version of the SC via the MQTT communications, where x is a place holder.
M_TIME = [x]	Requests the up-time of the SC via the MQTT communications, where x is a place holder.
M_SP = [x]	Sets the set-point via the MQTT communications, where x is the desired set-point.
A_DELAY = [x]	Provides a time delay in the test step, where x is a the amount of delay in seconds.
A_CONTINUE = [x]	Provides a skip to the waiting for response time, where x is a place holder.
G_QU = [x,x,x,x]	Requests an update report from the test bench readings, where x is the expected states to be returned.
VALS = x	A compare type used to compare an expected value to the reported value, where x is a number or a string.
THRES = x	A compare type used in conjunction with VALS to compare a number within a threshold, where x is a number.

Table 5.6: The Default Values Defined in the Configuration File, *DEFAULT_JSON* Section.

Key	Default Value
V	Open
R	Off
W	0
Hm	0
Cm	0

Table 5.7: Configuration File, *DEFAULT_COMPARE_SETTINGS* Section, Used to Bind Received JSON Keys to Functions in the *FUNCTION_LIST* Section for Comparing Results.

Function	Compare Key
SET_TEMP	['T1','T2','T3','T4']
SET_RELAYS	W
SET_DRIP	V
SET_WF	Hm
M_RSSI	ss
M_VER	ver_req
M_TIME	time
M_SP	R
G_QU	['VR','VB','GV3V3','GV3V4','GV12V']

another. This is achieved by defining a key to a function in the *FUNCTION_LIST*. The correct key is then used by the test steps manager when the *compare* step issues a compare to a defined function, which then identifies with the key to be compared with in the JSON report string.

The test procedure is set up with use of *step* and *compare* sections, starting at *step 0* and ending with an empty *step end* section. Each of the sections follow on each other, although every *step* section does not necessarily need a *compare* section. An example of how the configuration file can be used is listed in Listing 5.1

Helper Class The helper class provides methods to ease the data handling in the test steps manager. The comparing class implemented does the comparison handling in-between the test steps, when a result is received it is sent to the comparing class where it is compared to the expected result and stored. The compare class reads in the configuration file and extracts the defined functions in each *compare* section, these functions are used as keys in a dictionary of functions [70]. What this entails is the following, each function, listed in Table 5.7, is bound to a compare function, in the comparing class, which is either a string compare or an integer compare function. The integer compare function makes use of the threshold value set in the compare step, whereas the string compare function compares the received value with the expected value set in the compare step. The compare class extracts the data in the *DEFAULT_COMPARE_SETTINGS* section and adds it to a local compare settings dictionary. After obtaining all the default settings, the class provides two interfacing methods, the first method updates the measured results by receiving the latest report string of the SC and the current test step number. The measured results is stored into a local dictionary under a key: 'measured'. The second method only requires the test step number that needs to be compared, thus the method uses the test step number and compares it to the corresponding compare step function. The compare step function, defined in the configuration file, is extracted and used in the dictionary of functions to ensure the correct comparison function, string compare or integer compare, is used. The comparison functions return either True or False in an event of a pass or fail. These results of the comparison is stored in the same dictionary under the key: 'results'. The final method provided by the comparison class is a 'getter' method, which returns the local results dictionary.

The final helper method used to process the resultant dictionary after a test procedure, is a report generating method. This method aggregates all the received data during

Listing 5.1: Snippet of an Example Test Procedure Configuration File.

```

1  [step 0]
2  SET_RT = [0,10]           ;Switch of reporting
3  SET_RELAYS = [0,0,1,1]   ;Switch Device ON
4  SET_TEMP = [0, 0, 0 ,0] ;Set all temperatures to zero
5
6  [step 1]
7  SET_RELAYS = [1,0,1,1]   ;Switch Load 1 ON
8  M_NUM = [0]              ;Request the contact number
9
10 [compare 1]
11 SET_RELAYS           ;Compare the power reading
12 VALS = 1600          ;The expected value
13 THRES = 300          ;The value is in range 1300 and 1900
14
15 [step 3]
16 ;Set Temperatures (will keep setting since DAC has eeprom)
17 SET_TEMP = [50, 50, 50 ,50]
18 M_RSSI = [0]         ; Request signal strength
19
20 [compare 3]
21 SET_TEMP             ;VALS not necessary
22 THRES = 5            ;Expected values are defined in step 3
23
24 [step end]
25 ;Keep Empty for completion of steps
26
27 ;Test Completed
28

```

the test procedure and uses it to create a Latex file, tabularising all the relevant data into a human-readable format. The Latex file produces a Printable File Format (PDF) with the test results, as depicted in Appendix D.2.

Test Steps Manager The test steps manager makes use of the FSM designed in Chapter 4 which is implemented with the use of a dictionary of methods, where the dictionary keys are bound to the specific states of the FSM and the corresponding value of each key is the 'state method' that needs to be executed. A code-snippet of the implementation of the dictionary of 'state methods' are listed in Listing 5.2. The dictionary of methods allows transitioning to different states with ease by only changing the dictionary key. A second dictionary of methods is used for executing the pre-defined functions at each *step* section in the configuration file. The dictionary keys are directly obtained from the contents of the *FUNCTION_LIST* section in the configuration file. Each key has a corresponding 'command method', which is a method that is executed by the test step manager to either send a command via the UART communication to the test bench, a message to be published via the MQTT broker, to add a delay to the test procedure or skip the waiting state in the FSM. The dictionary of 'state methods' is used to maintain

Listing 5.2: Snippet of the Finite State Machine Implemented in a Dictionary of Methods - 'State Methods'

```

1 # Initialisation in the Test Steps Manager Class
2 self.state = 'start'
3 self.states = { 'start': self.state_start,
4                 'waiting': self.state_waiting,
5                 'compare': self.state_compare,
6                 'step': self.state_step,
7                 'end': self.state_end}
8
9 # Main Loop
10 while(not self.end):
11     try:
12         self.states[self.state]() # Execute state until new self.state
13

```

the FSM, whereas the dictionary of 'command methods' is used to execute the hardware peripheral commands, the MQTT commands or procedural commands.

The test steps manager class has a subclass of threading, which is used to manage the timings of the test procedure. The test steps manager requires the SC's IMEI number, the serial port name of the test bench and its baud rate. Additionally, the configuration file, tester identification and test type is required. The test steps manager reads in the configuration file and extracts all the sections from the file and pass the file to the comparing class. The test steps manager creates a instance for each of the MQTT handler and UART handler and starts their threads. The test steps manager's thread also requires the threading start method to be called, this is called when the communications are established to the MQTT handler and UART handler. As soon as the test steps manager is started it follows the FSM, as depicted in Figure 4.15, with the 'state methods' and executes the required 'command methods' acquired from the *step* functions. In the 'step' state of the FSM, all the items in the current *step* section is retrieved and each item, which is a pre-defined command, is sent as a key to the dictionary of 'command methods'. The item consists of a key-value pair, where the key is the pre-defined function name and the value the pre-defined setting for the function, thus the value is sent as argument to the dictionary of methods.

The test steps manager makes use of the 'command methods' to interact with each of the other handlers and apart from the helper class responsible for managing the data, a logging system is also used to store all the received messages and executed commands in a log file. The test procedure will continue with the FSM until the 'end' *step* section is reached, once it is reached the FSM's end state is executed. The end state method acquires all the data from the comparing class and sends the data to the report creating method. The Latex file is created and stored locally, the test steps manager stops all the other handlers and when the test procedure completes it returns a dictionary with all the results as well.

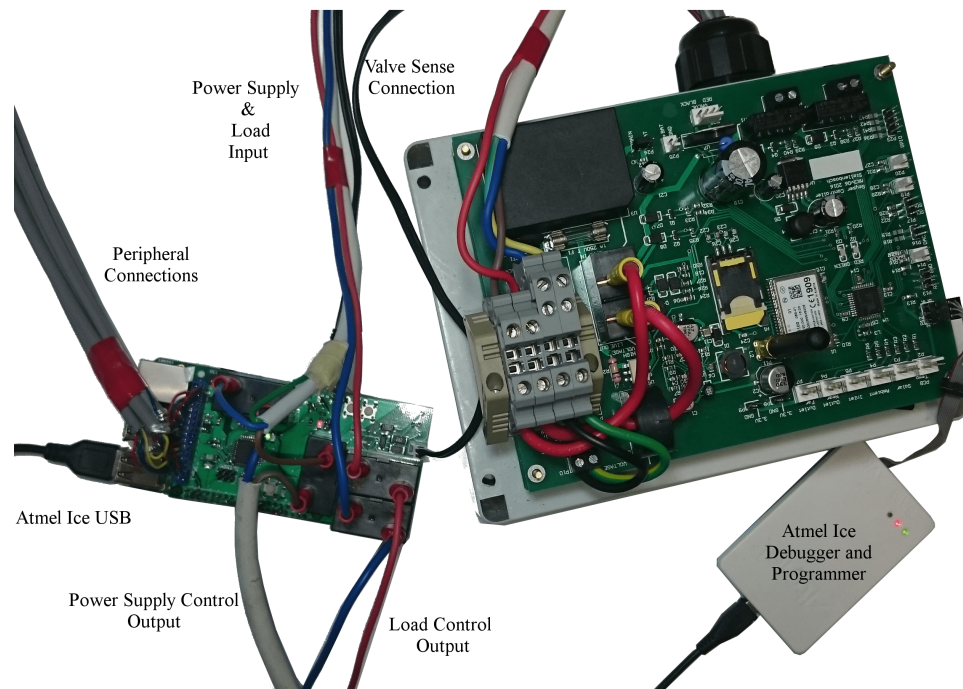


Figure 5.12: Described Test System Setup with Test Bench MK 2 and the Built Test Rig.

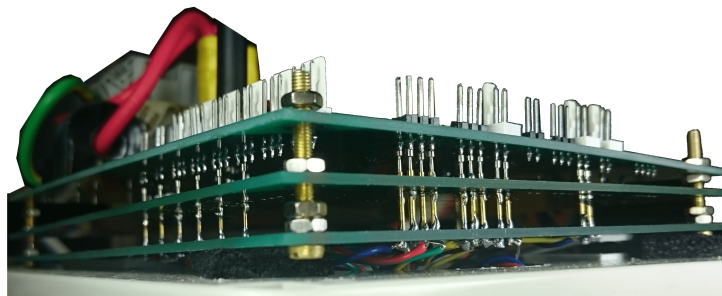


Figure 5.13: Pogo Pins Used in the Built Test Rig.

