Swansea University
Prifysgol Abertawe

# REAL-TIME CRITICAL SYSTEMS

## HRM Prototype & Ada Integration

AUTHOR: JORDAN LEE MAURO-BUHAGIAR

STUDENT ID: 848536

SEPTEMBER 2017

Project Dissertation submitted to Swansea University in Partial Fulfilment for the Degree of Master of Science

Department of Computer Science, Swansea University

# ABSTRACT

Real-Time Critical Systems are pervasive in human nature. Thus, this thesis outlines the main facts encompassing critical system design and perseverance towards decisiveness. In essence, every decision made throughout the project has had a shadowed theory or method, or fact outlining the directed concept. This thesis addresses a case study that illustrates factual concepts and real-life experiences regarding Ada and SPARK projects. Moreover, it outlines background, as well as research literature that aids on decision making, and elaborates the project's purpose. Additionally, the project follows a drastic approach towards building up the foundations that allows a HRM prototype to be programmed in Ada and an incremental development towards the success of the system.

Foremost importance of the thesis shall be to prove that Ada and SPARK is essential to critical systems across the globe, allowing users with similar interests to follow well-structured tutorials and implement the project.

# DECLARATIONS & STATEMENTS

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

STATEMENT 1

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by giving explicit references. A bibliography is appended.

STATEMENT 2

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

# ACKNOWLEDGEMENTS

First of all, I would like to thank Mr. Anton G Setzer for being an exceptional Supervisor and Inspiration. This project could have not been accomplished the way it was without his guidance.

I would like to thank Mr. Gary Tam for introducing me to RDP Algorithm and his effective teaching during his “Algorithms Complexity and Efficiency” lecture; making complexity not so complex, the miracle of pseudo code understandable and the lectures fun, believe it or not.

I would like to thank Rolf Ebert for his AVR-Ada Documentations, AVR-Ada Mailing List and consistency.

I would like to thank my darling girlfriend for being my biggest support and parents for making my education happen.

I would like to thank my cousin Charles Buhagiar for his generosity in taking his time on proof reading this document.

And finally, I would like to thank Swansea University for giving me the experience of a life time, allowing me to meet exceptional people and be part of an amazing environment.

# MOTIVATION

My curiosity towards pervasiveness in this world led me to this assignment. My questions required answers and my attitude shadowed ambition. I always enjoyed technology and loved to learn. Those qualities drove me towards my Graduate Degree in Computer Systems & Electronics. Upon completion, I was not ready to stop learning. After learning how systems were designed, programming was a must. Hence, Computer Science Master's Degree became my next destination. I was thrilled to learn all types of programming languages and I always adored a challenge. Ada was the biggest challenge faced and hardest programming language to grasp in my experience; only because it was different. However, the world's evolution and my vision towards it made me extremely interested.

Ada is a reflection of not just a programming language, but "the" programming language. It uses the concept of protecting, saving and preventing. We programmers program to develop systems that potentially, to an extent replace humans. Humans usually make more mistakes than a program does when it is designed to take one or multiple responsibilities and yet, programs are tested by humans. Ada uses SPARK to verify and validate critical programs that eventually saves lives; using "Theorem Provers". In my opinion, that was extremely realistic and smart; the way it should be done. When working with SPARK-Ada and reading about the miracles it has performed in this world simply inspired me.

Thus, I took the challenge on taking this subject for my thesis. I was honestly struggling to make the decision as I knew it was no easy task and completely out of my comfort zone. Somehow, I went forward with it and all my worries were then taken care of during an informal meeting with Anton G Setzer who assisted me on choosing a project topic that was not completely out of my knowledge or grasp. He elaborated how no system has yet been presented to him running an Ada program in real-time, how research was limited and how effective it would be to verify and validate it using SPARK-Ada. Therefore, my mind was set and my mission begun. I promised myself that I was going to expand Ada documentations regarding 8-Bit Microcontrollers, hoping Ada would expand on this field and get widely used via assistance from this thesis. In addition, I promised that my experience and struggles during this mission would condemn its roots and distinguish new ways to make this experience easier for others whom may be interested.

# CONTENTS

ABSTRACT ..... 0
DECLARATIONS & STATEMENTS ..... 2
ACKNOWLEDGEMENTS ..... 3
MOTIVATION ..... 4
LIST OF FIGURES ..... 8
CHAPTER 1 INTRODUCTION ..... 10
Project Goal & Objectives ..... 11
CHAPTER 2 RESEARCH MANAGEMENT ..... 12
2.1 – Research Methodology ..... 12
2.1.1 – The Research Onion ..... 12
CHAPTER 3 FOUNDATIONS ..... 15
Research philosophy ..... 15
3.1 - Literature Review ..... 16
3.1.1 - Real-world Projects with Ada & SPARK Integration ..... 16
3.1.1.A – Ada's Lines of Contribution ..... 16
3.1.1.B – SPARK Ada's Lines of Contribution ..... 20
3.1.2 – Programming Languages Used in Embedded Systems ..... 24
3.2 - Background Research ..... 31
3.2.1 - Embedded Systems ..... 31
3.2.1.A - Introduction to Heart Rate Monitors ..... 31
3.2.1.B – ECG ..... 31
3.2.1.C – PPG ..... 32
3.2.1.D – HRM Functionalities ..... 34
3.2.1.E. - Ada Support for Embedded Systems ..... 35
3.2.3 - Embedded Hardware ..... 37
3.2.3.A – Introduction to Microprocessors & Microcontrollers ..... 37
3.2.3.B – MCU Functional Requirements ..... 38
3.2.3.C – MCUs ..... 38
3.2.3.D – Sensors ..... 40
3.2.3.E - Liquid Crystal Displays ..... 41
3.2.3.F - Programmer Module ..... 42
CHAPTER 4 PROJECT MANAGEMENT ..... 43
4.1 - Research on Development Models ..... 43
4.1.2 – Software Development Models ..... 43

4.1.2 – Evaluation & Conclusion ..... 47
4.2 – Iterative Development Model ..... 48
4.3 – Project Risk Analysis ..... 50
4.4 – Time Management ..... 52
CHAPTER 5 SYSTEM DESIGN & DIAGNOSTICS ..... 53
5.1 – System Design ..... 53
Cycle 1 – HRM System Prototype ..... 53
Cycle 1 – Evaluation ..... 63
CHAPTER 6 ADA INTEGRATION & TUTORIALS ..... 64
6.1 – AVR-ADA Framework Design & Implementation ..... 64
Cycle 2 – AVR-ADA cross compiler setup ..... 64
Cycle 2 – Evaluation ..... 79
CHAPTER 7 SYSTEM INTELLIGENCE ..... 80
7.1 – Software Characteristics ..... 80
7.2 – Iterative Design Principle Continued ..... 83
Cycle 3 – One Second Delay ..... 83
Cycle 3 – Evaluation ..... 85
Cycle 4 – ADC Setup ..... 86
Cycle 4 – Evaluation ..... 93
Cycle 5 – LCD Setup ..... 94
Cycle 5 – Evaluation ..... 100
Cycle 6 – Pulse Detection Algorithm ..... 101
Cycle 6 – Evaluation ..... 107
Cycle 7 – Timer Setup ..... 108
Cycle 7 – Evaluation ..... 114
Cycle 8 – Safety & LED ..... 115
Cycle 8 – Evaluation ..... 121
Cycle 9 – SPARK Tools Integration ..... 122
Cycle 9 – Evaluation ..... 125
CHAPTER 8 EVALUATION ..... 126
CHAPTER 9 Conclusion ..... 127
CHAPTER 10 REFLECTION ..... 128
REFERENCES ..... 129
APPENDICES ..... 134
Appendix 1A – Ada Continues to revolutionise the world ..... 134
Appendix 1B – Ada continues revolutionising the world ..... 135

Appendix 2 – Software Capability Maturity Model ..... 136
Average Defect Density of Delivered Software ..... 136
Appendix 3 – SPARK 3 Level Analysis (Course Notes) ..... 137
Appendix 4 – SPARK's Design Methodology Diagram ..... 138
Appendix 5A – SPARK2C Solution Email (A) ..... 139
Appendix 5B – SPARK2C Solution Email (B) ..... 140
Appendix 6 – Soldering ..... 141
Appendix 7A – LCD (Implementation) Package ..... 142
Appendix 7B – LCD (Specification) Package ..... 144
Appendix 7C – LCD (Wiring) Package ..... 145
Appendix 8A – Wrapper Class (Implementation) Package ..... 146
Appendix 8B – Wrapper Class (Specification) Package ..... 150

# LIST OF FIGURES


Figure 2.1.1.1 – Saunders Diagram 12
Figure 3.1.1.A1 - Boeing 777 Jet Plane 1990 17
Figure 3.1.1.A2 – Siemens ILTIS Railway Traffic Control System 18
Figure 3.1.1.A3 – Clinical Analyser 19
Figure 3.1.1.B1 – DNS Server Performance 20
Figure 3.1.1.B2 – Tokeneer: Beyond Formal Program Verification 21
Figure 3.1.1.B3 - C-130J Super Hercules airlifter 23
Figure 3.1.2.A1 – Programming Language Comparison Chart 26
Figure 3.1.2.A2 – Ada & SPARK Relationship Diagram 27
Figure 3.1.2.A3 – Four Alternatives Beyond Software CMM 5 28
Figure 3.1.2.A4 – SPARK 3 Level Analysis 29
Figure 3.1.1.1.3 – Subset Comparison Chart 29
Figure 3.2.1.A1 – HRMs and POs Comparison 31
Figure 3.2.1.B1 – ECG Procedure & Display 32
Figure 3.2.1.B2 – ECG Monitors 32
Figure 3.2.1.C1 – Pulse Oximetry Revolution 32
Figure 3.2.1.C2 – PPG Method 33
Figure 3.2.1.C3 – PPG Representation 33
Figure 3.2.1.D1 – Resting HR Chart (men) 34
Figure 3.2.1.D.2 – Resting HR Chart (women) 34
Figure 3.2.3.A1 – Embedded Systems Encompasses the Globe 37
Figure 3.2.3.A2 – MCU Block Diagram 37
Figure 3.2.3.C1 – MCU Comparison 39
Figure 3.2.3.D1 – Table of Comparison 40
Figure 3.2.3.E1 – LCD Comparison Chart 41
Figure 3.2.3.F1 – USBASP Programmer 42
Figure 4.1.1 – Waterfall Model 43
Figure 4.1.2 – Iterative Model 44
Figure 4.1.3 – Spiral Model 45
Figure 4.1.4 – Agile Model 46
Figure 4.2.1 – Iterative & Incremental Development Model 48
Figure 4.2.2 – Pros & Cons 48
Figure 4.2.3 – Iterative Development Model Requirements 49
Figure 4.3.1 – Project Risk Analysis Matrix Diagram 50
Figure 4.3.2 – Risk Assessment Diagram 51
Figure 4.4.1 – Gantt Chart 52
Figure 5.1.1 – Systematic Process & User Interaction 53
Figure 5.1.2 – Circuit Block Diagram (Past Project) 55
Figure 5.1.3 – Improved Circuit Block Diagram 55
Figure 5.1.4 – Schematic Design 56
Figure 5.1.5 – USBASP Connections 57
Figure 5.1.6 – Circuit Design 57
Figure 5.1.7 – Software Implementation 60
Figure 5.1.8 – Testing Signal Representation (Before) 60
Figure 5.1.9 – Testing Signal Representation (After) 61
Figure 5.1.10 – Pure to Composite Transition 61
Figure 5.1.11 – Circuit Design Test 62
Figure 5.1.12 – MCU programmed 62
Figure 5.1.13 – LED Blink 62
Figure 6.1.1 – Flowchart 65
Figure 7.1.1 – Identifying Packages 81
Figure 7.1.2 – Software Design Principle 82
Figure 7.2.1 – Fuse Bits 83
Figure 7.2.2 – ATMega 328P Fuse Bytes 84
Figure 7.2.3 – Led_On: One Second Delay 84
Figure 7.2.4 – One Second Delay 85
Figure 7.2.5 – PRC Card 86
Figure 7.2.6 – ADMUX Register Byte 86

Figure 7.2.7 – ADCSRA Register Byte 87
Figure 7.2.8 – Bits of Interest & Procedure Initialize 89
Figure 7.2.9 – Procedure Start Conversion 89
Figure 7.2.10 – Procedure Conversion Complete 89
Figure 7.2.11 – Procedure Get Result 90
Figure 7.2.12 – Procedure Convert to Volts 90
Figure 7.2.13 – Specification File 91
Figure 7.2.14 – ADC_Main 91
Figure 7.2.15 – ADC Package Compiled 92
Figure 7.2.16 – Testing ADC_Main (1 - 2V) 93
Figure 7.2.17 – Testing ADC_Main (3 - 4V) 93
Figure 7.2.18 – LCD Wiring 94
Figure 7.2.19 – LCD Package Body 95
Figure 7.2.20 – LCD Specification File 95
Figure 7.2.21 – LCD Package Procedure Initialize 96
Figure 7.2.22 – Data Register Operation 96
Figure 7.2.23 – Entry Modes: Higher Nibble 96
Figure 7.2.24 – LCD Package Procedure Output 97
Figure 7.2.25 – Procedures Put, Command, Clear 97
Figure 7.2.26 – Procedure Display Integer 98
Figure 7.2.27 – LCD_Main 98
Figure 7.2.28 – LCD Main Compile 99
Figure 7.2.29 – LCD Integration Testing 99
Figure 7.2.30 – Rammer-Douglas-Peucker (RDP) Algorithm 101
Figure 7.2.31 – Threshold Analysis Algorithm 101
Figure 7.2.32 – Thresh Peak Algorithm 102
Figure 7.2.33 – Pulse Detection Algorithm Implementation 104
Figure 7.2.34 – Pulse Detection Compiled 105
Figure 7.2.35 – System Running Pulse Detection Algorithm 106
Figure 7.2.36 – Possible Malfunction 107
Figure 7.2.37 – PRC Card: Timer 108
Figure 7.2.38 – Timer Counter Register 1 B 108
Figure 7.2.39 – Timer Count Formula 109
Figure 7.2.40 – Timer Count 109
Figure 7.2.41 – Timer Setup Implementation 110
Figure 7.2.42 – Timer Unit Test Program 111
Figure 7.2.43 – Timer Setup Compiled 111
Figure 7.2.44 – Timer Setup Integration Testing Procedure 112
Figure 7.2.45 – System Running Timer Setup Unit Tests 113
Figure 7.2.46 – System Running Timer Setup Integration Tests 113
Figure 7.2.47 – PRC Card: Safety 115
Figure 7.2.48 – Resting Heart Rate (RHR) Averages 116
Figure 7.2.49 – States 116
Figure 7.2.50 – Safety Package (Body) 117
Figure 7.2.51 – Safety Package (Specification) 117
Figure 7.2.52 – LED Package (Body) 117
Figure 7.2.53 – LED Package (Specification) 117
Figure 7.2.54 – Controller Complete 118
Figure 7.2.55 – Safety and LED Package Compiled via Controller 119
Figure 7.2.56– Calibrate_System_UI Procedure 120
Figure 7.2.57 – System running all packages 120
Figure 7.2.58 – System Accuracy 121
Figure 7.2.58 – SPARK Integration Library 122
Figure 7.2.59 – SPARK Code Commented 123
Figure 7.2.60 – Wrapper Class Standard Procedures 123
Figure 7.2.61 – Wrapper Class Specification 123
Figure 7.2.62 – Wrapper Class Test 124

CHAPTER 1

# INTRODUCTION

Ada is fundamentally the largest and most rapidly growing programming language used for critical applications throughout the globe. Additionally, SPARK is the subset of Ada; a programming language used to formally verify and validate software automatically via theorem provers and SPARK examiner. Together, unimaginable systems have been developed and delivered on time. Furthermore, systems programmed and tested using SPARK-Ada are systems that are considered safe, reliable and dependable.

The project entails the production of a critical system used in the medical industry; a Heart Rate Monitor (HRM). Thus, SPARK-Ada was an essential choice towards the programming environment and infrastructure that the system should possess. However, the pervasive nature of microsystems is expanding gradually and globally alongside failing projects. Thus, expectations surrounding software designs and development, as well as the limited documentation posed by Ada in embedded systems pose major concerns.