5.3.3 System Application

A rudimentary test platform was built, depicted in Figure 5.12, with the use of unpopulated SC PCBs and pogo test pins, depicted in Figure 5.13. The built rig proved to be a bed of nails type test rig. The test rig was used to provide a convenient way of testing multiple SC PCBAs. The built test rig is used to apply the designed test procedure with the use of a RP and the designed MK 2 test bench.

Trial Test Procedure The test procedure is implemented with the use of the functions provided in the configuration file to create a trial test procedure. The threshold and expected values defined in the trial test procedure are obtained by initially running the test procedure and see what the outcome of the test is. Minor adjustments to the *THRES* and *VALS* compare types for each applicable comparison can be made to ensure the intended trial test procedure setup is done correctly and will produce the intended test results.

The trial test procedure used to validate the system functionality is listed in Appendix D.1. The test procedure is implemented on a RP, where all the test procedure

process flow steps, depicted in Figure 4.13, are set up into a single Python executable procedural script.

The procedural script issues a command to AVRDUDE to program the SC's MCU with the modem forwarding firmware, allowing direct serial communication to the on-board modem of the SC. The unique device identifier, the IMEI, is extracted from the modem forward setup script, which is used in the next step of the process flow. The procedural script commands AVRDUDE to program the SC with the test procedure firmware. The test procedure firmware is a duplicate of the designed firmware in Chapter 3 with changes made to the reporting times, which is decreased to report every 10 seconds instead of every minute. Once the test procedure firmware is programmed, the IMEI is passed to the test steps manager handler. The test steps manager pass the IMEI to the MQTT communications handler to allow subscription to the correct DUT on the MQTT broker. The UART communications handler connection requirements are predefined, along with the configuration file used to define the trial test procedure. This is done since the connection properties and test procedure stay the same for test conducted, although the configuration file can easily be changed in the procedural script in case alternate test procedures are required to run. The test steps manager handler iterates through all the trial test procedure's defined test steps and comparison sections. Finally, the resultant data returned and passed to the report creation tool responsible for generating a locally stored Latex file.

Test Results Report The report generation tool use the resultant data from the test procedure and produce a Latex file which can be produced into a compilation script, compiling all the Latex files of the tested SCs. The resultant PDF of a SC tested with the trial test procedure is attached in Appendix D.2. A test summary is provided on the first page, stating whether the DUT passed or failed the test. In the event of a failed test, all the failed test steps along with the functions compared at each failed step is tabulated. Each SCs modem details are tabulated to ensure device identification is possible when test reports are being reviewed. Finally, a complete set of test results are tabulated. The tabulated results list all the steps executed from the configuration file along with the predefined requested step values. The compare sections of the configuration file is listed for each of the corresponding compared step, along with which functionality is tested and the defined test and its boundary parameters per compare step. The measured column use the compare keys defined in Table 5.7 to compare the specific SC peripheral and extract the corresponding measured key's data from the resultant test data obtained from the completed test procedure. The results column use the same compare keys listed in Table 5.7 to produce a test score indicating whether the measured values have passed or failed the pre-defined boundary conditions. The '-1' result in the score column is an indicator for values divided by zero during the comparison stage of the test procedure, these scores do not affect the overall test result since the default value for test values are assumed to be zero.

5.3.4 Validation

The test system as a whole is validated by implementing the trial test procedure on multiple SCs. The validation test consisted of 10 devices picked at random. The test firmware used to test each of the devices were altered to report in every six seconds. A summary of the test procedure used is depicted in Table 5.8 where each step and the

Table 5.8: Validation Test Procedure Summary.

Step #	Compare Command	Expected Result	Threshold
0	g_qu	[1, 0, 1500, 0, 0]	500
1	set_relays	1500	200
2	set_temp	[5, 15, 35, 55]	5
3	set_wf	30	10
4	set_drip	Closed	
5	m_sp	Off	
6	g_qu	[0, 1, 1500, 0, 0]	500
7	N/A		
8	set_drip	Open	
9	m_sp	On	
10	g_qu	[1, 0, 1500, 0, 0]	

expected results are tabularised. The results obtained from the test procedures were aggregated and tabularised in Table 5.9.

Validation Results The results from the 10 devices, as depicted in Table 5.9, showed that six devices passed the test with no failures detected. Four devices failed, each failure is described as follows.

- Device 5 did not meet the expected temperature of 55°C and was below the threshold of 5°C, instead the measured result was 48°C. All the other test steps passed.
- Device 7 failed to produce any of the expected temperatures, but measured zero degree Celsius on all the temperature sensor pins. It also failed to produce any water flow measurements. It also failed to measure the opening of the shut-off valve in step 10.
- Device 8 failed to measure the shut-off close valve in step 6.
- Device 9 failed to measure the wattage in step 1.

Results Analysis The devices tested were manually inspected to compare to the validation results. The manual preparing of each device test requires connection of the power supply cables and the load cables with screw down terminals and then the placement of the device on a test platform built for the validation tests. This platform use pogo pins to connect to the respective measuring points and screwed down to connect properly.

With this background, the possible causes for failures are as follows:

- Temperature measurements - The test bench hardware failed to set the correct voltage, the connection between the temperature sensor pin and MCU is faulty or the pogo pins do not connect properly.
- Wattage measurements - The current transformer circuit is faulty, the load is not properly tied down on the screw terminals or the test software compared the incorrect report string.

- Water measurements - The test bench hardware failed to produce pulses, the connection between the water measuring pin and MCU is faulty or the test software compared the incorrect report string.
- Valve sense measurements - The test bench did not measure the pulse, the shut off valve circuit is faulty or the pogo pins do not connect properly.
- The other measurements are strings compared in the test software.

The errors detected in the results, in Table 5.9, are due to the following causes.

- Device 5 - The circuit between the temperature pin and the MCU is faulty.
- Device 7 - The incorrect string was used in the test software for the temperature and water measurements, either the test bench failed to detect the shut off pulse or the test software used the incorrect string.
- Device 8 - The test bench either failed to detect the shut off pulse or the pogo pins did not make proper contact.
- Device 9 - The load was not connected properly to the power supply terminals.

After inspection of the test rig, the pogo pin on the valve sense circuit was bent and does not extend itself every time it is pushed down, this can explain why device 8 detected the opening of the shut off valve pulse but not the closing pulse. The UART connection to the test bench proved to be at times faulty, which is either because of a problem in the test software or in the hardware. The communication error occurs when the DUT is switched off and the UART lines are not pulled high by the DUT but by the test bench. Although, communication to the test bench can be re-established only form either a new remote log in instance or restart of the RP.

Implementation Findings The validation test proved to give insight on complete system design choices, the design choices made in the SC as well as in the test system implicates how easily, accurate and efficient tests can be performed. The pogo pins on the test rig, seemed very fragile and resulted in some misreadings after numerous tests have run. The power supply terminal screw downs, of the SC, also proved to be a bottle neck in the whole system, thus if the tester is not careful loose wires can be hazardous and it will effect the test results too. Communication of the system is very important, and having to re-establish connection during tests proved to cause more problems than to just restart the whole test procedure. The test system proved to work 60% of the time, with concerns for false detection and communication interruptions.

The results can be validated during the test procedure, which can be used to ensure a proper connection to the test rig is present, and communication errors can be picked up during the test procedure since a strike out feature is present in the test manager software. The communication error was addressed by removing all the resistors between the three UARTS connected on the same wires, but to no avail.

It is evident that improvements in the test rig and the test manager software can be made. Safety checks should be set in place to ensure proper connection to all the pogo pins. The communication error between the test manager software and the test bench needs improvement.

Table 5.9: Test System Validation with 10 Smart Controllers Tested with the Test Procedure in Appendix D.1.

Step #	Device:	1	2	3	4	5	6	7	8	9	10
	IMEI	352432065551821	3524320655548785	3524320655548793	3524320655548801	3524320655548975	352432065551433	3524320655549106	3524320655549205	3524320655551607	352432065551722
0	Measured Result Score Result	[1, 0, 1673, 4, 2] [True, True, True, -1, -1]	[1, 0, 1672, 2, 1] [True, True, True, -1, -1]	[1, 0, 1674, 4, 3] [True, True, True, -1, -1]	[1, 0, 1670, 3, 3] [True, True, True, -1, -1]	[1, 0, 1677, 2, 2] [True, True, True, -1, -1]	[1, 0, 1673, 4, 3] [True, True, True, -1, -1]	[1, 0, 1671, 3, 4] [True, True, True, -1, -1]	[1, 0, 1671, 2, 3] [True, -1, True, -1, -1]	[1, 0, 1674, 5, 3] [True, True, True, -1, -1]	[1, 0, 1669, 5, 3] [True, True, True, -1, -1]
1	Measured Result Score Result	1588 TRUE	1591 TRUE	1607 TRUE	1586 TRUE	1600 TRUE	1576 TRUE	1609 TRUE	1575 TRUE	0 FALSE	1595 TRUE
2	Measured Result Score Result	[4, 14, 34, 53] [True, True, True, True]	[4, 14, 32, 50] [True, True, True, True]	[4, 14, 34, 53] [True, True, True, True]	[4, 14, 34, 52] [True, True, True, True]	[4, 13, 31, 48] [True, True, True, True]	[4, 14, 34, 52] [True, True, True, True]	[0, 0, 0, 0] [True, False, False, False]	[4, 13, 32, 50] [True, True, True, True]	[4, 14, 33, 51] [True, True, True, True]	[4, 13, 32, 50] [True, True, True, True]
3	Measured Result Score Result	30 TRUE	30 TRUE	30 TRUE	30 TRUE	30 TRUE	35 TRUE	0 FALSE	30 TRUE	30 TRUE	30 TRUE
4	Measured Result Score Result	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE	Closed TRUE
5	Measured Result Score Result	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE	Off TRUE
6	Measured Result Score Result	[0, 1, 1671, 3, 4] [True, True, True, -1, -1]	[0, 1, 1672, 1, 2] [True, True, True, -1, -1]	[0, 1, 1673, 3, 3] [True, True, True, -1, -1]	[0, 1, 1671, 1, 3] [True, True, True, -1, -1]	[0, 1, 1678, 1, 1] [True, True, True, -1, -1]	[0, 1, 1676, 1, 1] [True, True, True, -1, -1]	[0, 1, 1668, 2, 2] [True, True, True, -1, -1]	[1, 0, 1671, 2, 4] [False, False, True, -1, -1]	[0, 1, 1674, 4, 4] [True, True, True, -1, -1]	[0, 1, 1667, 3, 4] [True, True, True, -1, -1]
8	Measured Result Score Result	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE	Open TRUE
9	Measured Result Score Result	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE	On TRUE
10	Measured Result Score Result	[1, 0, 1671, 2, 2] [True, True, True, -1, -1]	[1, 0, 1669, 3, 4] [True, True, True, -1, -1]	[1, 0, 1675, 0, 1] [True, True, True, -1, -1]	[1, 0, 1670, 4, 4] [True, True, True, -1, -1]	[1, 0, 1680, 2, 2] [True, True, True, -1, -1]	[1, 0, 1675, 2, 2] [True, True, True, -1, -1]	[0, 1, 1670, 2, 3] [False, False, True, -1, -1]	[1, 0, 1670, 2, 2] [True, True, True, -1, -1]	[1, 0, 1674, 1, 4] [True, True, True, -1, -1]	[0, 1, 1668, 4, 3] [False, False, True, -1, -1]

Chapter 6

Conclusion

6.1 Overview

In this chapter the thesis is concluded by evaluating the firmware and test system developed with the research objectives listed in Chapter 1. Firstly, the SC firmware and test system is evaluated. Finally, recommendations and a summary for each is made.