Therefore, the thesis purely focuses on deliberating efficient proof of why Ada and SPARK surpass other programming languages in this particular field, how Ada can be implemented onto 8-Bit Microcontrollers and how SPARK can be used to examine the system through formal methods using SPARK Tools.

Moreover, deliberating clearly defined phases and well-structured tutorials, so that users or readers of similar interests can effectively execute it themselves in order to understand the fundamentals imposed in software development models and methods, system design concepts and project management techniques.

Ultimately, the thesis shall look to develop and possess a heterogeneous range of backgrounds from which all come together in a state of the art fashion to develop the system.

Upon completion, a fully functional and sophisticated HRM prototype should be presented in real-time.

# Project Goal & Objectives

### Goal

To develop a Heart Rate Monitor integrated with an Ada environment and critically test the system using SPARK tools and formal methods.

### Objectives

To accomplish the aim of the project a literature review, as well as background research must be conducted. Therefore, objectives were organised as a set of tasks in consequence to project completion as follows:

- Include a discussion encompassing the success of real-world projects deliberated by Ada, as well as SPARK.
- Deliberate an investigation on various programming languages used in embedded systems and provide sufficient proofs and benefits stating why the choice of Ada and SPARK would establish an efficient and coherent outcome on a system.
- Analyse components, tools and support available to deliberate the aim of the project.
- Design and develop a fully functional Heart Rate Monitor (HRM) system prototype.
- Build a framework to successfully install and integrate Ada support to program an 8-bit microcontroller.
- Subsequently, provide a solution to integrate SPARK Tools to verify and validate the system.

CHAPTER 2

# RESEARCH MANAGEMENT

## 2.1 – Research Methodology

Saunders Diagram also known as 'The Research Onion' is an ethnic model used for the organisation, as well as selection of research topics, which explicitly provides a divisive technique to design a state of the art research analysis. Thus, Saunder's Diagram (SD) has been utilised to emphasize the coherence in research design and fundamentally provide a well-structured and coherent piece of art.

### 2.1.1 – The Research Onion

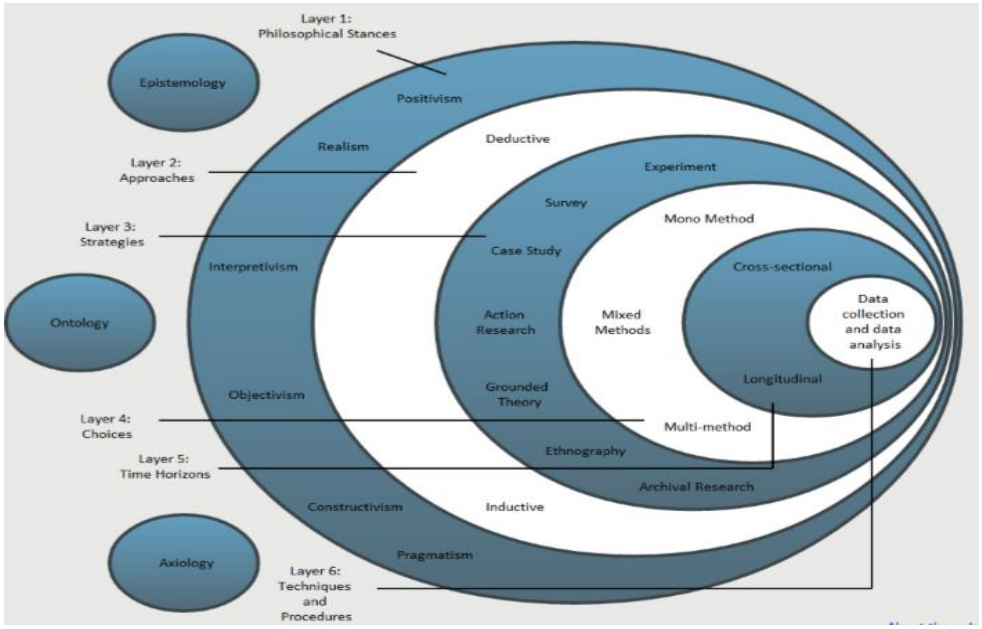
The Research Onion demonstrates the fundamentals of project design. Often, a project requires structure, methodical processes and most importantly background. Thus, background allows projects to innovate, expand and improve both structure, as well as methodical processes.

SD provides a structure that leads to a promising conclusion of background and theory. The structure entails the following components; each component is expressed metaphorically as an onion's layer, each layer provides a range of selective metaphysics (or paths) available to tackle and solve the research definition directly – Figure 2.1.1.1.

The research definition encompasses the following questions:

- Why is Ada and SPARK the elected programming language for this project?
- What security benefits or safe software solutions does Ada and SPARK impose?
- Why are HRMs considered as critical systems?

**Figure 2.1.1.1 – Saunders Diagram**




(Derby, 2009)

Additionally, SD dictates three different philosophies that a project manager should consider; Epistemology, Ontology and Axiology. One of these philosophies provide the first stepping stone towards the research definition and provides a level of guidance that prohibits the researcher from going off topic.

Subsequently, a personal review on all three philosophies was conducted; as a result, Epistemology was elected to procure the initial aim of the research definition.

**Epistemology**

Epistemology was selected for several reasons; mainly as it is a philosophy purely concerned on addressing facts and reviewing acceptable knowledge. This allows the author to define what would be classified as acceptable knowledge and what array of material is known to be true and treated as fact, subsequent to rigorous testing. A philosophical metaphysic commonly used in scientific research.

The research process was distributed after standardizing a formal approach towards the explanation of the research definition, while conforming to the Epistemology metaphysic. Thus, acquiring sufficient acceptable knowledge in the topic being investigated – **Ada** – **SPARK** – **HRMs**.

Research Distribution: Thoroughly analysing each layer of SD entailed efficacy and led to usage of multiple metaphysics in separate occasions.

**Philosophies – Layer 1 - Positivism – Realism**

The metaphysic **positivism** generates a hypothesis that in turn can be tested; reviewing and allowing explanations that conform to accepted knowledge. Thus, knowledge based theories that have been well defined, tested rigorously, proven and most importantly illustrated and considered fact.

Positivism generated a body of research that could be replicated by other researchers and conclude to same results.

Furthermore, since research is primarily based on software, realism was taken into consideration. As quantifiable results generated in software applications can very well be questionable in the case of new software solutions. However, to the present date the generated hypothesis includes acceptable knowledge that concluded with rigorously tested software, proven by countless entities and ISO certified for its critical participation.

**Approaches – Layer 2 – Inductive**

The approach of this research is defined as inductive, as it is purely based on creating theory. In prospective to the project, research was extremely limited. Thus, new observations, descriptions of analysis and software innovations towards the project must be discovered and employed.

**Strategies – Layer 3 – Case Study – Action Research**

Strategies are vital during the research process as it develops complete focus on related topics. Therefore, a Case Study design references background research based in a real-life context, and action research further justifies the issues involved and how to address them.

A case study design requires extensive research involving studies contributed by individuals based on real-life observations and experiences. In addition, to draw clear conclusions of research, the number of cases should be limited.

Action research allows the author to be part of a network that would benefit from a solution to given issues. The process involves a clear objective, which navigates to diagnosis of the problem and finally generates a list of actions to solve the problem.

**Choices – Layer 4 – Mixed-Methods**

A choice purely based on addressing quantitative and qualitative data is known as Mixed-Methods. In this project, both quantitative and qualitative data are being addressed. Furthermore, it has been considered throughout all layers of SD.

**Time Horizons – Layer 5 – Cross-Sectional**

The project comprises of both qualitative and quantitative research and measures multiple aspects of the research definition – **Ada** – **SPARK** – **HRMs**. Although, the project topic, as well as structure does not potentially conform to the definition of time horizons, cross-sectional best describes the research process shadowed.

**Techniques & Procedures – Layer 6 – Data Collection & Analysis**

Post to the fore-mentioned layers, a conspicuous approach towards the project solution must be formally documented. In essence, results of all research topics should generate clearly defined facts towards issues faced in today's software industry.

*Research Onion Information Reference:* (Derby, 2009)

CHAPTER 3

# FOUNDATIONS

## Research philosophy

The philosophy of the upcoming research constitutes on real-world practices. Software is generalised through condemned observations and formalisations of certifiable respects. Its pragmatic realism, determination in today's generation and revolutionising aspects of its constructive purpose has pursued a phenomenal mission in the foundations of business and organisations, life threatening machines and health and security prospects. Thus, failure of software is becoming common and uncontainable.

In addition, the software crisis is dramatically expanding from failing projects shadowing inconsistent software, management and time constraints. Time constraints is a concerning topic, as missions swell in complexity and deliverable time frames depress; alongside poor management programs and testing mechanisms bringing no surprise on the expansion of failing projects.

The International Standardization Organisation (ISO) is foremost responsible in rigorously testing and accepting software solutions; promoting them as safe and issuing the permissions for public launch.

Upcoming research focuses on real life experiences in mission-critical worlds or projects; Ada and SPARK covering the main foundations. Acceptable knowledge relates to project managers and personnel representatives who comment on their personal experience when pursuing critical missions. Missions that formulate decision making in project methodologies, programming languages and testing mechanisms that constitutes to revolutionising conclusions.

Moreover, technical comparisons on software will illustrate design structures and deficiencies, formulating acceptable knowledge and valid reasons why certain software infrastructures are not prepared to address the critical world.

Furthermore, the importance of HRMs should be clearly and formally deliberated.

# 3.1 - Literature Review

## 3.1.1 - Real-world Projects with Ada & SPARK Integration

Ada, in isolation has participated in multiple lines of work, revolutionising multiple aspects of the industrial world. In addition, SPARK has revolutionised the industrial world's safety and security of critical systems via adoption of highly efficient software-testing processes.

Thus, the upcoming sector purely focuses on Ada, as well as SPARK's contribution to the industrial world.

### 3.1.1.A – Ada's Lines of Contribution

**Boeing – 777 Jet Plane Project – 'Working Together' model**

The Boeing project was funded by a Seattle-based avionics company in 1990 and was fully functional 4.5 years later. Over 10,000 people were involved and were instructed that by "working together"; the highest quality in every part of the system could be accomplished. According to avionics software engineering manager Brian Pflug, "most companies disliked the idea of using a standard language at all, and seriously objected Ada as too immature" (Pflug, n.d., p. 1).

Honeywell, one of the project engineers decided to proceed with Ada after conducting an extensive study on the benefits of **Ada** and **C**. Results proved that Ada's built-in safety features would conclude to **less time**, **expense** and **concern** when debugging the software.

Program Manager of the 777's main electrical generating system, Dwayne Teske, quoted; "we had to start all over again, but the project went really smoothly after that, so Ada had a lot of positives" (Teske, n.d.).

The project was equipped with the following tools:

- Alsys' Ada development tools.
- AdaWorld cross-compilers with Smart Executive and Certification Package.
- DDC-I, Inc.'s Ada Compiler System – which allowed them to "build into the compiler a lot of optimization features specific to our hardware" as stated by Jeff Greeson, Honeywell's Project Leader.

After the success of the Boeing project, multiple suppliers involved continued using the language for other projects and enjoyed Ada's **portability** and **reusable** code. Furthermore, the Boeing project was delivered **on time**, at **low-costs** despite the conversion from 'PLM to Ada'.

Reference to this case study – (AdaIC, n.d.)

**Figure 3.1.1.A1 - Boeing 777 Jet Plane 1990**


**ILTIS- Railway Traffic Control**

Siemens Transportation Systems in Switzerland developed the ILTIS Railway Traffic Control. A Centralised Traffic Control (CTC) system primarily developed for the Swiss National Railways. The challenge they initially faced was "selecting a development language that meets requirements for **safety**, **availability**, **maintainability**, **portability** and **scalability**." (Rowden). The CTC system involved the listed functionalities.

- Fulfil safety related functions
- Prohibit life endangering decisions from faulty information presented
- Ensure that critical commands are not able to be inadvertently executed
- Ensure that critical commands are correctly executed

ILTIS is a Safety Integrity Level (SIL) 2 system. In 1990, Ada was their language of choice within reason.

The project was equipped with the following tools:

- Ada Programming Language
- ObjectAda – The environment of choice
- Windows after having switched from OpenVMS on Alpha platforms
- 20 – 30 engineers employed on the project to enhance the system

System requirements were subdivided into two phases;

- Safety-critical functions
- ILTIS Motif-GUI.

ObjectAda's static analysis capabilities highlighted many areas of deficiencies, undistinguishable by their previous compiler (DEC). The migration phase was accomplished within **budget**, **time-constraints** and benefited from enhanced **software quality**. By the year 2000, ILTIS was successfully running; authorised in Switzerland and Austria (went in service in 2006).

Neville Rowden, author of this fascinating case study quoted, "ObjectAda is being used as a powerful tool in migration of the final ILTIS-Features and has already demonstrated that the development environment using the command-line interpreter **scales up** effortlessly to larger projects" (Rowden, n.d., p. 2). Siemens Transportation Systems among others rely and depend on Ada for all their project successes.

Reference to this case study – (Rowden, n.d.)

**Figure – 3.1.1.A2 – Siemens ILTIS Railway Traffic Control System**

(Etienne, 2013)

**Ada Used to Develop Medical Analytical Systems**

Medical Analytical Systems are developed with high precision by Tegimenta Diagnostics Systems Division; "a 460-person subsidiary of Hoffman La Roche AG" (Institute, 1998), they are used in hospitals and laboratories to perform in vitro diagnostics worldwide. Ada was the chosen language aiming to provide control over "an integrated analytical system with a throughput of up to **750** tests per hour by use of Absorbency and Fluorescence Photometry, and Ion Selective Electrode" (Institute, 1998). The application contains up to 220,000 lines of code and was developed by a 10-person team during a 3-year period.

The project was equipped with the following tools:

- Alsys Development Environment,
- A proprietary GUI Builder
- Cadre TEAMWORK

They followed a Structured Design & Analysis process by Tom DeMarco using the Real-Time Modelling of Pages Jones. Martin Burri explained the reason why Ada was chosen; "Ada was initially chosen because it preserves our knowledge of programming languages like Pascal and Modula 2, it is more **reliable** than other languages because of the required **validation** of the compiler, and it proved (after comparisons with C) to provide significant **advantages** in terms of **software engineering**." The key benefits of Ada's **strong typing** were fundamentals to meeting the requirements of Tegimenta's compliance; ISO 9001 – Quality and Certification Requirements for the Development of Medical Instruments. Tegimenta confessed that their success was due to Ada and Alsys tools, and are convinced that Ada will be used in all their future projects.

Reference to this case study – (Institute, 1998)

**Figure 3.1.1.A3 – Clinical Analyser**


(Tecom, 2017)

**A Biomedical Engineer's View of Ada**

Robert C. Leif is a Biomedical Engineer from the Ada-Med Division of Newport Instruments. Leif explains that all medical devices include one significant software component; **Software Dependability**. Expressing that there are no programming languages or a complete solution that can provide concrete, dependable software. In contrast, Ada proved to be a starting point to ensuring a **concrete solution**. "The United States Department of Defence (DoD) certifies that any compiler it validates has passed numerous tests in the validation test suite." (Leif, 1993).

Furthermore, Mr Leif explicitly states that "The use of Ada for developing expensive, dangerous weapons and for projects requiring a high level of **safety**, such as air traffic control and aircraft has established an infrastructure of standards and experience for Ada programmers in **specifying**, **designing**, **creating**, and **testing** the software".

Moreover, when working on algorithmic projects for medical industrial applications, Mr Leif purchased the Alsys Ada Compiler for the development of a new mid-range haematology instrument, simply because it possessed significant training courses by Westley Mackey and Eugene Bingue in software engineering; Ada and AdaSAGE. Consequently, a "true 32-bit code under Microsoft Dos" produced by Alsys Ada did not require **overlays** (Leif, 1993).

Approximately, **three thousand** Ada Statements were written by two programmers who were familiar with Ada in a 2-year period. About **a thousand** of the Statements were **reused** on their second program. Although, Ada lacked support for third party vendors and outdated tools became an obstacle for an updated version of Alsys. Ada was considered as a significant stepping stone to their development, providing astonishing results that distinguished completion of the system's software before the hardware. Thus, "Both in terms of good **manufacturing practices** and of **cost savings**, Ada is, I believe, the correct language for **medical devices.**" (Leif, 1993).