6.2 Evaluation

6.2.1 Smart Controller Firmware

The SC was investigated and functional peripheral firmware was designed and implemented. The developed peripheral-side firmware combined with the communication-side firmware functions reliably on a total of 245 SC devices.

The firmware validation proved that the expected measurements have an error of less than 10%, as stipulated by specification HS[7] in Table 3.2. The firmware provides a base for future research and development with the ATXmega chip sets.

The SC for EWHs project deemed successful in eMkhondo, allowing various contributions to make use of the data gathered and contribute to EWH related research. The firmware deemed reliable to gather all the data used after addressing memory corruption problems and removing a interrupt based analog sweep functionality to measure all the analog pins sequentially. The firmware was stable at the time of development, any underlying firmware errors have not surfaced yet.

6.2.2 Test System

The test system consists of a hardware test bench and software to manage the test procedure. The hardware went through two iterations, finally resulting in the test bench MK 2 which is implemented with the test manager software. The software consisted of three main handler classes, two for communication and one for managing the test procedure. The communication classes, use MQTT to connect to the cloud and subscribe to the topics applicable to the SC under test. Alternately, UART communication was used to setup the modem on the SC and to communicate commands to the MK 2 test bench.

The test bench hardware measures and emulates the output and input signals of the SC accurately. Allowing the SC to gain telemetry results from the test bench which are the same as what can be expected from when the SC is connected to an actual EWH.

A prototype test rig was built to simulate a test application environment. A test procedure was developed to increase test productivity and allow a more simplistic way of testing a SC. This test can be run by anyone capable of screwing down screws and pressing a button. The test system integration proved to work for 60% of the SCs validated on the prototype test rig.

Insight was gained on possible errors, test system side and tester side. The errors prone to exist is mainly due to a communication error in the test system software, the test system software proved as a viable test platform most of the times. The communications were re-established once the test system software was restarted, indicating the test bench hardware was unaffected by communication errors. The SC connected to the UART lines might cause interference since it is continually switched on and off, causing voltage fluctuations on the UART lines.

6.3 Recommendation

6.3.1 Smart Controller Firmware

The future of IoT-based smart controllers are to measure and control devices completely remote. This would require battery powered SCs, which will require future work to focus on battery efficient MCU firmware to prolong the battery life. The dynamic memory access(DMA) features of the ATXmega128A4U might prove to achieve more efficient sensor measurements. The firmware is very dependant of the hardware connected to the MCU, thus firmware improvements will only be deemed necessary with hardware changes. These changes can either be when a large amount of SCs are required for a specific application or when the peripheral circuitry improved. Another aspect to which the firmware can change is, with the use of the modem. The PCB operational status can be monitored and transmitted to the cloud, such as brown out detections on the voltage levels. The modem can also provide access to a USSD terminal, where SIM card specific settings can be changed or requested remotely.

6.3.2 Test System

The hardware test bench of the test system makes use of tab terminal relays, which can cause the connected RP possible damage if too much force is used to connect to the tab terminals. Screw down terminals will suffice in future iterations if a test rig is used where the connected wires are not disconnected regularly. The communication reliability between the SC and computational platform can be improved to rather make use of the USB functionality on the MK 2 test bench and instead, connect to the SC with a USB to serial converter cable. An alternative is to route the RP, SC, and the test bench UARTS to a separate (the MCU has five) UART and relay messages between devices using a dedicated UART for each.

The test system software can make use of an retry when failed process, to ensure the failure detected is not caused by comparing mistimed status report strings with the wrong test step compare value.

The test software can be improved with a graphical user interface (GUI), providing the tester visual buttons and text entry fields to easily edit the report document with tester identification and required details. The test engineer can edit the test procedure on the GUI, or individual test steps can be issued and executed from the GUI.

The test rig can be improved with more durable pogo pins and provide an easier, yet safe, method of connection the high voltage supply to the SC. The test system can become more durable when it is encased in a sturdy case with only the required connection ports brought out.

The remote terminal access to the RP worked well, but can prove difficult to comprehend for a low level technically skilled tester. An alternative, such as a desktop personal computer will provide the processing power required and a screen where the tester can evaluate the results.

Future test systems can look into boundary scan or JTAG test procedures, which allow the measuring and controlling of the MCU pins. Providing instant feedback without the need to program firmware to the MCU to run a test sequence. Test sequences will be executed faster with direct pin management and accurate measurements are taken. The test hardware rig will become obsolete and the test procedure will be mostly software based.

6.4 Concluding Remarks

The need for managing resources more efficiently, and by monitoring and controlling known sources of high resource consumption, lead to the SC. Allowing resource management and monitoring incentives to adapt and grow with the ever increasing IoT sphere. The developed firmware contributes to the functionality of the SC and allows gathering of data from devices previously considered simple and unimportant, such as an EWH. In-house test procedures prove to be time consuming and dubious when the required measuring equipment is unaffordable or hard to come by. The test system developed, provides aid to testing low level FMEA devices at a low cost. Ultimately, if tests are required to be precise and consistent, a test module from National Instruments or alike is recommended [29]. A final remark, during product design, design the hardware with the test system in mind.

Appendices

Appendix A

Smart Controller Firmware

A.1 Firmware Function Diagrams

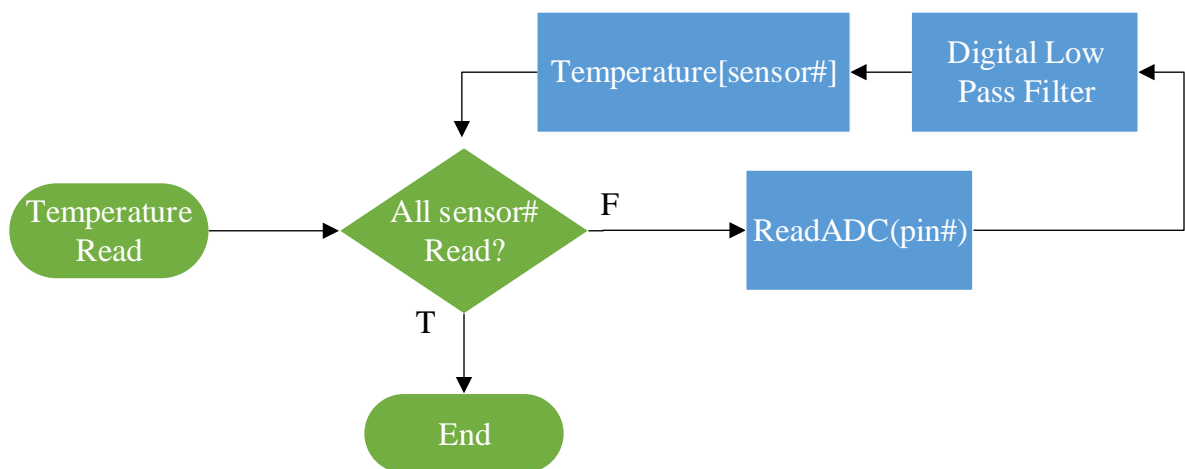


Figure A.1: Temperature Readings Diagram.

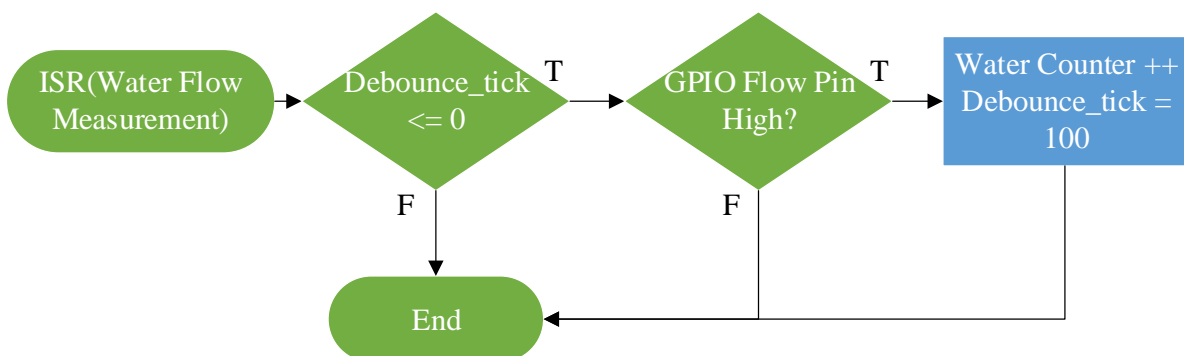


Figure A.2: Water Flow Measurement Flow Diagram.