Reference to this case study – (1993)

## 3.1.1.B – SPARK Ada's Lines of Contribution

**Ironsides – An Authoritative/Recursive DNS Server pair**

IRONSIDEs server was proved invulnerable to the known problems that plague other servers. That is, "remote code execution exploits and single packet denial of service", as explained by Mr Carlisle (Martin Carlisle, 2012).

IRONSIDES provides a service to the computer security community and was developed by the US Academy Airforce (Department of Computer Science).

The project was surrounded by the Ada programming language and SPARK 2012. The code was validated efficiently and the server was proved invulnerable by incorporating SPARK's formal methods in its design.

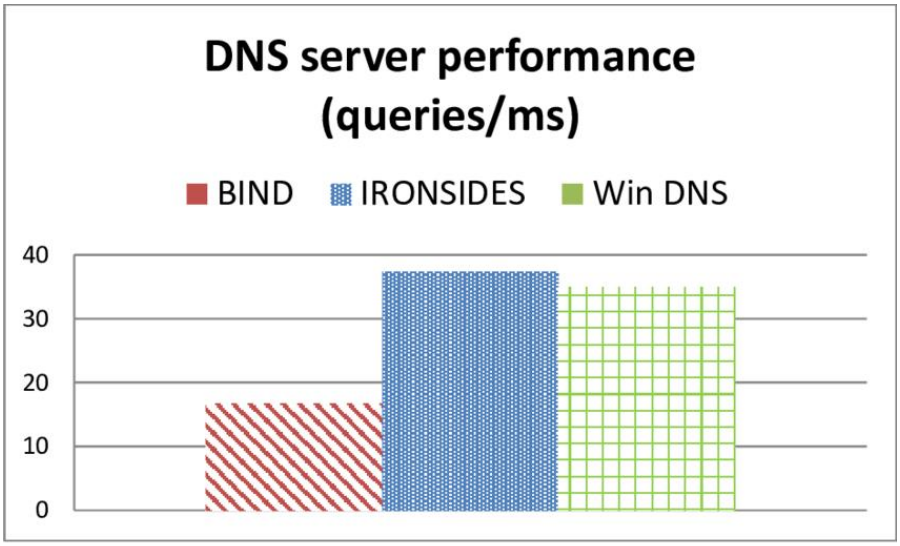
The results concluded the following:

- No data flow errors
- Exception error free
- No runtime errors – terminates only in the way the programmer explicitly states

All of which encompassed the desirable properties from a computer security perspective. The most common implementation of DNS is known as BIND, and IRONSIDE proved to be over **three times** as fast.

Reference to this case study – (Martin Carlisle, 2012)

**Figure 3.1.1.B1 – DNS Server Performance**




(Carlisle, 2014)

**The Tokeneer Project**

The Tokeneer project entailed a high security policy developed for trusted users when prototyping new Identification and Authentication (I&A) concepts, known as proof-by-concept security-critical control system. The system follows a unified security policy to protect the enclave perimeter and detect malicious threats. User's trust is enhanced by the system's confidentiality. Furthermore, an authentication procedure that utilises public key certificates and eliminates potential threats, while providing a **safe** and **secure** environment.

The Tokeneer project focused on one functional component of the Tokeneer system; the core functionalities of the Tokeneer ID Station (TIS). The development process was thoroughly summarised and Correctness by Construction (**CbyC**) methodology was applied. Having demonstrated relevant conditions to meet the Common Criteria standards, the project aim was fulfilled; a **safety-critical**, **low defect** and **cost-effective** software production was developed. As a result, faults and efforts were not fully addressed, assuming they had been, the system development would have satisfied the Common Criteria Assurance Evaluation Level 7, rather than Level 5 (standard AEL).
SPARK Ada tools verified vital conditions and validated sub-programs throughout approximately **ten thousand** lines of code, 9939 to be exact.

Overall, SPARK made software testing **feasible**, **direct** and **cost-effective**. Defects found per 1000 lines are currently unknown. Nonetheless, the 'PRAXIS' development process used was applied to commercial and government projects. Thus, their development process for high integrity applications concluded to an 'officially proven technology'.

***The above-mentioned project is an improved summary of a past coursework completed for CSCM10***

Reference to this case study – (David Cooper, 2008)

**Figure 3.1.1.B2 – Tokeneer: Beyond Formal Program Verification**

|   | # SPARK defects | # SPARK issues | # Ada defects | # Ada issues | # total (# exclusive) |
|---|---|---|---|---|---|
| GNAT | 1 | 1 | | 1 | 3 (0) |
| CodePeer | 1 | 4 | 3 | 1 | 9 (6) |
| SPARK tools | 2 | | | | 2 (1) |
| static analysis | 2 | 4 | 3 | 1 | 10 |
| hide annotations | | | | | 0 |
| loops | 1 | 1 | 1 | 1 | 4 |
| exceptions | | | 3 | 1 | 4 |
| other | 1 | 2 | | 1 | 4 |
| code review | 2 | 3 | 4 | 3 | 12 |
| total | 4 | 7 | 7 | 4 | 22 |

(Wallemburg, 2012)

**The Lockheed C130J**

The Lockheed C130j, also known as Hercules II Airlifter was an update of the world's most long-lived aircraft. Therefore, they planned to perform an enhancement of the aircraft "from the completely new avionics fit and new software that lay at its heart." (Amey, 2002).

Lockheed faced a critical mission, thus, a well-structured Correctness by Construction (CbyC) approach, based on the following was adopted:

- "Semi-formal specifications
  - Using Consortium Requirements and Parma's tables
- Thin-slice prototyping of high risk areas
- Template-driven approach to the production of similar and repetitive code portions
- Coding with SPARK
  - With tool supported static analysis carried out as part of the coding process and prior to formal certification testing."

(Amey, 2002, p. 2)

Coding with SPARK provided sufficient elimination of errors during the coding stage, prior to formal testing reviews – an approach that effectively brought Lockheed significant dividends. The most striking was the reduced **cost** of formal testing required for the DO-178B Level A certification. **Few errors** were distinguished during the upmost rigorous levels of Federal Aviation Administration testing.

As a result of Lockheed's testing and approach, it was proven that software quality had improved by a **factor of 10**, productivity by a **factor of 4**, saved **80 percent** in testing costs and **50 percent** on overall development. Subsequent to code reuse and process maturity, a further **factor of 4** was enhanced in productivity. In contrast, results were justified when tested by the UK Ministry of Defence, safety-critical errors were found by static analysis, properties of SPARK code and proof of exception freedom could be verified using Lockheed's semi-formal specification approach.

In conclusion, SPARK code obtained 10 percent of residual errors of the complete Ada code. However, residual errors originated from code written in the C programming language.

Reference to this case study – (Amey, 2002)

**Figure 3.1.1.B3 - C-130J Super Hercules airlifter**

(Corporation, 2014)

Thus, Ada, as well as SPARK have contributed to the success of astonishing projects, revolutionising the world in unimaginable aspects on multiple lines of industrial applications.

**Defence Strength Transportation Exploration Safety Trust & more**

*Appendix 1A exemplifies a range of recent wondrous projects developed in Ada and tested using SPARK tools. In addition, Appendix 1B exemplifies a wide list of revolutionising projects developed and tested using Ada's Security Features.*

## 3.1.2 – Programming Languages Used in Embedded Systems

**Assembly Language**

The Assembly language also abbreviated as ASM, despite what people in the software industry argue; the ASM is not a relic of the past. In almost all cases, it is not recommendable to program an application or system in ASM, however, it lies beneath most applications and computers. High level programming languages convert the code programmers can read to machine code (ASM) using an intermediate language. Understanding ASM positively impacts one's understanding of computer engineering (underpinning of compilers, assemblers, debuggers, linkers, and more). (Ebrahim, 2010)

ASM enables mutual communication with hardware and enables accurate timing physiognomies via instruction counts. However, ASM provides a serious overhead upon coding, debugging and maintaining; lacking on portability, readability, scalability and more. (Ebrahim, 2010)

**The C Programming Language**

The C programming language has an unforeseen legacy in embedded systems and is still growing strong. Will it vanish into thin air? If so, what other programming language would take its place?

'C' was born in the late 1960s, early 1970s. Designed as an imperative high-level programming language. Nowadays, high-level languages do not compare to what C offers – C is becoming the new Assembly. The harder it is to reuse, read and test code, the harder it is for the world of technologies to consistently evolve.

Nowadays, computer science courses are excluding C from being a paramount learning language. Intel's software engineering manager David Stewart quantified that "Computer Science curricula use Python as their introductory language" (Wilson, 2016) and that "the growing complexity of embedded algorithms is another force for change. As simple control-loops give way to Kalman filters, neural networks, and model-based control, high-performance computing languages." (Wilson, 2016) For instance, Python, Open Computing Language (OpenCL) and model-based environments such as MATLAB. However, a research study conducted stated that 95 percent of embedded systems run in C. (Wilson, 2016)

Evidently, C is available on virtually all platforms – one of its greatest advantages. Additionally, it's efficiency and support of low-level operations encompasses its infinite use on many systems, a born Legacy. However, the pervasive C is error-prone; immense work should be invested in debugging and run-in of deliverables. Even after the prototype has had its own run-through and debugging, it is time consuming. As stated by Lars Ole Anderson PHD, "errors not present in the prototype are likely to creep in, and debugging is an order magnitude harder" (1994). Thus, accomplishing high levels of efficiency is tough and then again, problematic. Consequently, the code is often indecipherable. Poorly understood to anyone but the original coder, and is subject to unexpected hardware dependencies. (Anderson, 1994)

**The C++ Programming Language**

For this reason, C++ came to light for embedded-system architectures. Software development can be structured using object-oriented programming (OOP). This diminishes the lack of reusable code and incorporates the following benefits:

- Classes
- Automatic Resource Clean-up
- Parametric Polymorphism
- Additional Type Safety
- More Expressive Firmware

(Kline, 2017)

Matt Kline a software engineer working for Fluke Network expresses that "Some language features depend on system facilities that we don't want to provide in embedded environments" (2017). He offers a solution to set-up a toolchain as a cross compiler using GCC, enabling their team to program with both C and C++ features. In conclusion, C++ encourages high-quality code structure, well-orchestrated assignment of responsibilities between components, reusable templates and is easy to read and support code. In contrast, automatic generation of class methods and implicit creation of objects affects performance; a crucial lack in efficiency.

**MISRA-C – C Programming Language Subset**

MISRA-C is a C subset, and is implemented using software development guidelines designed for the C programming language to enable more **reliable**, **safe**, **secure** and **portable** code, similar to SPARK. However, MISRA-C remains a working progress and does not ensure that it will fulfill its aims completely (Hills, 2014). Furthermore, Yannick Moy proves why SPARK 2014 supersedes MISRA-C, providing twenty-seven excellent examples of why and when C might make certain undecipherable rules hard to verify. Yannick Moy states that "It is therefore not surprising that almost all MISRA-C rule checkers, at the notable exception of PolySpace C Verifier, focus almost exclusively on decidable rules" (2013).

**Python Scripting Language**

Python is known as a scripting language, also characterized as an interpreter and as previously referenced, the most popular introductory programming language in Computer Science (CS); a statistic analysis accumulated throughout the US. This language emphasizes the principles of wise choices, as the benefits reflect strongly upon **writability**, **error-reduction** and **readability**. The advantage of Python's readability is exceptional as developing speed is enhanced, providing maintainable contributions to team-friendly environments and delivering quality assurance. However, since Python is an interpreter, processes are consumed with responsibilities, such as reading each line of code, parsing them, performing runtime checks and lastly, calling routines to execute coding operations. Hence, Python would make a selective programming language for embedded-systems if the embedded project had no effect to substantial reductions in efficiency during runtime, as well as higher energy consumption (Radcliffe, 2016). Furthermore, Anton G Setzer quoted that "Python would be

an ideal language if variables could be declared and if it had a 'type' system to it" (Supervisor, 2017).

**Ada Programming Language**

Ada was developed in the 1980s by the US Department of Defence (DoD). A mandate was proposed by the DoD to replace all military applications aiming to save vast capital and was dropped within a decade regarding both lack of support and the force of waves C++ and Java possessed (Ochem, 2017).

Fortunately, a new version was introduced prior to the revocation of the mandate and tool suites, vast support on modern architectures, as well as language enhancements were adopted by funding an open-source project to ensure a reference implementation was available. Thus, GNAT was introduced; *the complete development environment for Ada95 based on GNU Technology* (Ochem, 2017).

Ada begun to thrive and matured exponentially (Ada 2005, 2012 SPARK 2014). A comparison of Ada and other languages (including C) was conducted in the past, proving Ada had better qualities than all other languages – Figure 3.1.2.A.

**Figure 3.1.2.A1 – Programming Language Comparison Chart**

| | Structured assembler | C | CORAL 66 | ISO PASCAL | Modu-la 2 | Ada |
|---|---|---|---|---|---|---|
| Wild jumps | + | ? | ? | ? | ? | + |
| Overwrites | ? | - | - | ? | ? | ? |
| Semantics | ? | - | ? | ? | + | ? |
| Model of mathematics | ? | - | ? | + | + | ? |
| Operational arithmetic | ? | - | - | ? | ? | ? |
| Data typing | ? | - | ? | ? | ? | + |
| Exception handling | - | ? | - | - | ? | + |
| Safe subsets | ? | - | + | + | ? | + |
| Exhaustion of mem. | + | ? | ? | ? | ? | - |
| Separate compil. | - | - | ? | ? | + | + |
| Well understood | + | ? | ? | + | + | ? |

*{obtained from Anton G Setzer's course notes and previous CSCM10 assignment}*

Ada has had significant enhancements, and has contributed to large scale projects in safety critical environments. Significant comparisons have been conducted in the past prior to becoming part of high integrity real-time systems. Overall, Ada constituted astonishing feedback and results, such as making an impeccable difference to project deliverables. Other than this, Ada also provides the following advantages:

- Contains a well-defined, restricted and unambiguous structure
- Minimizes level of assembly, overall costs and development time
- Provides a starting point to a complete solution
- Detecting erroneous behaviour undistinguishable by other compilers

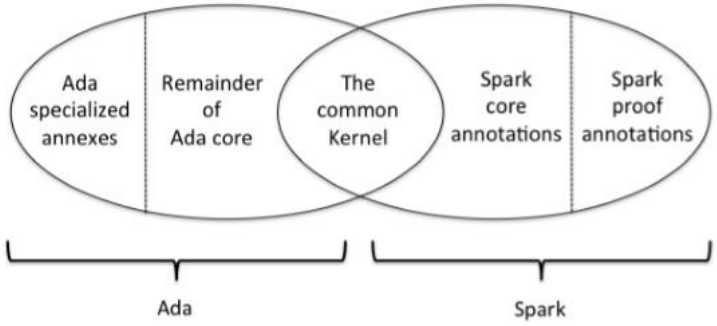
- Enhancing software quality
- Improving software testing and processes
- Maintaining system safety and security
- Permitting scalability towards larger systems
- Handling large quantities of data effortlessly
- Possessing an impeccable readable nature
- Delivering software dependability and portability
- Satisfying International Standardization Organization (ISO) requirements for the Development of critical applications including Medical Instruments (ISO 9001)

In conclusion, Ada's strong typing, readable nature, and the fact that it requires validation of the compiler, as well as enhanced security features accomplishes unimaginable results, which is verified by revolutionary projects such as those previously discussed.

**SPARK Programming Language – A subset of Ada**

The relationship of SPARK and Ada can be visualised using Figure 3.1.2.A2.

**Figure 3.1.2.A2 – Ada & SPARK Relationship Diagram**




Software testing has proved that on its own it cannot meet current, future and quality needs encompassing three disparate reasons:

- Complete software testing is impossible.
  - Addition of two 32-bit integers withholds over 2^64 (1.84467441E19) combinations for instance.
- Testing on its own does not provide the desired quality of software.
  - Users regularly find innovative and unintended ways of using applications.
- Hostile Users
  - Applications are under attack by people with bad intentions searching for untested data and execution paths to exploit.

(John W. McCormick, 2015, p. 6)

Consequently, Watts Humphrey declared a solution of four alternatives beyond Software CMM level 5 – Appendix 2. Figure 3.1.2.A3.

**Figure 3.1.2.A3 – Four Alternatives Beyond Software CMM 5**

| | |
|---|---|
| Clean Room | Focuses on defect prevention rather than elimination.<br>Achieved via a combination of applied formal methods in requirements, design and statistical testing.<br>10 times greater results than CMM level 5 is achieved. |
| Team Software Process (TSP) | A process based approach for defect prevention.<br>Quality results are more than 10 times better than CMM level 5. |
| Correctness By Construction (CbyC) | A software development process developed by Praxis Critical Systems.<br>Makes use of formal methods throughout the life cycle and uses SPARK for strong static code verification.<br>Quality results are 50 to 100 times better than CMM level 5. |
| CbyC in a TSP Environment | A process combining formal methods of CbyC using SPARK with process improvements of the TSP. |

(John W. McCormick, 2015, p. 6)

SPARK is a programming language designed to encompass formally defined semantics. It focuses on error prevention, rather than error detection and underwent through rigorous mathematical study. (John W. McCormick, 2015, p. 8)

The SPARK design was based on the Ada programming language. The reason behind using a subset of the Ada language encompassed the restricted, well-defined and unambiguous structure that Ada possessed. Allowing SPARK to eradicate features that could not be statically examined, such as:

- Recursion
- Dynamic Memory Allocation
- Dynamic Dispatch
- Access Types and Generics

(John W. McCormick, 2015, p. 8)

As well as, providing restrictions, such as:

- Array dimensions must be defined by previously declared type/s or subtype/s

Subsquently, add annotations to the Ada language, such as allowing:

- Specification of information and data-flow
- Creation of abstract functions & data types
- Use of structured assertions

***Last 4 points were retained from the presentation performed on behalf of CSCM10***

In Summary, SPARK tools trails three levels of analysis – Figure 3.1.2.A4.

**Figure 3.1.2.A4 – SPARK 3 Level Analysis**

**Level 1 Analysis**

- Correct Ada Syntax
- SPARK Subset of Ada Chosen (Pragma SPARK_Mode (ON or OFF)

**Level 2 Analysis**

- Data Flow Analysis
- Information Flow Analysis

**Level 3 Analysis**

- Generation of Verification Conditions
- Proof of Verification Conditions (Using automated and interactive theorem provers)

Reference: Anton G Setzer's Course Notes – Appendix 3

Recent studies express that the use of formal methods in conjunction to software testing further increases development costs and time; SPARK has proved otherwise. An article by Peter Amey states that "The exact semantics of SPARK require software writers to think carefully and express themselves clearly; any lack of precision is ruthlessly exposed by its support tool, the SPARK Examiner." (Amey, 2002, p. 3). As previously discussed, by using a well-documented CbyC approach, SPARK proved to have reduced the development testing time and costs by **80** percent (Amey, 2002, p. 5).

Verification and validation are paramount when developing software as it raises the question "Are we building the product right?" (John W. McCormick, 2015, p. 5). It demonstrates the correctness of software and reassures that the software meets user requirements.

Moreover, SPARK was compared with other subsets in the past and proved to be the most efficient subset of them all – Figure 3.1.1.1.3.

**Figure 3.1.1.1.3 – Subset Comparison Chart**

| | CORAL subset | SPADE-Pascal | Modula2 subset | Ada subset |
|---|---|---|---|---|
| Wild jumps | + | + | + | + |
| Overwrites | + | + | + | + |
| Semantics | + | + | + | ? |
| Model of mathematics | ? | + | + | + |
| Operational arithmetic | ? | + | ? | + |
| Data typing | ? | + | + | + |
| Exception handling | - | - | ? | + |
| Safe subsets | ? | + | + | ? |
| Exhaustion of mem. | + | + | ? | ? |
| Separate compil. | ? | ? | + | + |
| Well understood | + | + | + | + |

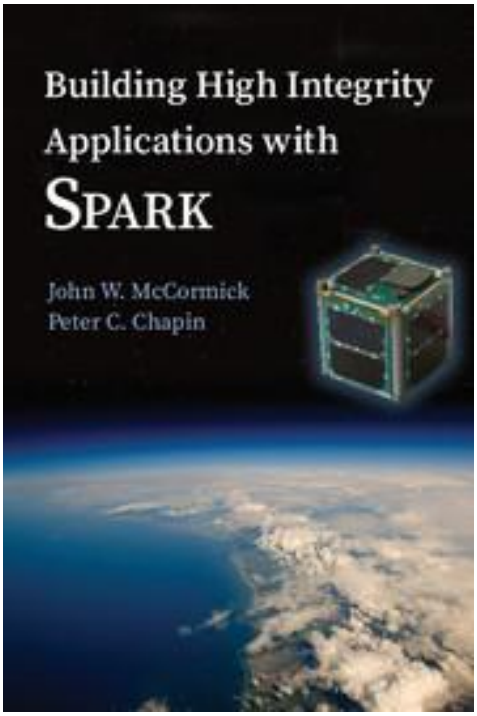
In conclusion, SPARK comprises a verification toolset, as well as a design method. Together, they both form a multiplex of benefits desired in high integrity critical applications – **Appendix 4.**

There are five versions of SPARK available; one for each version of Ada (83, 95, 2005, 2012 and SPARK 2014 which is based on Ada2012). SPARK 2012 and 2014 are exceptional in comparison to previous versions.

SPARK verification applies to explicitly defined contracts. By no means will SPARK detect deficiencies in compilers used to translate code into machine code, nor will it find defects associated to the operating system or hardware from which it runs, nor will it eliminate the verification testing completely. SPARK's benefits are fully dependent on the software developer's code construction and verification testing by the developer is required to determine that it runs correctly with the given operating system and hardware. Other than that, using SPARK correctly can solve the disputed Software Crisis. (John W. McCormick, 2015, p. 8)

(John W. McCormick, 2015)

John W. McCormick's latest book based on SPARK 2014.




(Altran, 2017)

# 3.2 - Background Research

## 3.2.1 - Embedded Systems

### 3.2.1.A - Introduction to Heart Rate Monitors

Heart Rate Monitors (HRM) and Pulse Oximeters (PO) is a vital component in the medical and sports industry, recognised as critical systems. A HRM or PO is a medical device utilised recursively by doctors, nurses, athletes, as well as a wide range of people across the globe for personal or professional uses. A multiplex of HRMs and POs have been introduced throughout the course of their history. They all vary in both specifications and shapes or sizes – Figure 3.2.1.A1.

**Figure 3.2.1.A1 – HRMs and POs Comparison**

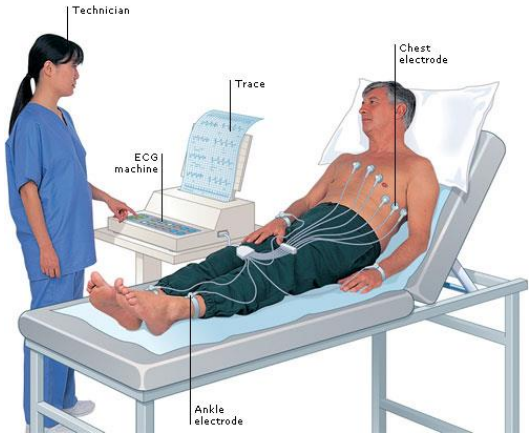
HRMs are recognised as HR detection systems commonly used for sports, and POs are HR detection systems commonly used for medical purposes. In conclusion, they both specify crucial evidence of a person's physical well-being and inaccurate results can lead to inappropriate treatments.

HR is directly related to health, it self-stabilises and varies from person to person. A person who attains good health acquires a strong heart. There are various methods to monitor a person's HR, each of which delivers accurate data regarding a person's physical welfare. The most common HR detection methods are electrocardiography (ECG) and photoplethysmography (PPG).

### 3.2.1.B – ECG

ECG exists as a non-invasive procedure discovered by Willem Einthoven in 1924. An ECG is achieved by using multiple skin electrodes adapted at different origins of the human body. Each electrode can detect the variations of electric current generated by electronic pulses responsible for heart contractions. The signal is filtered, amplified and displayed into an electrocardiogram for healthcare specialists to analyse – Figure 3.2.1.B1. (AB, 2002)

**Figure 3.2.1.B1 – ECG Procedure & Display**




(Diagnostics, 2017)

The system used is extremely old in comparison to ECG Systems available today – Figure 3.2.1.B2.

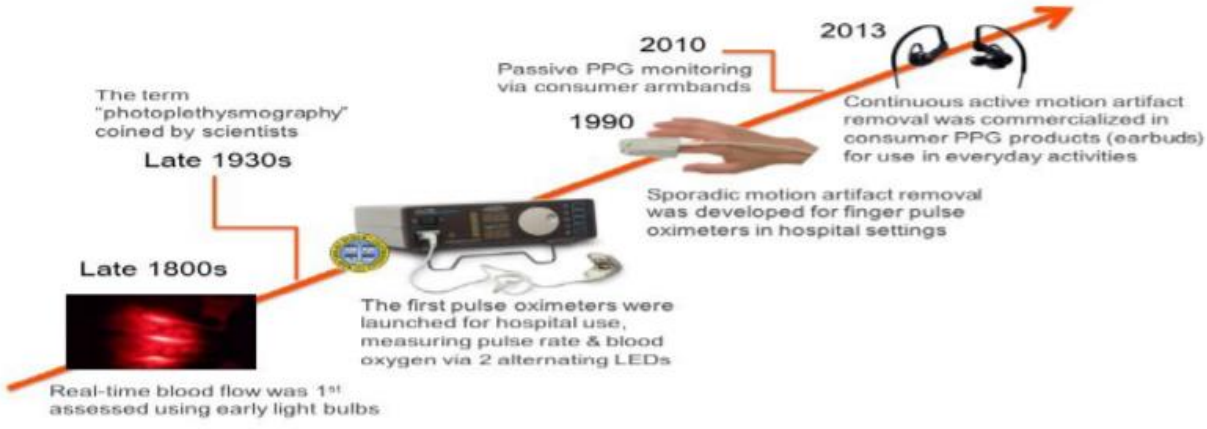
**Figure 3.2.1.B2 – ECG Monitors**


(Google, 2017)

3.2.1.C – PPG

PPG also exists as a non-invasive procedure, discovered back in the 1800s using light bulbs. However, pulse oximetry and PPG were fondly recognized as marketed products used for medical procedures in the 1980s – Figure 3.2.1.C1. (Sinex, 1999).

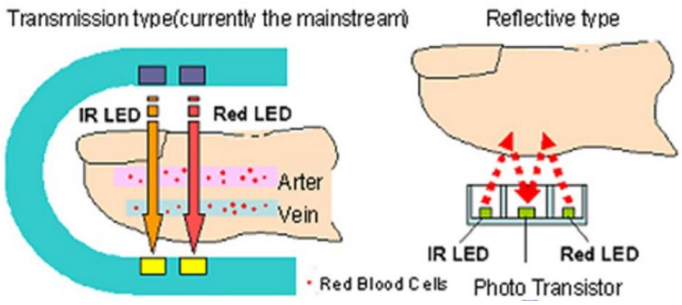
**Figure 3.2.1.C1 – Pulse Oximetry Revolution**




(Valencell, 2015)

Pulse Oximetry or PPG can be accomplished using an optical sensor. It consists of two components; a light emitting diode (LED with a specified wavelength) and a photo-detector (PD); both placed on a translucent measuring site (TMS). TMSs can be found in any of the following body parts – Earlobe, Index Finger, Wrist or Forehead. The LED generates a beam of light that penetrates the skin and the PD monitors light intensity variations caused by changes in blood volume, engendering a pulsatile physiological, analogous waveform attributed to the cardiac synchronous changes. Figure 3.2.1.C2

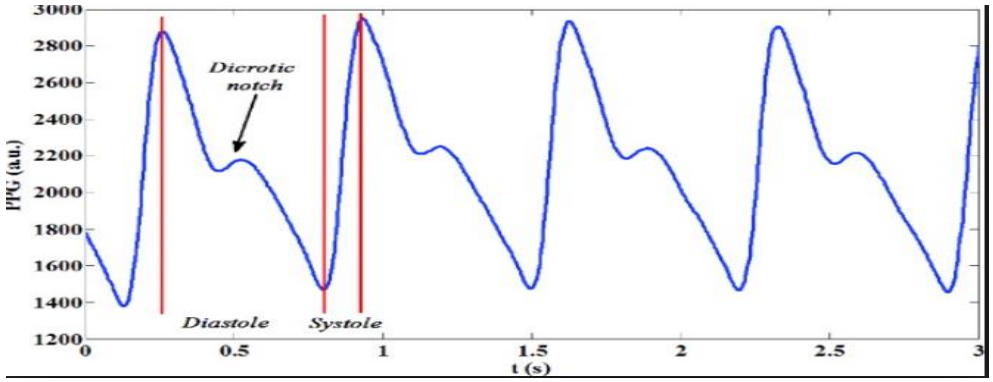
**Figure 3.2.1.C2 – PPG Method**




(Jake, 2014)

Figure C2 exemplifies an interesting fact work. Using two LEDs of different wavelengths – Infrared Range and 600 to 800nm, it can detect blood oxygen levels (SPo2) in conjunction to HR. The Red Blood Cells displayed is known as Oxygenated Blood or Oxy-Haemoglobin and is responsible for the reflectance of light. Figure 3.2.1.C3 illustrates the analogous representation obtained when analysing a PPG waveform.

**Figure 3.2.1.C3 – PPG Representation**




(Nina V. Sviridova, 2015)

The PPG characteristics exemplified are crucial to healthcare specialists; diastolic and systolic peaks, as well as dicrotic notch require thorough understanding. However, when analysing PPG signals in systems design, timing, cycles and peaks become crucial instead. PPG waveforms can be found in diverse, though, similar representations.

3.2.1.D – HRM Functionalities

HRMs are extremely scalable. In this section, HRM functionalities are going to be analysed as their significance is profound. Furthermore, the system proposed in this project will require this unforeseen knowledge to efficiently adopt certain functionalities. Since scalability is certainly available, functionalities will be adopted in series; considering the upmost significant first.

1. **Displaying Heart Rate (HR)**

Once the HR is displayed, it is vital to inform the user what the value of that HR truly means. In addition, since the system will be developed for medical purposes and not for training purposes, it is crucial to provide some sort of indication whether the displayed HR is safe or if medical attention is required. Thus, resting HR zones must be considered.

2. **Resting HR Zones – Indication**

Resting HR varies accordingly and is dependent on both fitness level and age as can be seen in figure 3.2.1.D1.

**Figure 3.2.1.D1 – Resting HR Chart (men)**

| Age | 18–25 | 26–35 | 36–45 | 46–55 | 56–65 | 65+ |
|---|---|---|---|---|---|---|
| Athlete | 49–55 | 49–54 | 50–56 | 50–57 | 51–56 | 50–55 |
| Excellent | 56–61 | 55–61 | 57–62 | 58–63 | 57–61 | 56–61 |
| Good | 62–65 | 62–65 | 63–66 | 64–67 | 62–67 | 62–65 |
| Above Average | 66–69 | 66–70 | 67–70 | 68–71 | 68–71 | 66–69 |
| Average | 70–73 | 71–74 | 71–75 | 72–76 | 72–75 | 70–73 |
| Below Average | 74–81 | 75–81 | 76–82 | 77–83 | 76–81 | 74–79 |
| Poor | 82+ | 82+ | 83+ | 84+ | 82+ | 80+ |

The above dataset was retained from a site called 'Top End Sports' (Wood, 2010). This site clearly justifies that the resting HR ranges between 49 and over 84 overall. However, a HR above 82 on average (for men) is considered as unhealthy (poor).

Figure 3.2.1.D2 below shows the same for women.