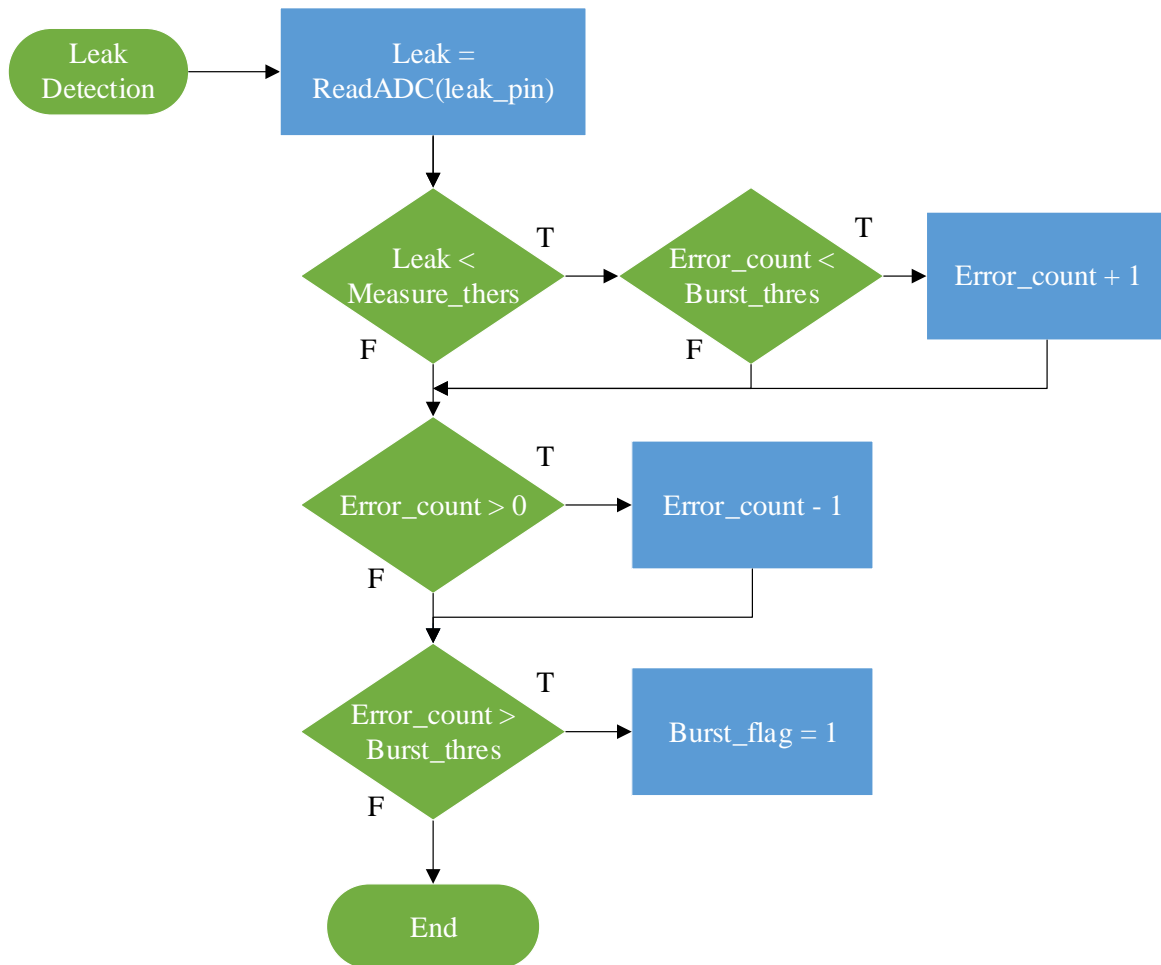


Figure A.3: Leakage Detection Flow Diagram.

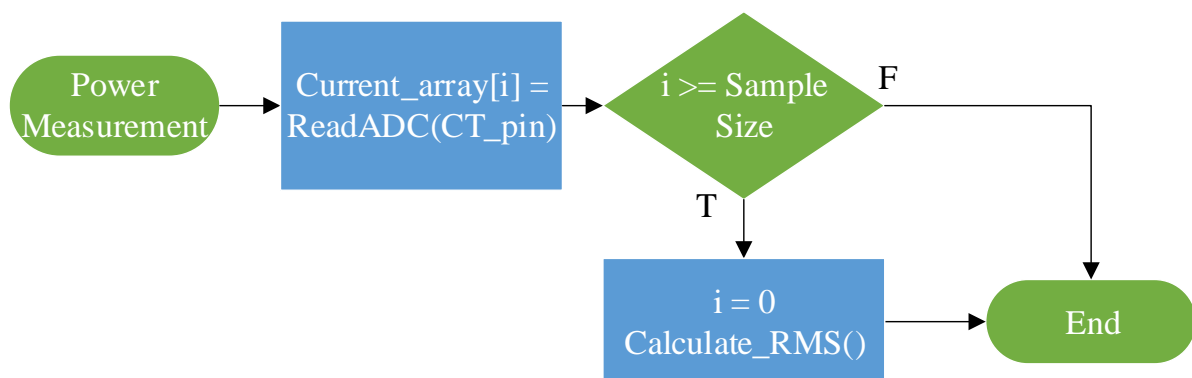


Figure A.4: Power Measurements Service Function Flow Diagram.

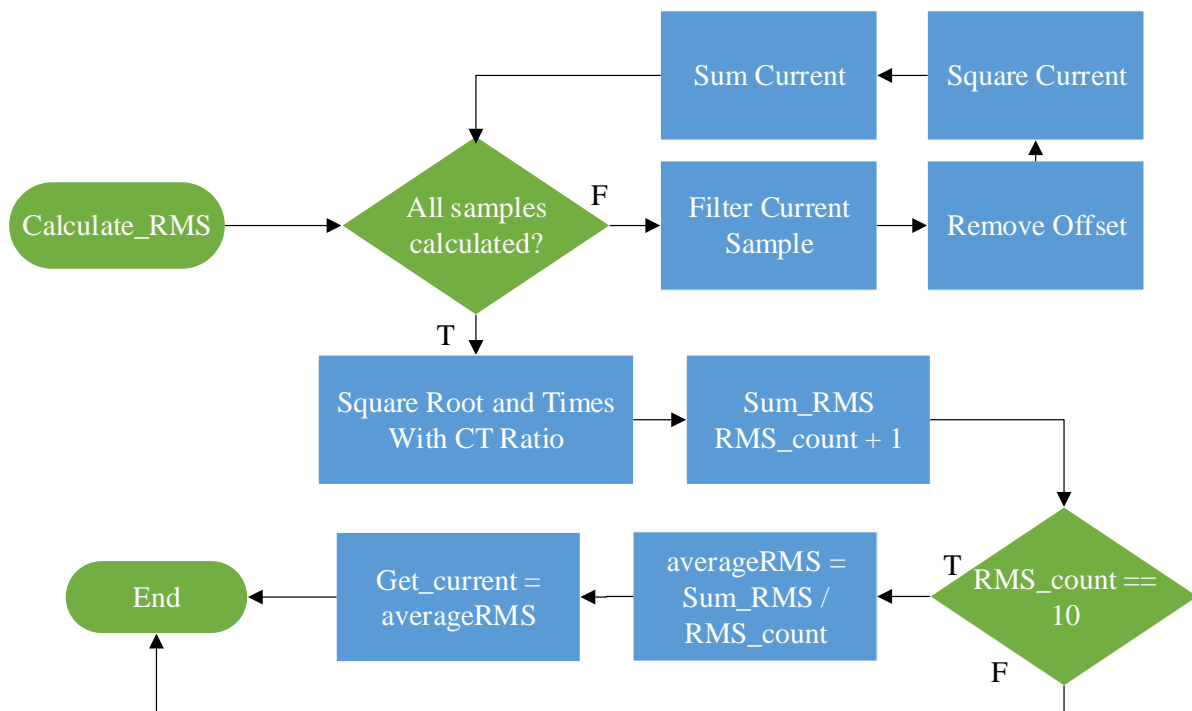


Figure A.5: Power Measurements RMS Calculation Function Flow Diagram.

Appendix B

Test Bench MK 1 and 2 Hardware

B.1 Schematic Test Bench MK 1

APPENDIX B. TEST BENCH MK 1 AND 2 HARDWARE

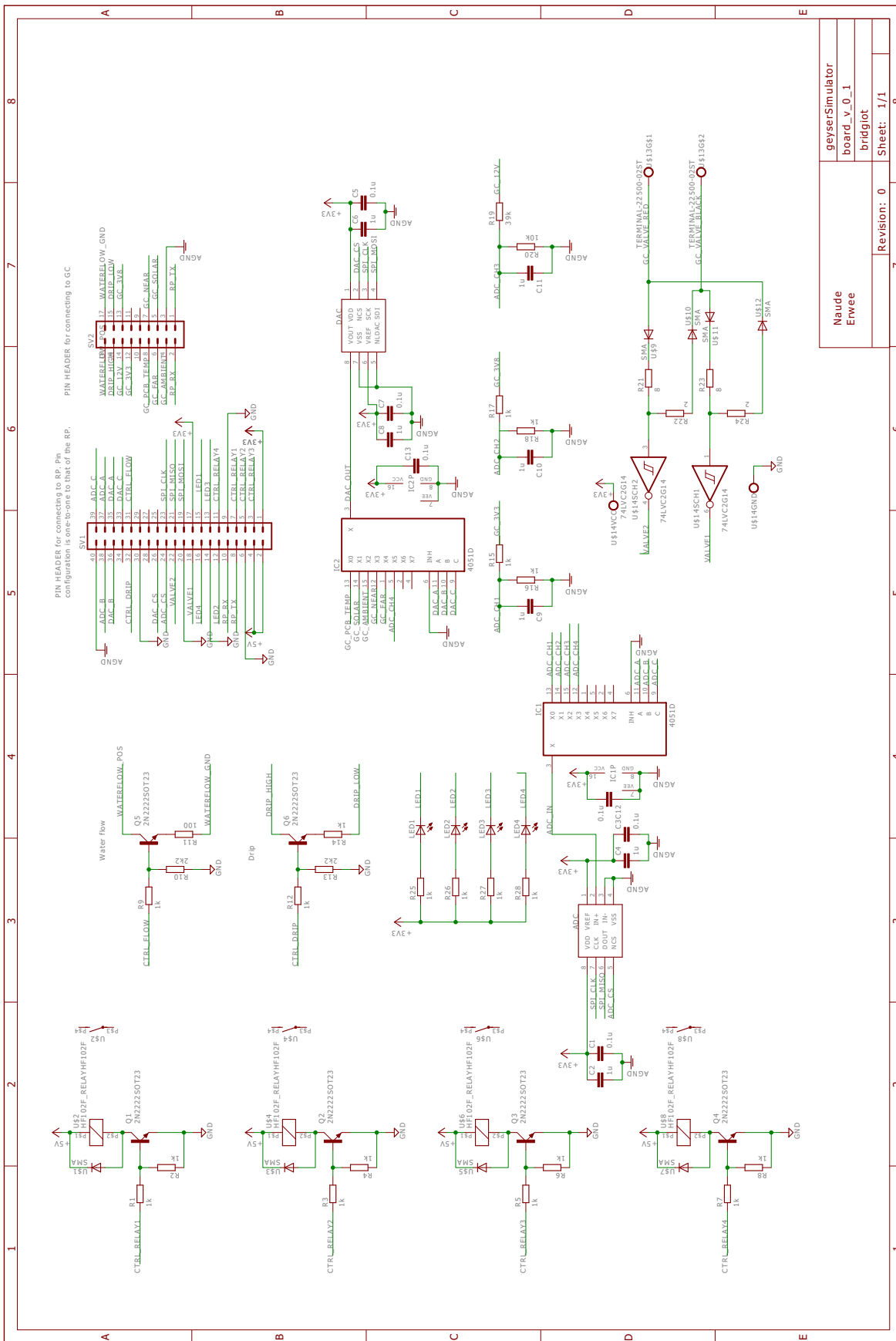
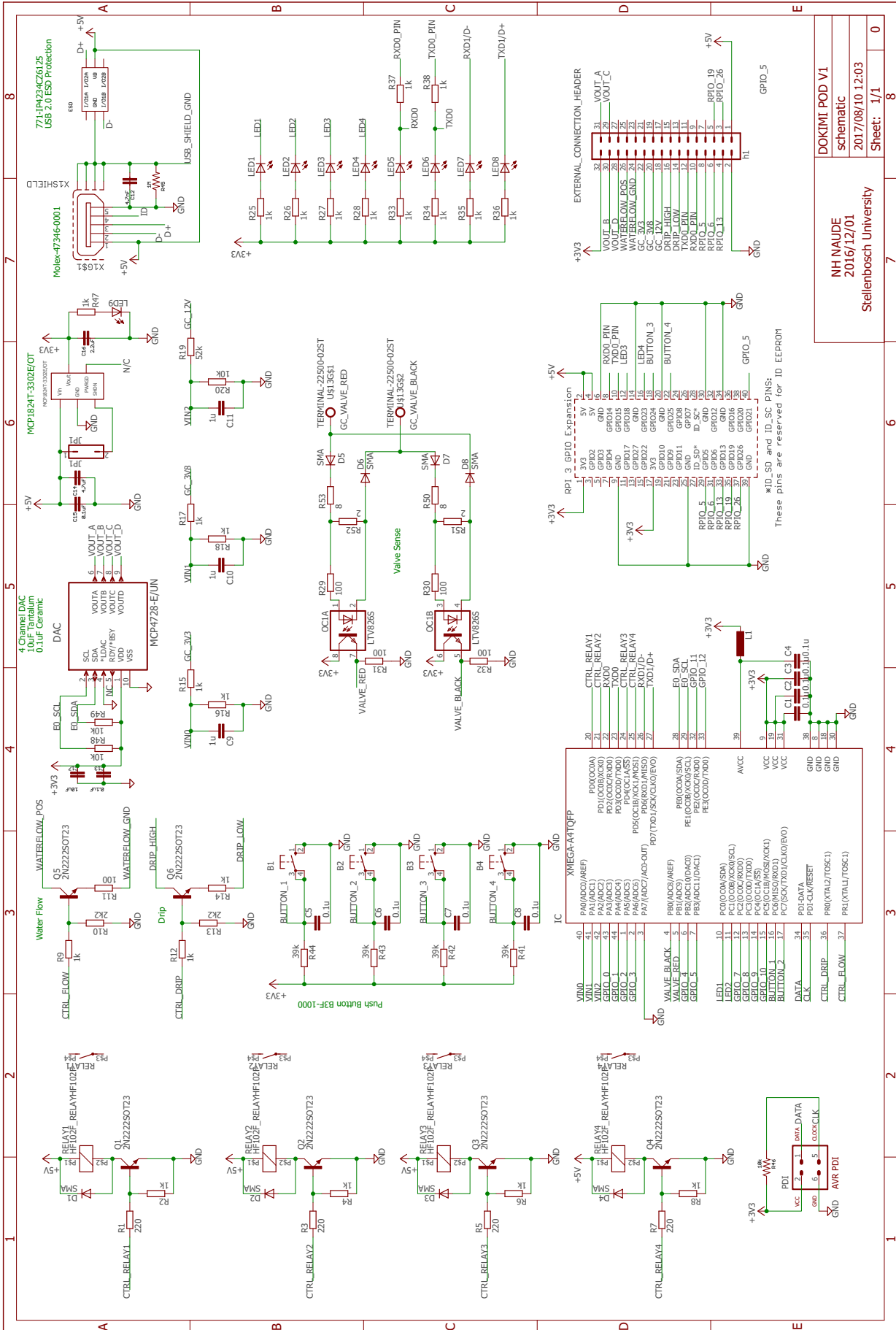


Figure B.1: Test Bench MK 1 Schematic Diagram.

B.2 Schematic Test Bench MK 2



NH NAUDE
2016/12/01
Stellenbosch University

DOKIMI POD V1
schematic
2017/08/10 12:03
Sheet: 1/1

Figure B.2: Test Bench MK 2 Schematic Diagram.

B.3 MK 2 Firmware Functions

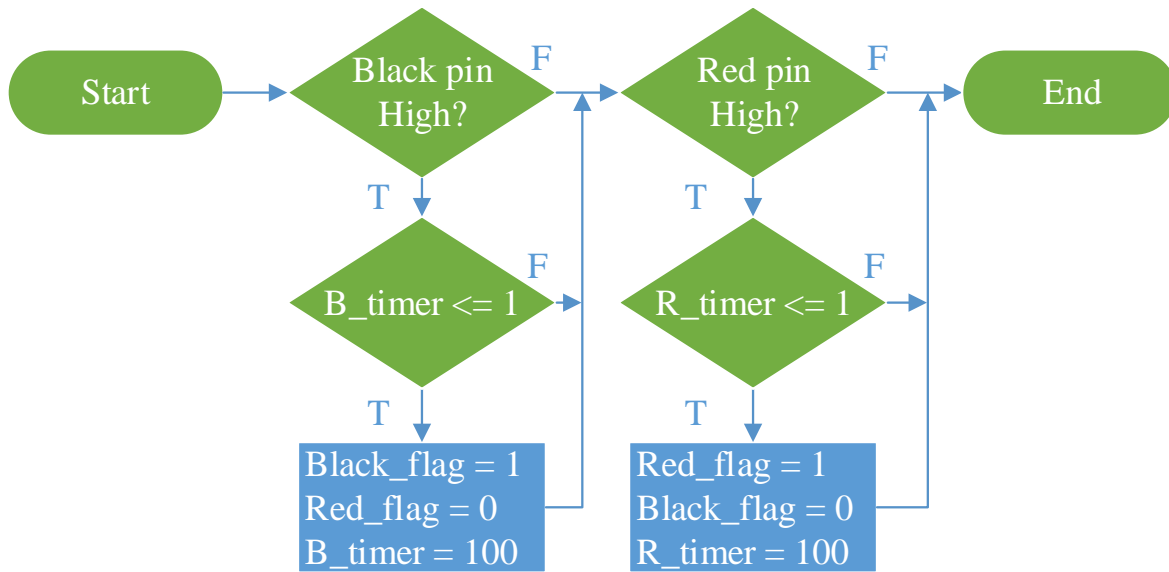


Figure B.3: Valve Sense Service Flow Diagram.

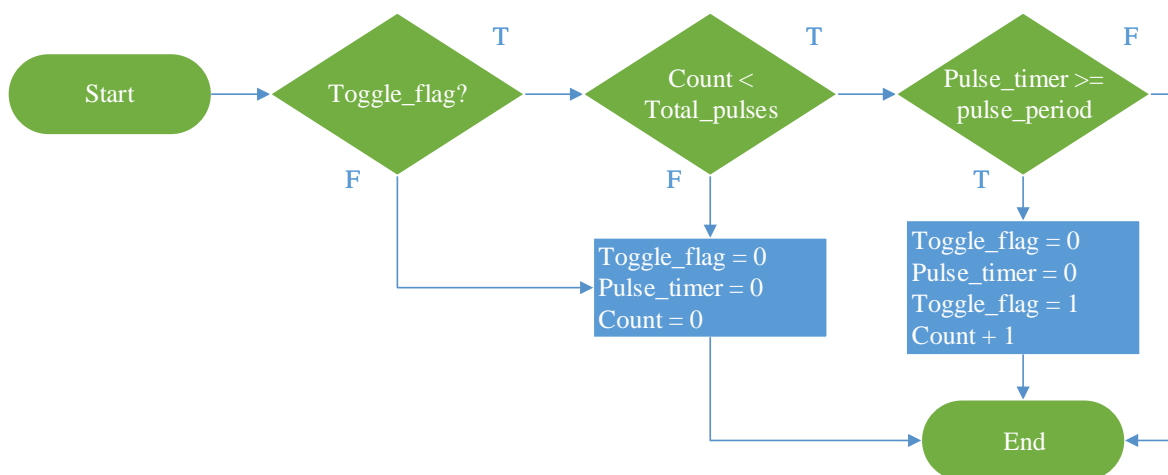


Figure B.4: Pulse Service Flow Diagram.

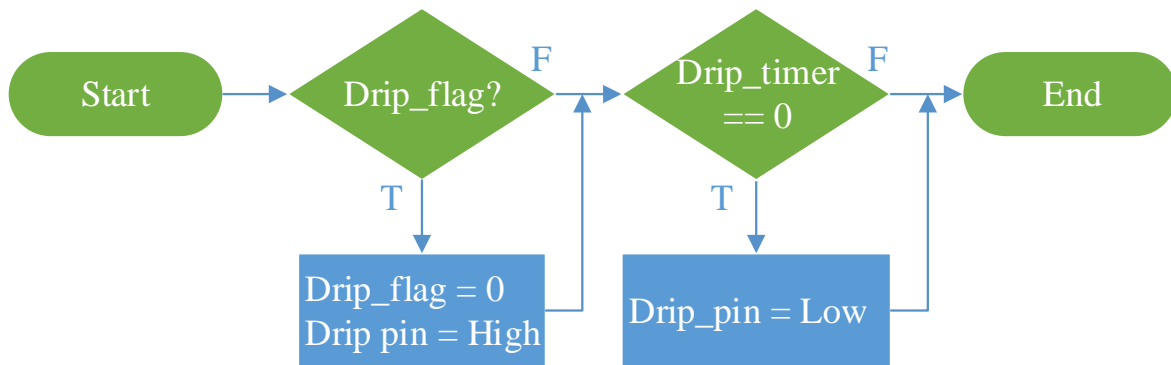


Figure B.5: Drip Service Flow Diagram.