**Figure 3.2.1.D.2 – Resting HR Chart (women)**

| Age | 18–25 | 26–35 | 36–45 | 46–55 | 56–65 | 65+ |
|---|---|---|---|---|---|---|
| Athlete | 54–60 | 54–59 | 54–59 | 54–60 | 54–59 | 54–59 |
| Excellent | 61–65 | 60–64 | 60–64 | 61–65 | 60–64 | 60–64 |
| Good | 66–69 | 65–68 | 65–69 | 66–69 | 65–68 | 65–68 |
| Above Average | 70–73 | 69–72 | 70–73 | 70–73 | 69–73 | 69–72 |
| Average | 74–78 | 73–76 | 74–78 | 74–77 | 74–77 | 73–76 |
| Below Average | 79–84 | 77–82 | 79–84 | 78–83 | 78–83 | 77–84 |
| Poor | 85+ | 83+ | 85+ | 84+ | 84+ | 84+ |

In contrast, a poor HR for women is known as anything above 84 on average. Nonetheless, anything above 100 is considered unhealthy especially on completely rested individuals and medical attention should be sought. During exercise, your maximum HR is usually calculated as the following:

- **(220 – Age) = Maximum HR**

In conclusion, displaying heart rates accurately and advising individuals with rested heart rates of 100+ to seek medical attention are the two main functionalities that a HRM should obtain. Other functionalities involve the following:

3. **Blood Oxygen Levels (requires an additional sensor of a disparate wavelength)**
4. **Accelerometer (Training Purposes)**
5. **Daily Activity Monitoring (steps walked, calories burnt, and so on)**

3.2.1.E. - Ada Support for Embedded Systems

GNAT tools provide support towards the ARM Cortex-M0 (32bit processor). Unfortunately, the author experienced an unforeseen circumstance, GNAT does not provide complete support for (8 bit) microcontrollers. Therefore, after conducting research and reviewing the different solutions, the author got in contact with Adacore based in the US for advice. As a result, two separate solutions were discussed:

- An IDE available with a separate toolchain, known as SPARK2C. The toolchain is designed to execute SPARK Ada code for verification and validation purposes and when the system gets fully verified using SPARK, the code is then converted to C; at compile time. This toolchain is available, though, not affordable. The package is worth **$6000**, and includes:
  - IDE
  - Toolchain
  - Online support
  - Installable in 3 different operating systems

  **Appendix 5 (A) - Email**

- The alternative solution was a toolchain specifically designed to code with Ada 2005 (not SPARK compatible), developed in 2011. Price was not identified or discussed, the author assumed it would not be cheap either. The package includes:
  - IDE
  - Toolchain
  - Online support

  **Appendix 5 (B) - Email**

Not mentioned by Adacore, an alternative solution was found available subsequent to research. It offers **two** major challenges.

**Challenge 1**

Manually building (a framework) a toolchain that uses a cross-compiler known as *AVR-ADA* under Linux / Windows OS. Features based on AVR-ADA include: cross compiler, built-in libraries supporting AVR 8-bit microcontrollers and an online mailing list. Nevertheless, this approach has a series of disadvantages.

- Building a toolchain/cross-compiler is extremely tedious
- Research material is very limited.
- The process of it all can be unaccomplished.
- Uses Ada2005
- SPARK is not compatible
- No IDE

In contrast, the advantages:

- Affordable (£0)
- Support of a mailing list is available
- Ada programs integrated on 8-bit microcontrollers
- Available on both Linux / Windows OS

(Ebert, 2015)

**Challenge 2**

Developing a method that consists of using SPARK tools to verify and validate the code.

Advantage:

- The solution could emit to an innovative approach towards solid verification and validation of AVR-ADA programs.

Disadvantage:

- Failure to accomplish challenge emits to a HRM Prototype not fully verified or validated.

The following tools are required to set up the AVR-ADA toolchain under Ubuntu OS – Linux Distribution:

- binutils (assembler, linker, etc.), 2.24
- gcc (compiler), 4.9.2
- avr-libc (C run time system, start-up code, linker scripts), 1.8.1
- avr-ada (Ada run time system, gcc and binutils patches), 1.2.2

(Ebert, 2015)

In Addition, programming any program to a microcontroller requires the following program/application to be installed – AVRDude.

## 3.2.3 - Embedded Hardware

### 3.2.3.A – Introduction to Microprocessors & Microcontrollers

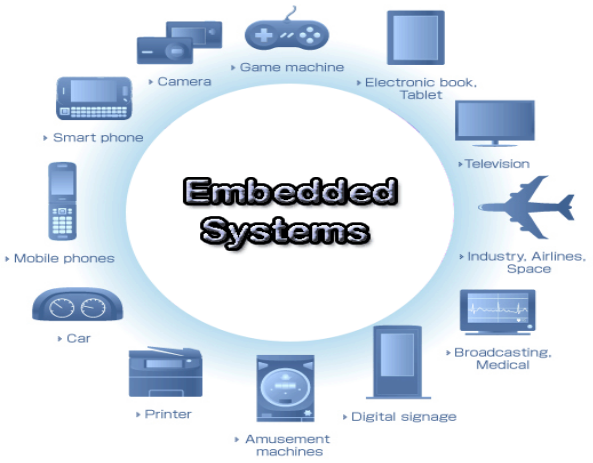
Microprocessor (MPU) and Microcontroller (MCU) units are not the same and are usually misunderstood. Nevertheless, they both have something in common – the Central Processing Unit (CPU).

- An MPU is a single chip CPU
- An MCU is a complete microcomputer system

(Singh, 2008)

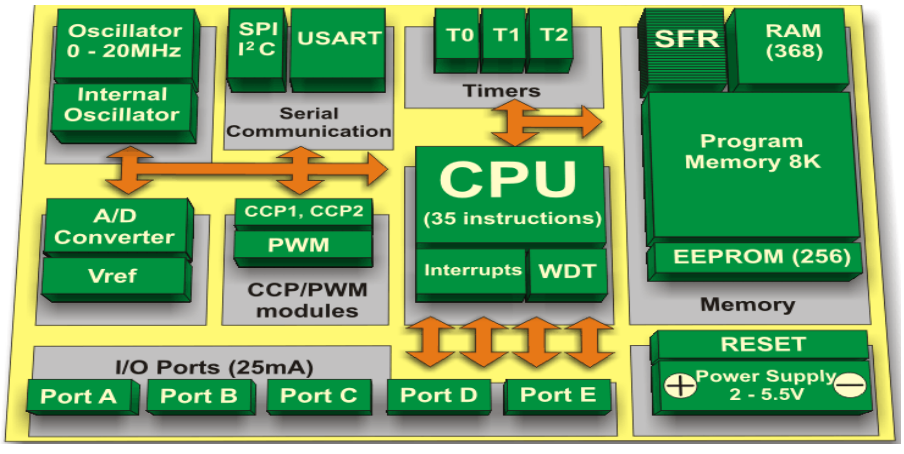
These components are crucial in our lives and in our world; their impact is profound. Unfortunately, they are frequently overlooked – Figure 3.2.3.A1.

**Figure 3.2.3.A1 – Embedded Systems Encompasses the Globe**




It is crucial to be aware of what drives these systems to perform the way they do. In this project, an embedded system will be developed (HRM prototype) and an MCU will be present to provide the correct functionality/operations to the system – Figure 3.2.3.A2.

**Figure 3.2.3.A2 – MCU Block Diagram**




(MikroElektronika, 2017)

The diagram exemplifies a sophisticated set of interconnected components that characterizes the internals of an actual computer and peripherals. In the preceding section, system components are briefly analysed and a thorough analysis is conducted on a series of MCUs, prior to electing the most suitable one for the job.

### 3.2.3.B – MCU Functional Requirements

The selected MCU must be/have:

- Ada compatible (AVR)
- Analogue Input (Pulse Sensor [x1pin])
- Digital I/O (LEDs x2-4 [x4pins], LCD Display [x4-8pins])
- Program Memory (32+ KB)
- RAM (2+ MBs)
- Sufficient MCU Speed (preferably 16 – 20MHz)
- Low power (5v)
- Affordable (<= £3)

In summary, components will use from 12 – 15 pins, eradicating pins used by ground(1-2pins), voltage input (1-2pins), reset (1pin) and external oscillator (2pins). Thus, an MCU with a total of at least 22pins is required.

Furthermore, future enhancements must be considered. For instance, accelerometer for steps monitoring. An additional 5 pins would conclude to flexibility and scalability of the system design.

Total pins essentially required – 27 at least.

### 3.2.3.C – MCUs

Section 3.2.3.B translates to the type of MCU required on the system. Before a comparison is conducted, it is worth mentioning that Atmel AVR MCUs are the only MCU family that support the software (Ada). Therefore, a series of Atmel's 8-bit MCUs will be analysed and compared – Figure 3.2.3.C1.

**Figure 3.2.3.C1 – MCU Comparison**

| Specification | AtMega128 | AtMega1280 | AtMega168 | AtMega32 | AtMega328P |
|---|---|---|---|---|---|
| Operation Voltage | 2.7 - 5.5V | 1.8 - 5.5 | 1.8 - 5.5 | 1.8 - 5.5 | 1.8 - 5.5 |
| Temperature Range | - 40 - 85 | -40 - 85 | -40 - 85 | -40 - 85 | -40 - 85 |
| UART | 2 | 4 | 1 | 1 | 1 |
| SPI | 1 | 5 | 2 | 1 | 2 |
| I2C | 1 | 1 | 1 | 1 | 1 |
| PWM Outputs | 7 | 15 | 6 | 4 | 4 |
| MAX A/D Res. (Bits) | 10 | 10 | 10 | 10 | 10 |
| A/D Channels | 8 | 16 | 8 | 8 | 8 |
| Internal Oscillator | Yes | Yes | Yes | Yes | Yes |
| MCU Speed (MHz) | 16 | 16 | 20 | 16 | 20 |
| Pin Count | 64 | 100 | 32 | 44 | 32 |
| EEPROM | 4096 | 4096 | 512 | 1024 | 1024 |
| RAM (Bytes) | 4 | 8 | 1 | 2 | 2 |
| Program Memory (KB) | 128 | 128 | 16 | 32 | 32 |
| CPU (Bit AVR) | 8 | 8 | 8 | 8 | 8 |
| Cost (£) | 6.38 | 5.97 | 1.6 | 3.03 | 1.13 |

| | |
|---|---|
| | Vital |
| | Chosen |

(Technology, 1998-2017)

Subsequently, the analysis concluded. The desired MCU required the following:

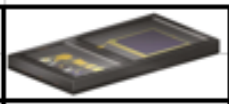
- At least 27 pins and must have enough to:
  - Power an LCD display
  - Read Analogue Inputs
  - Power 2-4 LEDs
- Furthermore:
  - Should contain a decent size of Program Memory and RAM
  - Should have pins to ensure Scalability and flexibility
  - Should retain low power consumption
  - Must be low in cost (affordable)

The selected MCU satisfies all the above and leaves room for scalability. Moreover, the AtMega328P has been used in multiple lines of projects in the past and sustains many positive reviews regarding its low power consumption and great performance. In addition, it is affordable, compact and withholds enough pins to ensure flexibility. Therefore, it was no doubt the best MCU for the job.

3.2.3.D – Sensors

A series of optical sensors will be analysed so that the pulsatile signal can be detected accurately and efficiently. The sensor that best suits the project will be selected; i.e. it must be compact, affordable and reliable – Figure 3.2.3.D1.

**Figure 3.2.3.D1 – Table of Comparison**

| Optical Sensor Specs | HSDL9100021 | OsramSFH7050 | Heart Pulse | OPB745 |
|---|---|---|---|---|
| Sensor Image |  |  |  |  |
| Pin Count | 4 | 8 | 3 | 4 |
| No. of Emitters | 1 | 3 | 1 | 1 |
| Emitter Range (nm) | 940 | 535, 660, 940 | 609 | 890 |
| Photo-Detector | YES | YES | YES | YES |
| Efficiency | High | High | High | Medium |
| Sensitivity | High | High | High | High |
| SNR | YES | YES | NO | No |
| Op. Voltage (V) | 1.5 | 2.1 | 3-5 | 5 |
| Temp Range (Deg) | -40 - 85 | -40 - 85 | -40 - 85 | -40 - 85 |
| Dimensions (mm) | 2.4x2.75x7.1 | 4.7x2.5x0.9 | 18x15x4 | 15.2x17.7x5.08 |
| Cost (£) | (x5) 7.41 | (x5) 9.38 | 4.95 | 2.09 |

| | |
|---|---|
| | Vital |
| | Chosen |

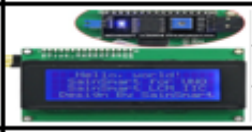
(Instruments, 1942 - 2017) (margi, 2017) (SPARKFUN, 2003 - 2017)

Concluding the analysis, the "Heart Pulse" sensor was selected as it comes factory ready to read the pulse and generate the signal to be fed into the microcontroller, constituting of several filtering and amplification techniques integrated and required on other sensors. Thus, making the system more compact and virtually less complex. Furthermore, the sensor is extremely sophisticated, affordable and saves a lot of time regarding system design.

3.2.3.E - Liquid Crystal Displays

LCDs are crucial in this project, since they are what users see and interact with. Therefore, the most suitable LCD would be one that could display relevant information to the user and must display content in a clear and visible format. A series of LCDs will be analysed and one shall be selected – Figure 3.2.3.E1.

**Figure 3.2.3.E1 – LCD Comparison Chart**

| | HD44780 | LCD-2004 Char | OLED-SSD1306 |
|---|---|---|---|
| LCDisplay |   |  |   |
| Character Display | 16x02 | 20x04 | 126x64 |
| Pin Count | 16 | 4 | 4 |
| Efficiency | High | High | High |
| Sensitivity | Medium | Medium | High |
| LED BackLight | White/Blue | White/Blue | OLED SelfLight |
| Contrast | Potentiometer | Potentiometer | N/A |
| Op. Voltage(V) | 5 | 5 | 3.3 - 5 |
| Temp Range (Deg) | -20 - 70 | -20 - 70 | -30 - 80 |
| I2C Limited Interface | No | YES | YES |
| PCB Dimension(mm) | 84x44 | 60x99 | 27x27x4.1 |
| Screen Dimension(mm) | 14.5x64.3 | N/A | N/A |
| Soldering Pins Req | YES | NO | NO |
| Cost (£) | 3.25 (Pins Inc) | 5.79 | 6.38 |
| | | Vital | |
| | | Chosen | |

(scooterboy101, 2017) (Mieiel6792014, 2017) (moore_estates, 2017)

It is worth mentioning that the analysis was quite contradicting, given the aims of the project and the revolutionised world of today, the OLED would have been the best choice regarding its sophisticated abilities and efficiency. However, the 16x02 display was selected due to the following facts:

- Personal Experience.
- The prototype focuses on the functionality of HR detection and software.
- No need to display a load of characters.
- Software challenges will be time consuming.
- Cost effective.

## 3.2.3.F - Programmer Module

In addition, a programmer module is required to program the microcontroller. The module illustrated in Figure 3.2.3.F1 was selected for several reasons.

- Personal Experience
- Affordable (£3.60)
- Easy Setup – (Electronics, 2015)
- In Stock

**Figure 3.2.3.F1 – USBASP Programmer**

(margj17, 2017)

In conclusion, crucial components were thoroughly analysed and decisions were deliberated with reference to personal experience, as well as beneficial facts. The vision attained from the results regarding the system prototype would comprise of the following fundamental components:

- **ATMEGA328P**
- **Heart Pulse Sensor**
- **HD44780 LCD 16x02 Display**

CHAPTER 4

# PROJECT MANAGEMENT

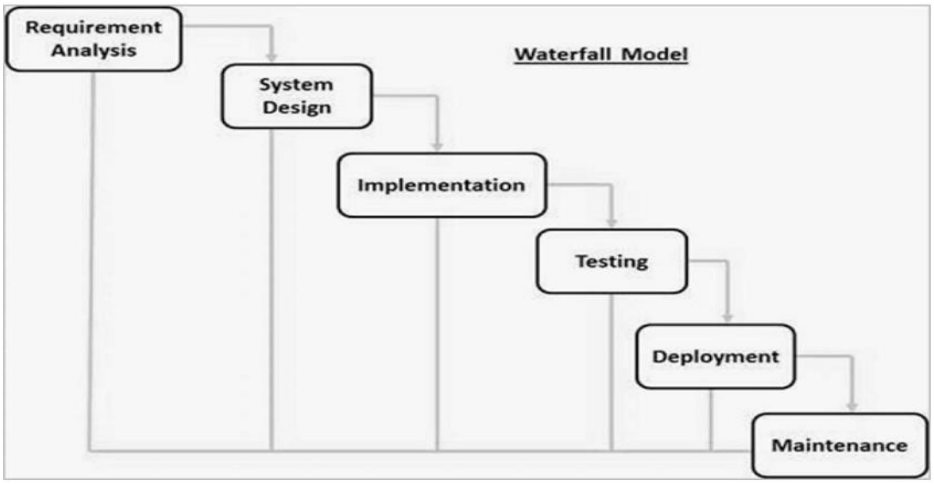
## 4.1 - Research on Development Models

As project manager and author of the ongoing project, development methodology has proven to outcast poorly organised industrial projects. Thus, emphasis of the importance in planning a development approach can't be expressed enough. As a result, Software Development Models have been formally addressed.