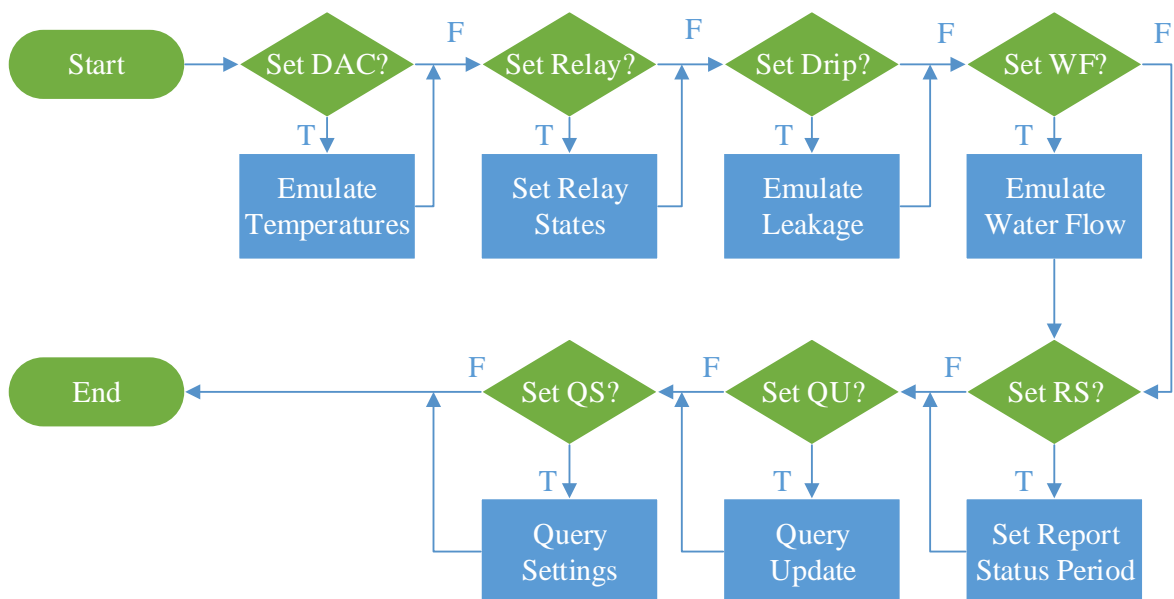


Figure B.6: Control Service Flow Diagram.

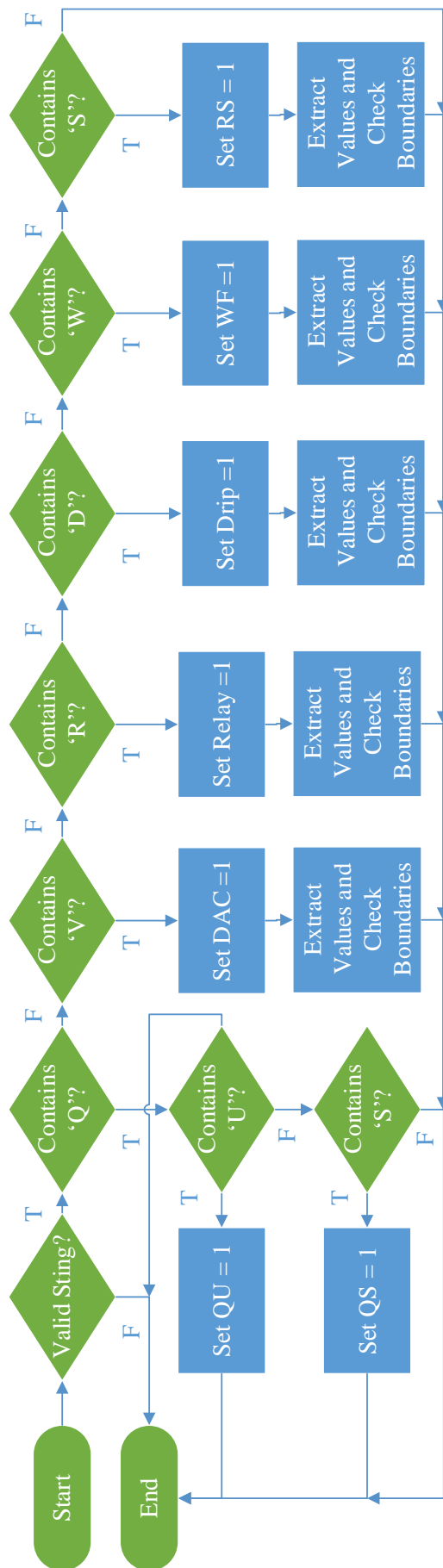


Figure B.7: Communication Service Parser Flow Diagram.

Appendix C

Test System Software

C.1 Test Manager Software Class Diagrams

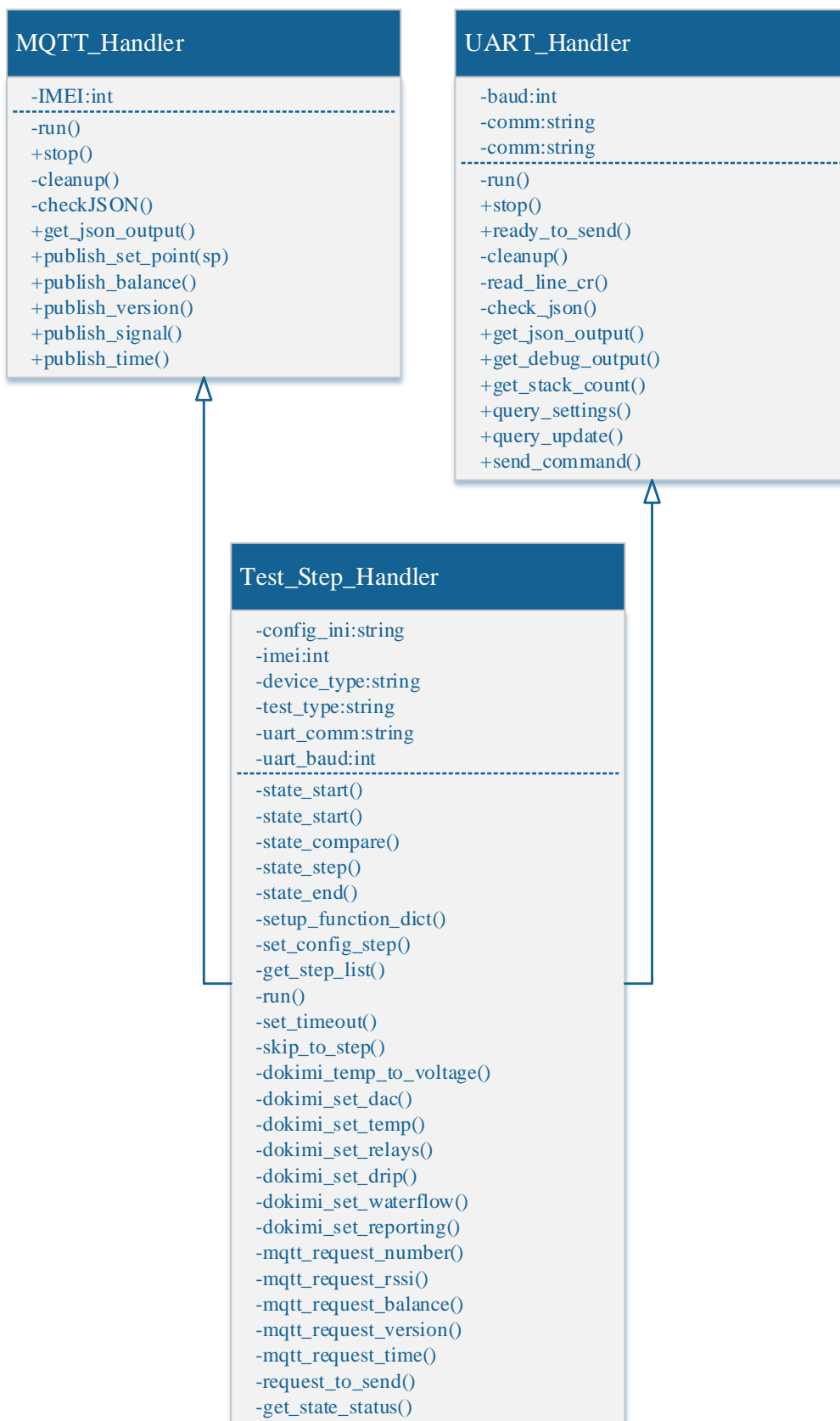


Figure C.1: Test Manager Class Diagrams.

C.2 Test System Handler UML

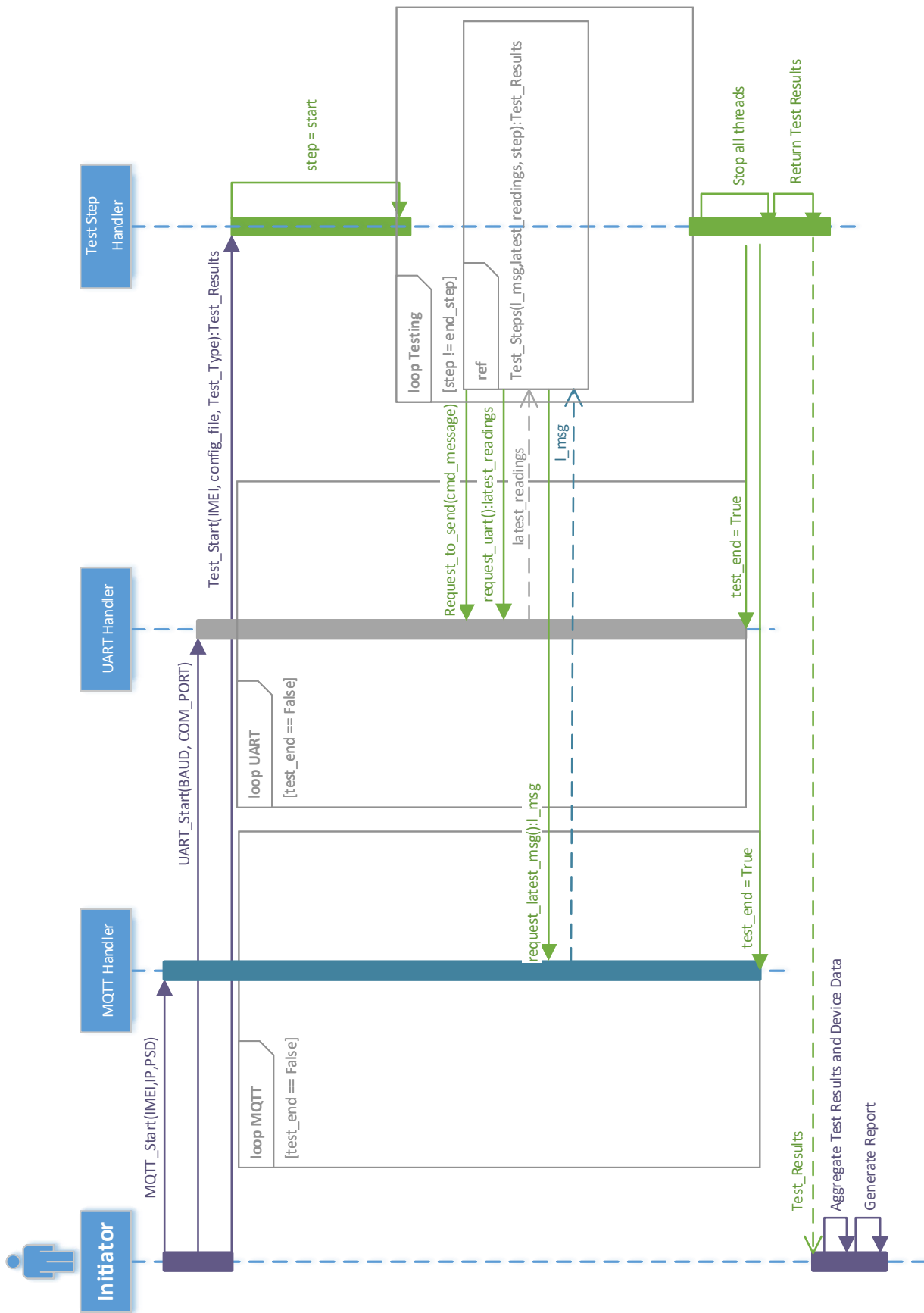


Figure C.2: Test System Handler UML Part 1.