### 4.1.2 – Software Development Models

- **Waterfall Model -** (2017)

**Figure 4.1.1 – Waterfall Model**



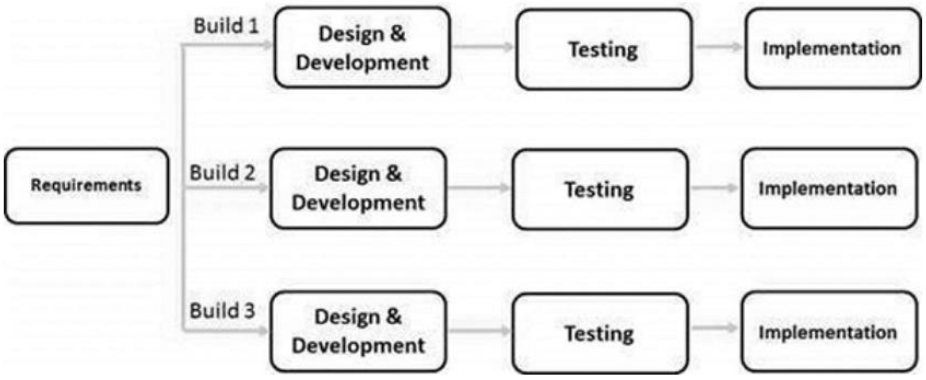

The Waterfall Model is recognised as the first model to ensure project successes. The whole development is organised into separate phases, where the output of one phase becomes the input of another. As a result of each cascaded phase, the process can be witnessed as a downwards flow.

However, the Waterfall Model is not made for unambiguous requirements or dynamic technology.

Nonetheless, it benefits from its simplistic design and manageable phases regarding specific deliverables and clearly defined stages. It works well on smaller projects, accommodated with clearly defined requirements.

- **Iterative Model -** (2017)

**Figure 4.1.2 – Iterative Model**




The development commences via a specification and implementation of just a simple part of the software, and is then reviewed to further justify other requirements. The process is repeated, producing a new version of the software at the end of each iteration. On a personal perspective, the development of this model demonstrates a realistic approach. However, the author believes that the implementation should be performed before testing.

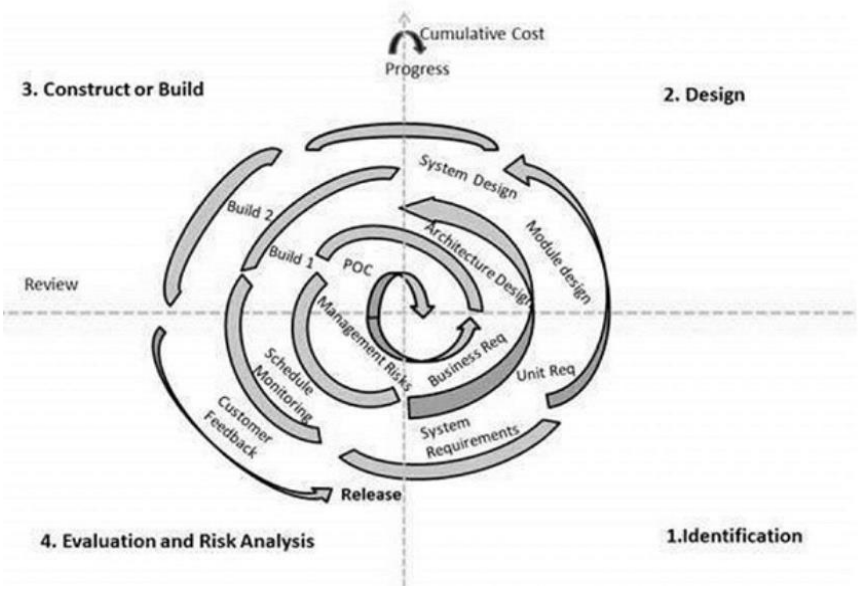
This method aims to develop a system through repeated cycles and small portions. Thus, inducing an iterative design and incremental build model for development. A process described as an evolutionary acquisition approach.

Iterative Model is successful in projects that entail rigorous validation of requirements, verification and testing of each version of software. Typically used on projects dealing with new technology, high-risk features and goals. It is well suited for large and mission-critical projects.

A common disadvantage faced in the iteration model is that system architecture or design issues may arise as not all requirements are gathered at the end of the first cycle. Furthermore, highly skilled resources are required.

- **Spiral Model -** (2017)

**Figure 4.1.3 – Spiral Model**




The Spiral Model is a combination of the iterative model and systematic controlled aspects of the waterfall model. This model poses high emphasis on risk analysis allowing incremental releases or refinement of the product on each consecutive iteration.

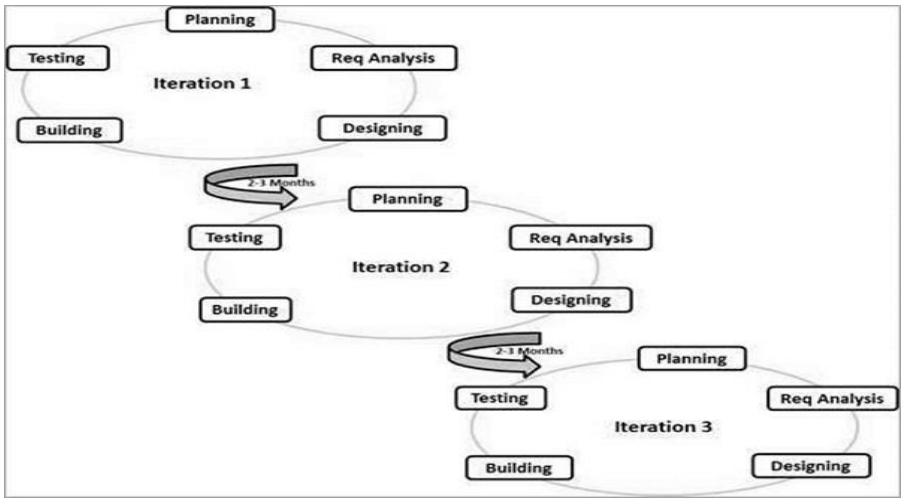
It is typically used on medium to high risk projects, where requirements are complex with the need of evaluation for clarity.

The Spiral Model allows different elements to be added when they become available or known and is consistent with approaches that require several software builds. In addition, it forces an early user involvement in the system development effort.

A major disadvantage towards this model is that the process may be spiralling indefinitely.

- **Agile Model -** (2017)

**Figure 4.1.4 – Agile Model**




The Agile Model treats every project differently and tailors existing methods to suit project requirements. After every iteration, working software build is delivered and the final build holds all the features required by the customer. It has become a very popular SDLC due to its flexibility and adaptability. Agile Model contains agile methodologies that have become popular for their unique focus and approach towards software projects such as Rational Unified Process, SCRUM, Crystal Clear, Extreme Programming and more.

It is typically used on projects that follow an adaptive approach, working hand-in-hand with their customers. A realistic approach to software development where resource requirements are minimum, planning is not really required and customer interaction is the *Backbone* of this methodology.

### 4.1.2 – Evaluation & Conclusion

As the project constitutes to several challenges that requires time and dedication. The Iterative Model provides an incentive towards the completion of disparate tasks/challenges, enforcing a divisive approach towards the conformant of large-scale, mission-critical projects.

The project was deliberated into separate entities. Each entity is completed and tested prior to the commencement of subsequent iterations or challenges. Every challenge builds from the last, incrementing the systems' intelligence further down the timeline.

This model provides a manageable, controlled and systematic approach towards the overall project solution. Moreover, the iterative method builds a solid organisation plan organised in smaller portions.

In large software projects, research proved that time keeping played a crucial part of project management for which many projects failed to keep. Thus, managing a set of "smaller projects" is wiser than managing a large project all at once. In essence, the iterative process focuses on small projects that build up to a large one systematically.

The Spiral Model would have been a great choice regarding the underlying project, however, the author believed that the Iterative Model provided an effective systematic approach that the Spiral Model lacked in. Therefore, the Iterative model was elected.

The Waterfall Model would have worked in a smaller project and the Agile Model would have been a good choice in a customer oriented environment. Since the underlying project is large and not customer oriented, it was easily discarded from the decision.

# 4.2 – Iterative Development Model

Distributed research on Development Models constituted of four distinct models commonly used in software projects. As a result, the most effective model for the underlying purpose was the Iterative & Incremental Development Model – Figure 4.2.1 and Figure 4.2.2 deliberates the pros & cons of the elected model.

**Figure 4.2.1 – Iterative & Incremental Development Model**

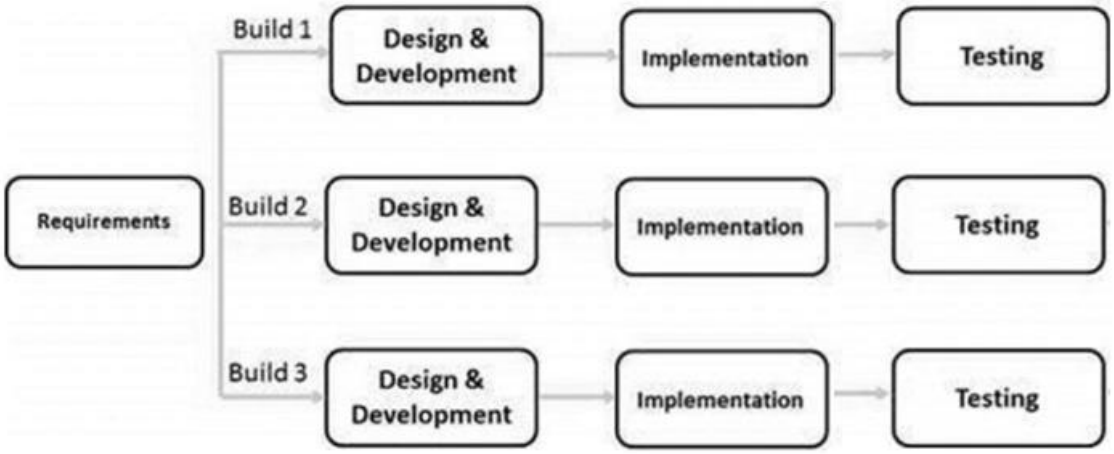



**Figure 4.2.2 – Pros & Cons**

| | |
|---|---|
| **Advantages** | ➢ Induces a working model of the system at an early stage. |
| | o Makes it easier to find functional / design flaws |
| | o Enables corrective measures at an early stage |
| | ➢ Working functionality is developed quicker and earlier as opposed to other software development models. |
| | ➢ Results are produced at an early stage and periodically. |
| | ➢ Software development is flexible and can be planned in parallel. |
| | ➢ Progress is Measurable. |
| | ➢ Changing requirements or scope are less costly and supported. |
| | ➢ Testing and debugging are more feasible during smaller iterations. |
| | ➢ Resolution of identified risks are performed during iteration. |
| | ➢ Every iteration or increment delivers an operational product. |
| | ➢ Time of operation is reduced |
| | ➢ It is suitable for large, mission-critical projects. |
| | ➢ Risk analysis is more beneficial. |
| **Disadvatages** | ➢ May require additional resources. |
| | ➢ Changing requirements are supported but not suitable. |
| | ➢ Management complexity swells. |
| | ➢ Management attention is paramount and heavily required. |
| | ➢ Design issues may appear as not all requirements are always gathered. |
| | ➢ Not suitable for small projects. |
| | ➢ End of project may be unknown. |
| | ➢ Risk analysis may require highly skilled resources. |

(Point, 2017)

The Iterative Development Model poses visions that other models lack from a personal perspective. In reality, a project should be developed constructively and incrementally. Therefore, a critical examination of the project infrastructure formulates to the following foundations:

- A fully functional prototype used for testing and quality purposes.
- A framework that enables compatibility with the prototype's programmable component, enforcing the criticality of software by means of revolutionising conclusions previously addressed.

The development of these foundations is crucial to the success of the project. Without possessing solid foundations the project is a failing project. Thus, complete focus was formulated to these two distinctive requirements.

Addressing each requirement articulates to distinct cycles that in essence, build upon previous cycles (if any). Thus, the project's infrastructure articulates to the cycles or requirements listed – Figure 4.2.3.

**Figure 4.2.3 – Iterative Development Model Requirements**

| | |
|---|---|
| **Cycle 1** | Consists of the System Prototype Circuit Design, tested using a compatible programming language ©. |
| **Product** | System prototype running a program that blinks LEDs |
| | |
| **Cycle 2** | Consists of the AVR-ADA Cross-Compiler, Library Set-up Tutorial and Installation. Tested by combining AVR-ADA with the hardware. |
| **Product** | System running an AVR-ADA program from which LEDs responds to. |
| | |
| **Cycle 3** | Refined software design with Improvements - 1 Second Delay |
| **Product** | System flashing LEDs every 1 Second |
| | |
| **Cycle 4** | Refined software design with Improvements - ADC Setup |
| **Product** | System reading ADC pulsatile signal and tested using LEDS |
| | |
| **Cycle 5** | Refined software design with Improvements - LCD Setup |
| **Product** | System running and tested by displaying strings on LCD |
| | |
| **Cycle 6** | Refined software design with Improvements - Pulse Detection Algorithm |
| **Product** | System detecting peaks on an estimated timeframe of 10 seconds, tested by displaying BPM on LCD. |
| | |
| **Cycle 7** | Refined software design with Improvements - Timer Setup |
| **Product** | System running a timer on CTC mode, tested using LEDs and adopted to Pulse Detection Algorithm. |
| | |
| **Cycle 8** | Refined software design with Improvements - Safety & LED |
| **Product** | System running successfully and tested by displaying useful advice on completion. In addition, an LED Package will be included to avoid the controller from accessing the MCU directly |
| | |
| **Cycle 9** | Verifies software design using SPARK Tools |
| **Product** | System fully verified and validated. |

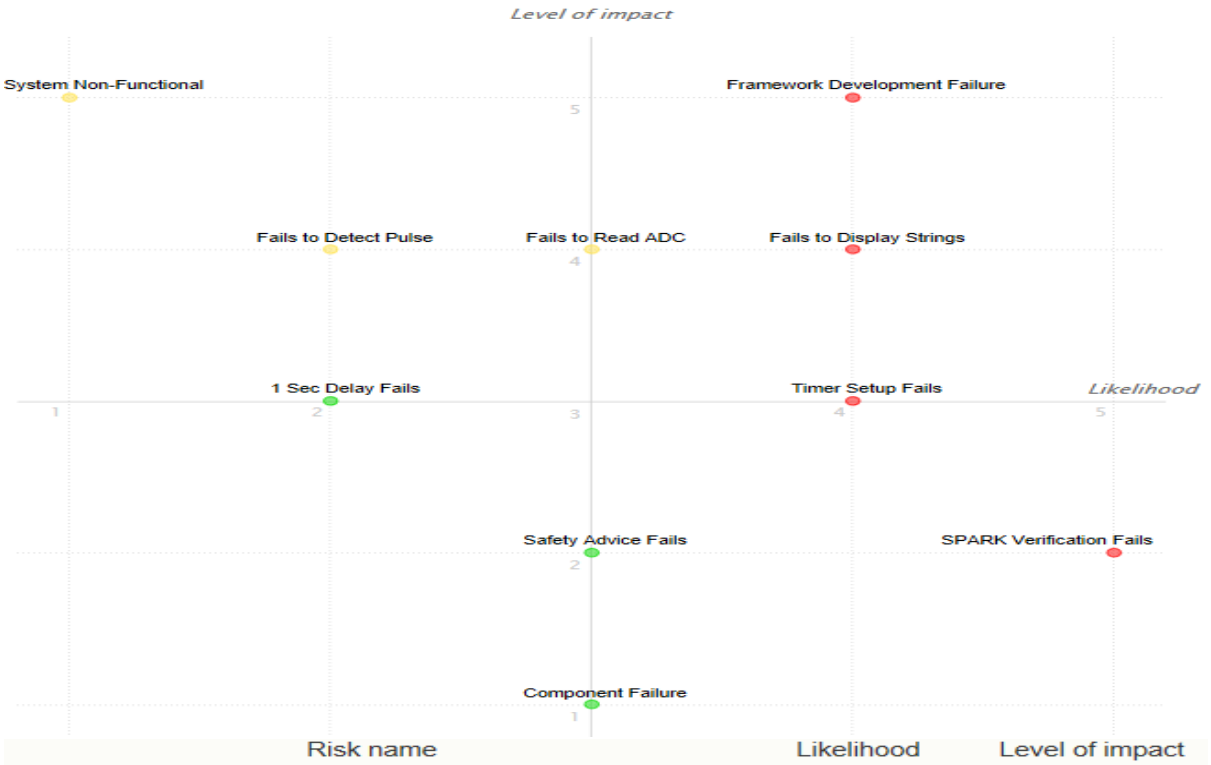
Figure 4.2.3 illustrates a clear structure of the requirements associated to the project. Cycle 1 and Cycle 2 are paramount. Cycle 3 and onwards incrementally builds software and is tested prior to initiation of the next cycle. Thus, each cycle finalises with rigorous testing and a description of the developed product; underlining the fundamentals of an evolutionary acquisition approach.