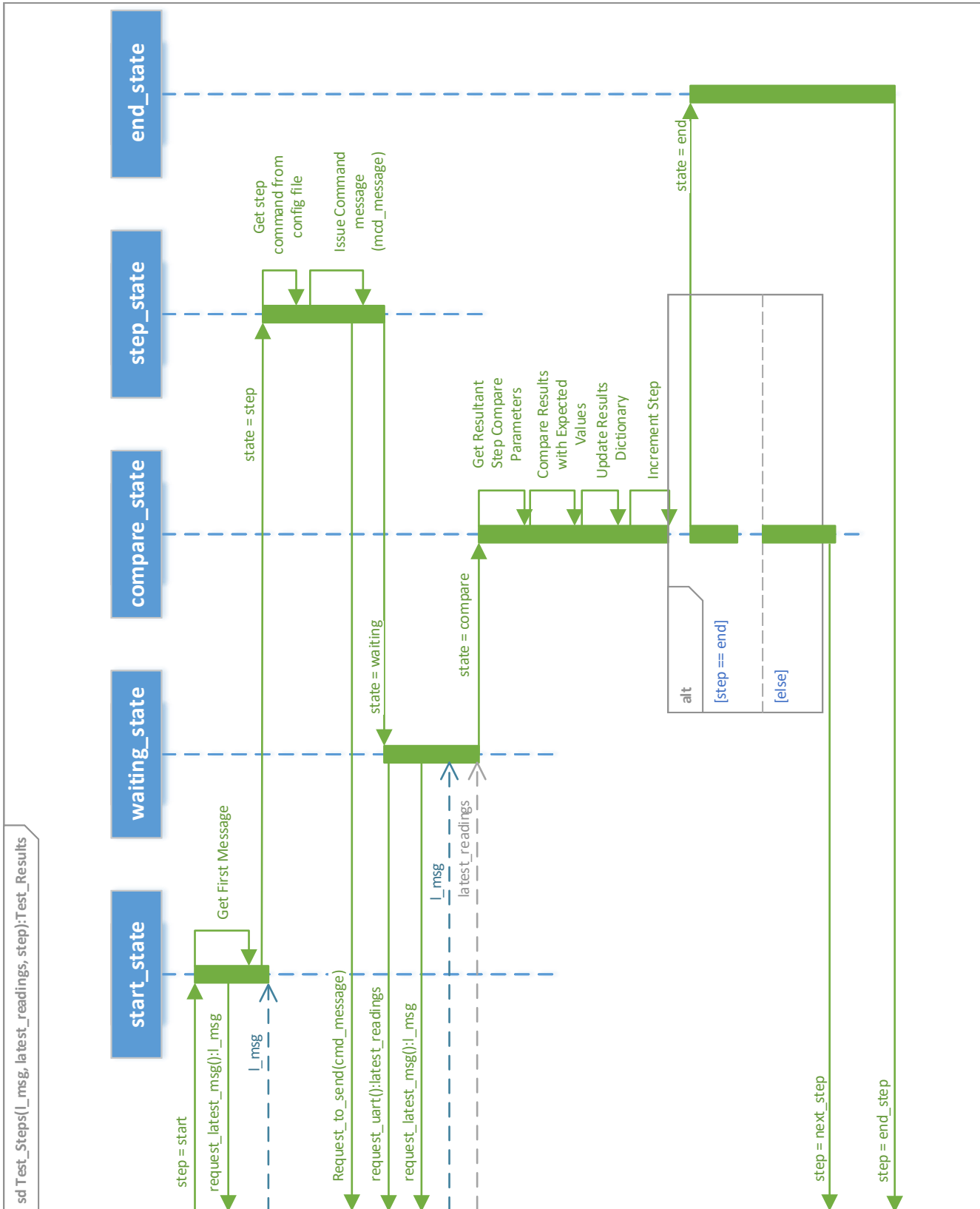


Figure C.3: Test System Handler Part 2.

Appendix D

Results

D.1 System Validation Configuration File Setup

Listing D.1: Configuration File used in Validation of Test System as a Whole.

```
1  [step 0]
2  #Switch Device ON
3  SET_RT = [0,10]
4  SET_RELAYS = [0,0,1,1]
5  G_QU = [1,0,1500,0,0]
6  SET_TEMP = [0, 0, 0 ,0]
7
8  [compare 0]
9  G_QU
10 THRES = 500
11
12 [step 1]
13 # Switch Load 1 on
14 SET_RELAYS = [1,0,1,1]
15 M_NUM = [0]
16
17 [compare 1]
18 SET_RELAYS
19 VALS = 1500
20 THRES = 200
21
22 [step 2]
23 # Set Temperatures (will keep setting since DAC have eeprom)
24 SET_TEMP = [5, 15, 35 ,55]
25 M_RSSI = [0]
26
27 [compare 2]
28 SET_TEMP
29 # No need for vals
30 THRES = 5
31
32 [step 3]
```

Listing D.1 (Cont.): Configuration File used in Validation of Test System as a Whole.

```
33 SET_WF = [20, 30]
34
35 [compare 3]
36 SET_WF
37 VALS = 30
38 THRES = 10
39
40 [step 4]
41 M_VER = [0]
42 SET_DRIP = [1, 30]
43 ;A_DELAY = [4]
44
45 [compare 4]
46 SET_DRIP
47 VALS = Closed
48
49 [step 5]
50 M_SP = [50]
51
52 [compare 5]
53 M_SP
54 VALS = Off
55
56 [step 6]
57 G_QU = [0,1,1500,0,0]
58
59 [compare 6]
60 G_QU
61 THRES = 500
62
63 [step 7]
64 SET_RELAYS = [0,0,0,0]
65 A_DELAY = [5]
66 A_CONTINUE = [0]
67
68 [step 9]
69 SET_RELAYS = [0,0,1,1]
70 A_DELAY = [40]
71 M_BAL = [0]
72 SET_DRIP = [0, 0]
73
74 [compare 9]
75 SET_DRIP
76 VALS = Open
77
78 [step 10]
79 M_SP = [100]
80 ;A_DELAY = [5]
81
82 [compare 10]
```

Listing D.1 (Cont.): Configuration File used in Validation of Test System as a Whole.

```
83 M_SP
84 VALS = 0n
85
86 [step 11]
87 G_QU = [1,0,1500,0,0]
88
89 [compare 11]
90 G_QU
91 THRES = 500
92
93 [step end]
94 # Keep Empty for completion of steps
95
96 [DEFAULT_JSON]
97 V = Open
98 R = Off
99 W = 0
100 Hm = 0
101 Cm = 0
102
103 # FUNCTION LIST is hard coded in system files, always change additions
    there too.
104 [FUNCTIONS_LIST]
105 SET_TEMP
106 SET_RELAYS
107 SET_DRIP
108 SET_WF
109 SET_RT
110 M_NUM
111 M_RSSI
112 M_BAL
113 M_VER
114 M_TIME
115 M_SP
116 A_DELAY
117 A_CONTINUE
118 G_QU
119
120 [DEFAULT_COMPARE_SETTINGS]
121 # Keys for comparing
122 SET_TEMP = ['T1', 'T3', 'T2', 'T4']
123 SET_RELAYS = W
124 M_SP = R
125 SET_DRIP = V
126 SET_WF = Hm
127 M_RSSI = ss
128 M_VER = ver_req
129 M_TIME = time
130 G_QU = ['VR', 'VB', 'GV3V3', 'GV3V4', 'GV12V']
```

D.2 Generated PDF Report

Dokimi Test Results

Tester Name: Test Bench Mk2 Tester

Friday 22nd September, 2017
16:18

Summary

Overall Result: **Failed**

Step	Function
2	set_temp

Device Details

Details	
Device Tyoe	Geasy
IMEI	352432065549106
Signal Strength	12,0
Test Type	Production Setup
no	27732289266
sp	50

Bibliography

- [1] “Make water saving a way of life.” [Online]. Available: <http://www.capetown.gov.za/Family%20and%20home/residential-utility-services/residential-water-and-sanitation-services/make-water-saving-a-way-of-life>
- [2] “Conserving electricity.” [Online]. Available: http://www.eskom.co.za/AboutElectricity/ElectricityTips/Pages/Conserving_Electricity.aspx
- [3] “City of Cape Town.” [Online]. Available: <http://www.capetown.gov.za/Family%20and%20home/residential-utility-services/residential-water-and-sanitation-services/make-water-saving-a-way-of-life>
- [4] “News24 Mandela Bay on brink of disaster as water crisis deepens :Tuesday 7 March 2017.” [Online]. Available: <https://www.news24.com/SouthAfrica/News/mandela-bay-on-brink-of-disaster-as-water-crisis-deepens-20170307>
- [5] “Nelson Mandela Bay Municipality.” [Online]. Available: <http://www.nelsonmandelabay.gov.za/dam.aspx>
- [6] “Eskom load shedding.” [Online]. Available: <http://loadshedding.eskom.co.za/loadshedding/description>
- [7] “Power Alert.” [Online]. Available: <http://www.eskom.co.za/sites/idm/Pages/Power-Alert.aspx>
- [8] L. Venter, F. G. T. Radloff, L. J. Grobler, and A. Z. Dalglish, “Power alert: Still a unique and innovative way to reduce residential demand,” in *Domestic Use of Energy Conference (DUE), 2013 Proceedings of the 21st*. IEEE, 2013, pp. 1–5. [Online]. Available: <http://ieeexplore.ieee.org/abstract/document/6524785/>
- [9] Nelson Mandela Bay Municipality, “Official Municipal Notice to Residents - Geyser Control Installation.” [Online]. Available: http://www.nelsonmandelabay.gov.za/datarepository/documents/NTvqc_RES%20NOTICE%20signed%20final.pdf
- [10] “CENTLEC | Geyser Control.” [Online]. Available: <http://www.centlec.co.za/Pages/Page/Geyser-Control>
- [11] “Geyser Control.” [Online]. Available: <https://www.citypower.co.za/customers/Pages/Geyser-Control.aspx>
- [12] BridgIoT, “Geasy - Smart Controller for Your Home.” [Online]. Available: <http://www.geasy.co.za/>