# 4.3 – Project Risk Analysis

Risk analysis is crucial during project management as it enhances the level of awareness and determination when undertaking a task that withholds an expected risk. A risk is defined as a situation or an event that may have a negative impact on the delivery of the project.

Primarily, identifying the risks is fundamental, that is the reason why a risk analysis matrix diagram was designed. The matrix deliberates the level of impact it has on the system and the likelihood of that risk occurring – Figure 4.3.1.

**Figure 4.3.1 – Project Risk Analysis Matrix Diagram**



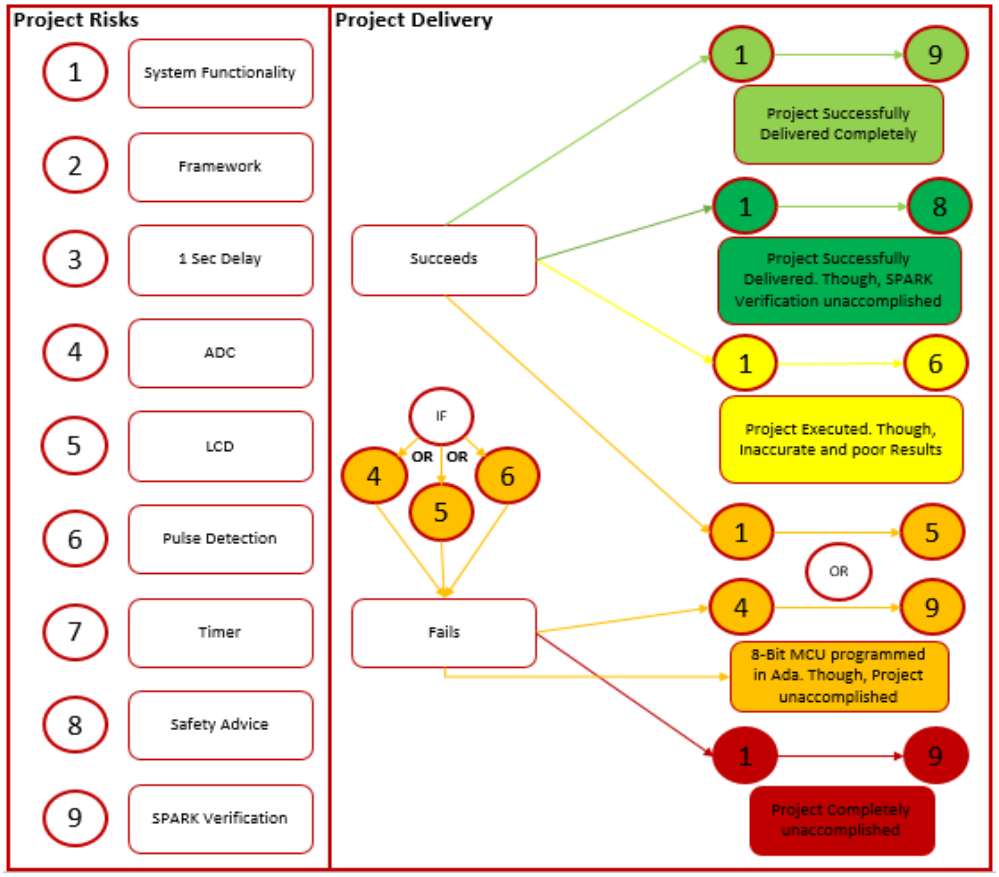

| Risk name | Likelihood | Level of impact |
|---|---|---|
| System Non-Functional | 1 | 5 |
| Framework Development Failure | 4 | 5 |
| 1 Sec Delay Fails | 2 | 3 |
| Fails to Read ADC | 3 | 4 |
| Fails to Display Strings | 4 | 4 |
| Fails to Detect Pulse | 2 | 4 |
| Timer Setup Fails | 4 | 3 |
| Safety Advice Fails | 3 | 2 |
| SPARK Verification Fails | 5 | 2 |
| Component Failure | 3 | 1 |

(beeye, 2017)

The Project Risk Analysis deliberated results that pose major threats to project deliverables. The key risks that should be avoided is clearly displayed at the top end of the chart. Avoiding those risks is purely dependent on the project developer's enthusiasm, motivation and problem-solving techniques.

Nonetheless, it seems as most risks have a critical impact on project delivery. The reason why this is true is because the project encompasses the fact that it can't be tested from a software perspective. Results are purely tested on the prototype. Thus, a personal risk assessment was designed to clearly justify the success of project deliverables depending on what elements are accomplished – Figure 4.3.2.

**Figure 4.3.2 – Risk Assessment Diagram**



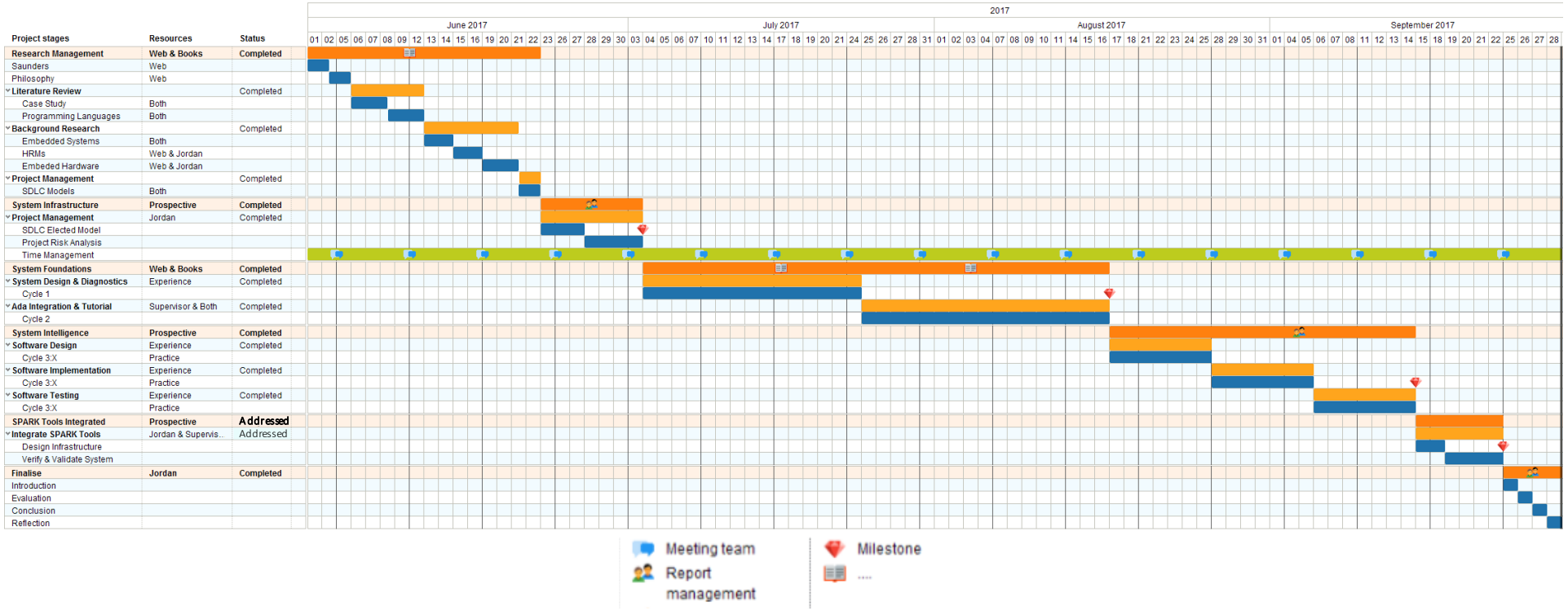

One element was discarded from the observation and not all possible states were addressed. Component failure was discarded as it can easily be replaced. All states do not require their presence as the diagram covers crucial states that effectively speak for the rest. In addition, they are colour coordinated and correspond to the associated or same coloured box.

The aim of the project, with reference to the risk assessment is to accomplish the colour green. The lighter green would result as an exceptional accomplishment, meaning that SPARK tools integrated and successfully verifying the software can constitute to an innovative solution in the world of microcontrollers. However, both greens are exceptional accomplishments regarding project delivery as research verified that no person has yet been able to accomplish that on microcontrollers in general.

# 4.4 – Time Management

Time managing a large project has proved its vitality in multiple sectors of this thesis, prioritizing cannot be expressed enough. Thus, Figure 4.4.1 expresses a well-structured Gantt Chart encompassing all crucial sectors within the project – Timeline: June 1st – 28th September 2017.

**Figure 4.4.1 – Gantt Chart**




There are three extensive timelines, Research Sector, System Foundations and System Intelligence, three assets that allows the project to be completed to an exceptional level if accomplished. Research, meetings, milestones and report management are all expressed through icons. Being guided by this Gantt chart throughout the whole project enhanced confidence and projected visions regarding the remainder of the project towards completion.

CHAPTER 5

# SYSTEM DESIGN & DIAGNOSTICS

## 5.1 – System Design

The project is based on a Heart Rate Monitor design, an 8-bit microcontroller will be programmed using the Ada Language. Ada programs will be designed using the AVR-Ada library, previously discussed as the source used to access and manipulate AVR microcontroller's CPU and peripherals.

In summary, the main challenge of this project consisted of a fully functional system design, as well as, a framework designed to allow compatibility of Ada programs with AVR MCUs. Thus, adopting the Iterative Model to these set of goals:

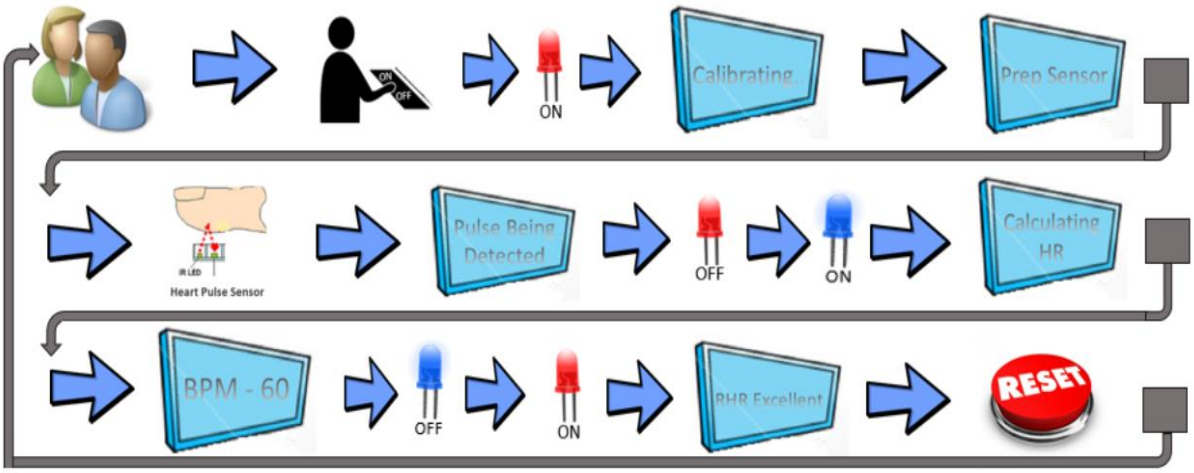
Cycle 1 – HRM System Prototype

- Consists of the System Prototype Circuit Design, tested using a compatible programming language ©.
- **Product**: System prototype running a program that blinks LEDs.

**Cycle 1 – Design: System Design Requirements**

The aim of the prototype is to provide a direct and informative user experience. Furthermore, allowing users to easily understand the system's functionality and provide relevant information to result with minimum user interaction. A diagram following a systematic process and minimum user interaction was designed – Figure 5.1.1.

**Figure 5.1.1 – Systematic Process & User Interaction**




The prototype's systematic process in Figure 4.1.1 was deliberated with complete focus on what a user would expect from the system, as well as, what type of information would be vital for users from such system. The prototype focuses solely on HR, thus, providing users with accurate information will critically examine their physical welfare.

In addition, following the displayed diagram, the user would only interact with the system three times. Moreover, no specific input is required from the user. Thus, user interaction is minimum.

1. **Activating System**
2. **Positioning Sensor**
3. **Resetting / Deactivating System**

The systematic process illustrated can thereby be used as a reference throughout all cycles of project design/management. It is crucial for functional requirements to be engraved prior to developing or implementing software, as it would avoid confusion and time wasting thoughts regarding how the system should function. Hence, engraving what the system does restricts the project to specific priorities, enabling a developer's complete emphasis towards managing goals.

As previously discussed, the prototype requirements concluded the following elements:

- ATMega328P MCU

MCU will be used to distribute system intelligence accordingly, with reference to the developed software.

- HD44780 - 16x02 Liquid Crystal Display (LCD)

The LCD should inform the user with instructions throughout its functional process. Such as:

  - When the system is calibrating.
  - When to position sensor prior to performing a BPM calculation.
  - When the BPM calculation is about to begin.
  - BPM results – Beats Per Minute.
  - What their results mean.

- Heart Pulse Sensor – Fully Integrated

To provide minimum user interaction, the following components will benefit the system:

- An ON/OFF Switch

An ON/OFF switch would provide users with direct access to activate/deactivate the system.

- A Reset Button

A Reset Button allow users to recalculate their HR or BPM, for reassurance or new BPM reading.

- Two LEDs – Red and Blue

LEDs determine the state of the system at every stage of the systematic process.

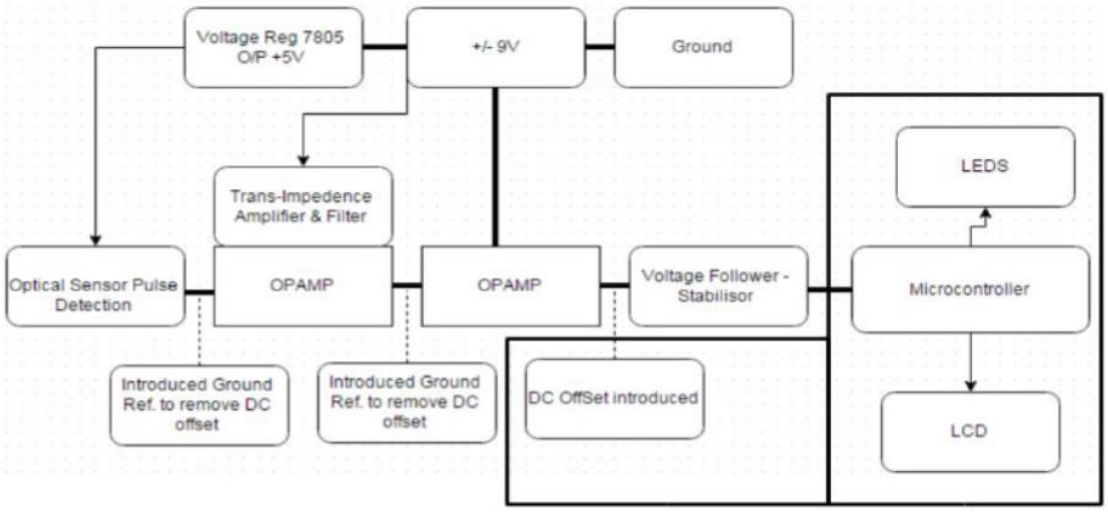
  - RED stating that the system is calibrating or awaiting a pulse prior or subsequent to performing a BPM calculation.
  - BLUE stating that the system is reading and performing a BPM calculation.