- [13] E. Jordan, “The geyser that gives you advice based on your water consumption patterns.” [Online]. Available: <https://mg.co.za/article/2016-07-29-00-the-geyser-that-gives-you-advice-based-on-your-water-consumption-patterns/>
- [14] “News - Smart Water Meter provides innovative solution...” [Online]. Available: <http://www.sun.ac.za/english/Lists/news/DispForm.aspx?ID=4735>
- [15] M. Roux, N. Naude, M. Booysen, and A. Barnard, “Electric water heaters in smart-grids: Individual savings versus network peak load management,” *SAUPEC. In stampa*, 2017.
- [16] “Internet of Things | Define Internet of Things at Dictionary.com.” [Online]. Available: <http://www.dictionary.com/browse/internet-of-things>
- [17] A. Meola, “IoT For Utilities: Smart Water, Gas & Electric Utilities Coming Soon.” [Online]. Available: <http://www.businessinsider.com/internet-of-things-utilities-water-electric-gas-2016-10>
- [18] A. H. Cloete, J. W. K. Brown, M. J. Booysen, R. Steinke, and T. Magedanz, “Smart Grid Application using ETSI M2m: Domestic Electric Water Heaters,” *SAUPEC 2016 Proceedings*, 2016. [Online]. Available: https://www.researchgate.net/profile/Mj_thinus_Booyesen/publication/289819637_Smart_Grid_Application_using_ETSI_M2M_Domestic_Electric_Water_Heaters/links/5692cc1608aee91f69a72b53/Smart-Grid-Application-using-ETSI-M2M-Domestic-Electric-Water-Heaters.pdf
- [19] M. Roux and M. J. Booysen, “Use of smart grid technology to compare regions and days of the week in household water heating,” in *2017 International Conference on the Domestic Use of Energy (DUE)*, Apr. 2017, pp. 276–283.
- [20] “Intelligent Geyser System | MTN Mobile Intelligence Lab.” [Online]. Available: <http://mtn.sun.ac.za/projects/intelligent-geyser-system/>
- [21] “Electrical & Electronic Engineering | Stellenbosch University.” [Online]. Available: <http://www.ee.sun.ac.za/>
- [22] “Mkhondo Local Municipality.” [Online]. Available: <http://www.mkhondo.gov.za/>
- [23] “Companies - LaunchLab.” [Online]. Available: <https://launchlab.co.za/companies/>
- [24] J. W. K. Brown, “Design, and implementation of an intelligent water heater control module for feedback demand side management,” Ph.D. dissertation, Stellenbosch: Stellenbosch University, 2016.
- [25] D. Petterson, “SABS, SANS or SATAS? Clearing up the confusion | Infrastructure news.” [Online]. Available: <http://www.infrastructurenews.com/2016/06/14/sabs-sans-or-satas-clearing-up-the-confusion/>
- [26] A. H. Cloete, “A domestic electric water heater application for smart grid.” Ph.D. dissertation, Stellenbosch: Stellenbosch University, 2017.
- [27] Rico Möckel, “How to design and implement firmware for embedded microcontrollers.pdf,” Jun. 2010. [Online]. Available: <https://biorob.epfl.ch/files/content/sites/biorob/files/users/165511/public/How%20to%20design%20and%20implement%20firmware%20for%20embedded%20microcontrollers.pdf>

- [28] Vassili van der Mersch, “Automated Testing for the Internet of Things,” May 2016. [Online]. Available: <https://nordicapis.com/automated-testing-for-the-internet-of-things/>
- [29] “VirtualBench All-in-One Instrument - National Instruments.” [Online]. Available: <http://www.ni.com/en-za/shop/select/virtualbench-all-in-one-instrument>
- [30] “Advice for Early-Stage Hardware Startups.” [Online]. Available: <https://blog.ycombinator.com/advice-for-early-stage-hardware-startups/>
- [31] D. H. Stamatis, *Failure mode and effect analysis: FMEA from theory to execution*. ASQ Quality Press, 2003.
- [32] “ATxmega128a4u - 8-bit AVR Microcontrollers.” [Online]. Available: <http://www.microchip.com/wwwproducts/en/ATxmega128A4U>
- [33] Atmel, “XMEGA A MANUAL, 8-bit Atmel XMEGA A Microcontroller.” [Online]. Available: <https://partsearch.su/files/datasheet/doc8077-32158.pdf>
- [34] M. J. Booyesen, J. Engelbrecht, and A. Molinaro, “Proof of concept: Large-scale monitor and control of household water heating in near real-time,” 2013.
- [35] P. J. C. Nel, “Rethinking electrical water heaters,” Ph.D. dissertation, Stellenbosch: Stellenbosch University, 2015.
- [36] J. Brown and M. J. Booyesen, “An intelligent water heater with wi-fi access to support demand-side management,” 2015.
- [37] “MQTT.” [Online]. Available: <http://mqtt.org/>
- [38] MongoDB.com, “What Is MongoDB?” 2017. [Online]. Available: <https://www.mongodb.com/what-is-mongodb>
- [39] u-blox, “LEON-G1,” Jun. 2015. [Online]. Available: <https://www.u-blox.com/en/product/leon-g1>
- [40] Zach Supalla, “The Photon changelog - what’s new and different from the Core,” May 2015. [Online]. Available: <https://community.particle.io/t/the-photon-changelog-whats-new-and-different-from-the-core/11823>
- [41] “Product Design & Development.” [Online]. Available: <https://www.rqriley.com/pro-dev.htm>
- [42] “Quality control | Define Quality control at Dictionary.com.” [Online]. Available: <http://www.dictionary.com/browse/quality-control>
- [43] Sylvia Wu, “Hardware Testing Guide for Mass Production | Fictiv - Hardware Guide,” Jan. 2016. [Online]. Available: <https://www.fictiv.com/hwg/test/hardware-testing-guide-for-mass-production>
- [44] “OEM Electronic Foundry,” 2017. [Online]. Available: <http://www.diamondnpi.com/index.html>

- [45] Jim Eastman, “A Primer on Manufacturing Tests for Your Electronics | Mindtribe,” Oct. 2016. [Online]. Available: <http://mindtribe.com/2016/10/a-primer-on-manufacturing-tests-for-your-electronics/>
- [46] A. Voto, I. Dai, P. Oleniuk, and B. Todd, “Standardization of automated industrial test equipment for mass production of control systems,” *Journal of Instrumentation*, vol. 11, no. 01, pp. C01047–C01047, Jan. 2016. [Online]. Available: <http://stacks.iop.org/1748-0221/11/i=01/a=C01047?key=crossref.707fbc7567a4077da571c265e696bdf2>
- [47] “Shop - National Instruments.” [Online]. Available: <http://www.ni.com/en-za/shop.html>
- [48] International Organisation for Standardisation, “ISO 9001 Quality management,” 2015. [Online]. Available: <https://www.iso.org/iso-9001-quality-management.html>
- [49] Y. Berra, “Software Development Life Cycle (SDLC),” *DePaul University*. [Online]. [Cited: 2 12, 2012.] <http://condor.depaul.edu/jpetlick/extra/394/Session2.ppt>. [Online]. Available: <https://s3.ap-south-1.amazonaws.com/btechktu/placement/se.pdf>
- [50] “Atmel Studio 7: Easier to Use and More Powerful than Ever - Overview.” [Online]. Available: <http://www.atmel.com/microsite/atmel-studio/>
- [51] “JSON.” [Online]. Available: <http://www.json.org/>
- [52] “The V110 KSM Polymer Volumetric Water Meter Range.” [Online]. Available: https://www.elster.nl/downloads/V110_KSM_Polymer_bodied_volumetric_meter.pdf
- [53] L.L. Leeuwner, “AC and DC Energy Monitor with Remote Load-Switching Capabilities for use in IoT Applications,” Stellenbosch : Stellenbosch University, Technical, 2016.
- [54] T.G. Durand, “Detachable Solar Vehicle Tracking Solution with IoT Application.” Stellenbosch : Stellenbosch University, Technical, 2016.
- [55] D. N. Oruganti and S. S. Yellampalli, “Design of a power efficient SPI interface,” in *2014 International Conference on Advances in Electronics Computers and Communications*, Oct. 2014, pp. 1–5.
- [56] “I2c Info: I2c Bus, Interface and Protocol.” [Online]. Available: <http://i2c.info/>
- [57] “Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART).” [Online]. Available: <https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/>
- [58] D. Robinson, “The Incredible Growth of Python,” Sep. 2017. [Online]. Available: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>
- [59] K. Henney, “Java vs. Python: Which One Is Best for You? | Blog |,” Mar. 2017. [Online]. Available: <https://blog.appdynamics.com/engineering/java-vs-python-which-one-is-best-for-you/>

- [60] “Water conductivity - lenntech.” [Online]. Available: <http://www.lenntech.com/applications/ultrapure/conductivity/water-conductivity.htm>
- [61] doceme, “Python Spidev (py-spidev),” Feb. 2012, original-date: 2012-02-08T05:46:09Z. [Online]. Available: <https://github.com/doceme/py-spidev>
- [62] B. S. Dean, “AVRDUDE,” Jan. 2010. [Online]. Available: <http://www.nongnu.org/avrdude/>
- [63] C. Liechti, “pyserial: Python Serial Port Extension.” [Online]. Available: <https://github.com/pyserial/pyserial>
- [64] R. Light, “paho-mqtt: MQTT version 3.1.1 client class.” [Online]. Available: <http://eclipse.org/paho>
- [65] “PyLaTeX : PyLaTeX 1.2.1 documentation.” [Online]. Available: <https://jeltef.github.io/PyLaTeX/latest/index.html>
- [66] “Python Data Analysis Library: pandas: Python Data Analysis Library.” [Online]. Available: <http://pandas.pydata.org/>
- [67] “ATAtmel-ICE.” [Online]. Available: <http://www.atmel.com/tools/atatmel-ice.aspx>
- [68] “JTAGICE3.” [Online]. Available: <http://www.atmel.com/webdoc/GUID-9D10622A-5C16-4405-B092-1BDD437B4976/index.html>
- [69] Paul Wise, “5.6.Â INI formats.” [Online]. Available: <http://www.nongnu.org/chmspec/latest/INI.html>
- [70] Richard Wall, “Dictionary of Methods/Functions Python recipes ActiveState Code,” Jun. 2001. [Online]. Available: <http://code.activestate.com/recipes/65126-dictionary-of-methodsfunctions/>