Knowledge of the ATmega328P datasheet is an absolute necessity. The schematic must be carefully constructed to avoid unnecessary errors when programming the MCU.

A schematic design was developed using Proteus 8 professional, Atmega328P and HD44780 datasheets and a block diagram designed in the past. In addition, personal experience gained from past projects was employed – Figure 5.1.2.

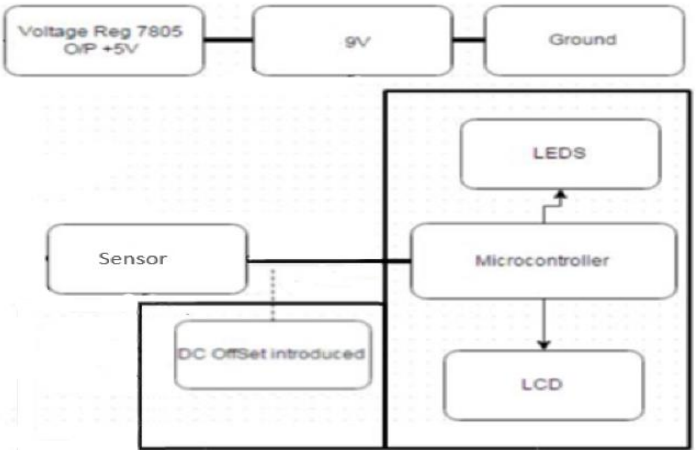
Datasheets: (Corporation, 2017) (Hitachi, 2017)

**Figure 5.1.2 – Circuit Block Diagram (Past Project)**



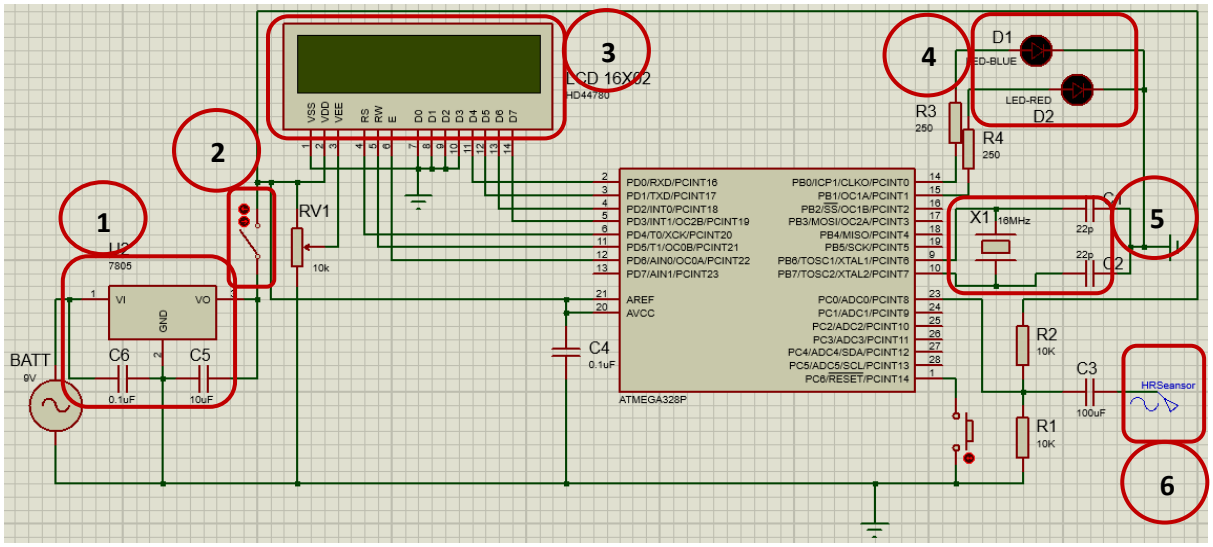

This block diagram references a circuit that was designed and developed by the author of this thesis. The black frame visible on the diagram are changes that were required on the circuit to make it compatible with today's existing project. However, the circuit was redesigned with direct reference to the above-mentioned requirements. Consequently, the circuit was significantly improved and significantly reduced – Figure 5.1.3.

**Figure 5.1.3 – Improved Circuit Block Diagram**




The circuit schematic design was deliberated and examined using the above circuit block diagram. Furthermore, component values were withdrawn from the ATmega328p Datasheet, LCD Datasheet, personal circuit design experience and formulas – Figure 5.1.4.

**Figure 5.1.4 – Schematic Design**

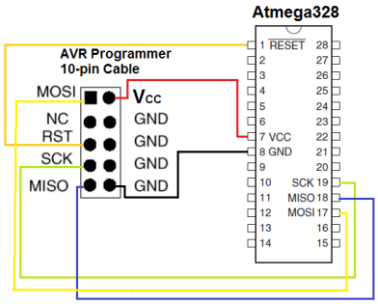



The circuit design consists of the following:

- (**1**) Voltage regulator is used to restrict the circuit supply voltage of **9V**olts to **5**V.
  - Protects the MCU and the LCD which have a **5.5** max Voltage.
  - Two Capacitors of **0.1uF** and **10uF** are used in the design to filter out noise/interference generated from the voltage reduction.
  - Values have been obtained from past experience.
- (**2**) A switch is used to allow users to activate the system when necessary.
- (**3**) An LCD is connected to the MCU using **4Bit** mode; **D4:7** (on LCD) as stated on LCD datasheet.
  - Uses PORTD as I/O ports and a variable resistor (RV1) of **10KOhms** to adjust the brightness of the LCD.
- (**4**) Two LEDs (Red/Blue) are connected to MCU's **PORTB0:1**. LEDs go connected to **250Ohm** resistors and Ground (**GND**).
  - The MCU will trigger an LED by writing a 1 to the specified PORTB**0/1** which generates a voltage.
  - Resistor values can range from **100Ohms** to **350Ohms**; emitted light strength will be affected with high resistance.
- (**5**) An External Crystal Oscillator (**X1**) circuit has been designed and connected to the MCU using two **22pF** capacitors as stated in the ATmega328P datasheet. The crystal oscillates at a frequency of **16MHz,** thus, **0.0625uS** per clock cycle.
  - 1/Freq = Time
  - The MCU depends on X1 for timing events.
- (**6**) The Heart Pulse Sensor represents an AC (alternating current) signal in the schematic. It uses a **100uF** Capacitor to eliminate all DC (direct current) interference and a potential divider to control the signal in conformance to the MCU.

In addition, a programmer called **USBASP** is used to program the microcontroller. Therefore, cables were employed on the circuit with reference to the USBASP programmer datasheet and looks like the following setup – Figure 5.1.5.

**Figure 5.1.5 – USBASP Connections**




(Electronics, 2015) (Protostack, 2015)

**Cycle 1 – Design: Partial Software Design**

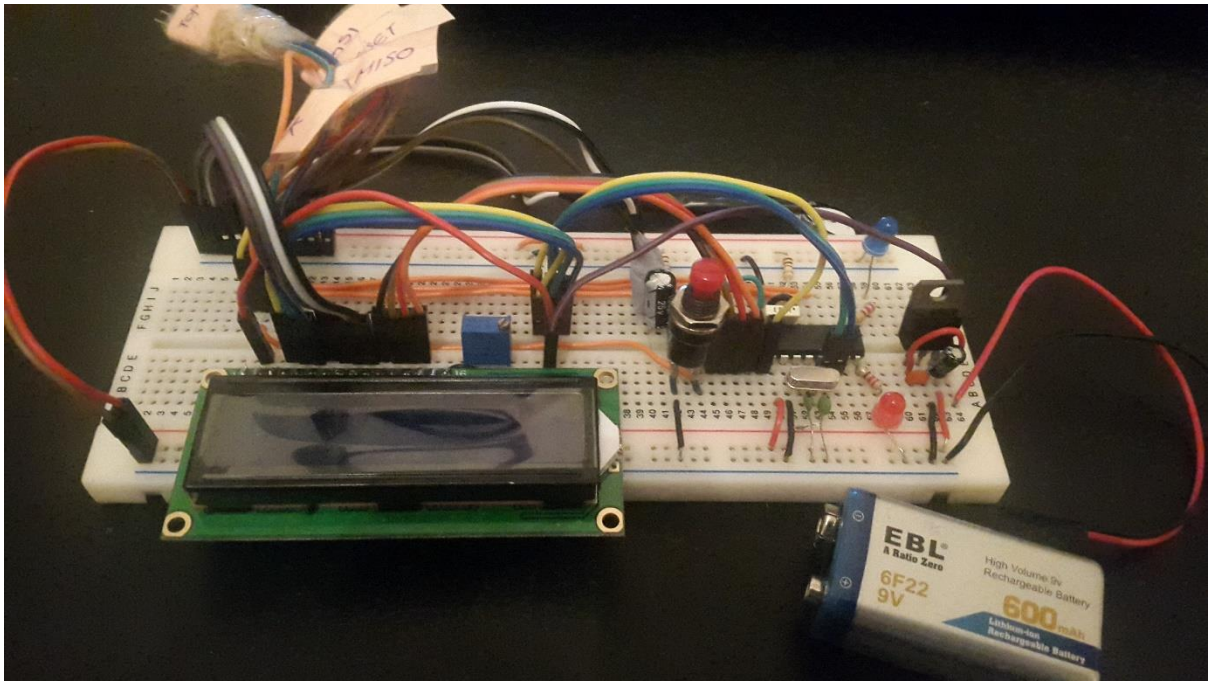
Pseudo code:

```
Initiate PORTB as Output
While True
    PortB0, True
    PortB1, False
    Delay 1 Sec
    PortB0, False
    PortB1, True
    Delay 1 Sec
End While
```

**Cycle 1 – Implementation: Circuit Design**

Subsequently, the circuit design was developed with reference to the above schematic – Figure 5.1.6.

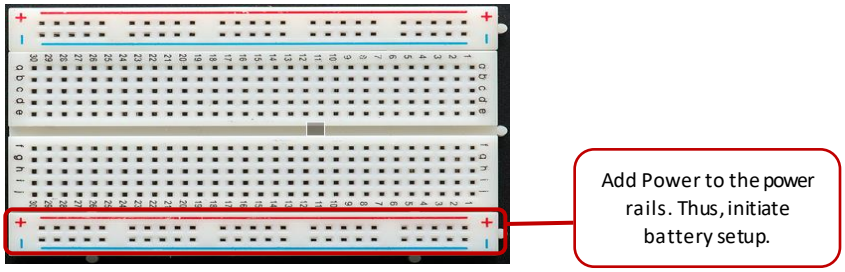
**Figure 5.1.6 – Circuit Design**



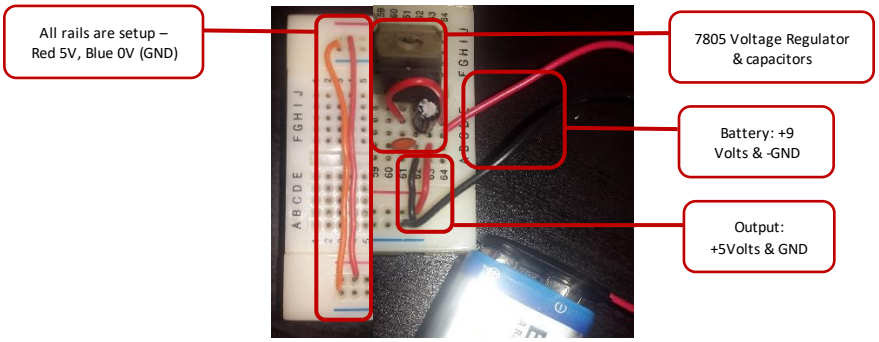
The Circuit Design exemplifies an exact representation of the above schematic. Although cables and setup looks slightly messy, it was the best way to distribute cables along one single breadboard, given the position of pins on the MCU. Nonetheless, the circuit was implemented in separate entities with meticulous attention subscribed.

A systematic approach was followed towards a practical setup attained from past experience, to avoid unnecessary discrepancies.

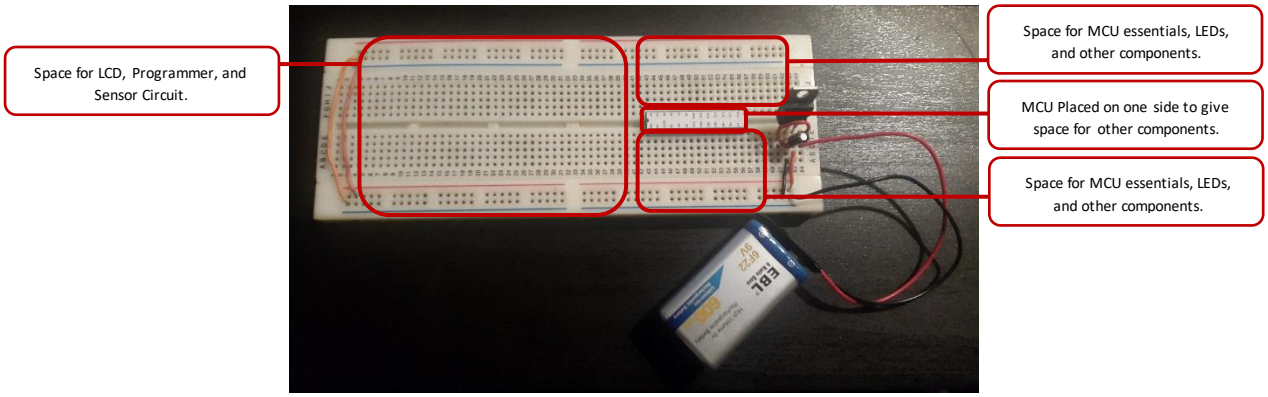
**Step 1**: Initiate power on the breadboard



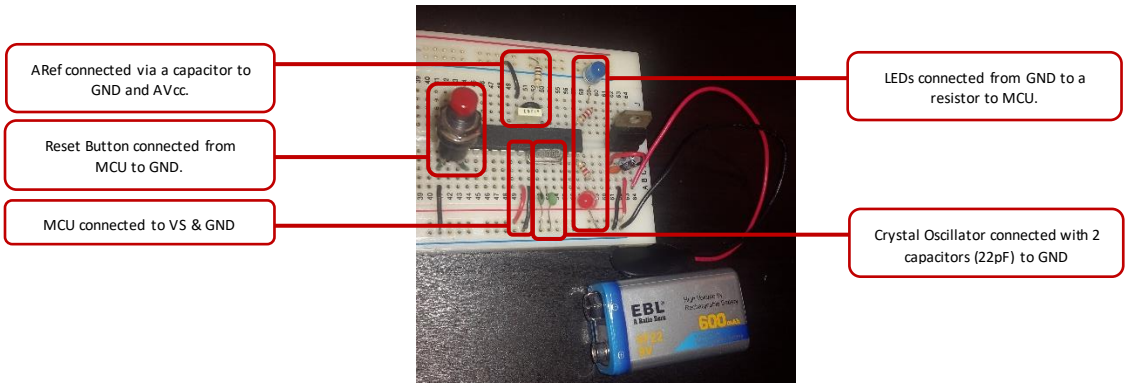

**Step 2:** Setup Voltage Regulator circuit to initiate a 5V voltage supply.



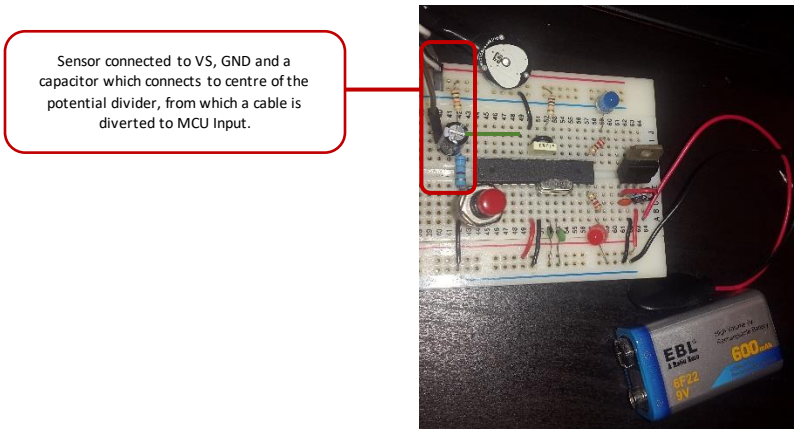

**Step 3:** Position the MCU in an appropriate area on the breadboard; MCU will not be moved once setup begins. Datasheet is used to identify pin characteristics.



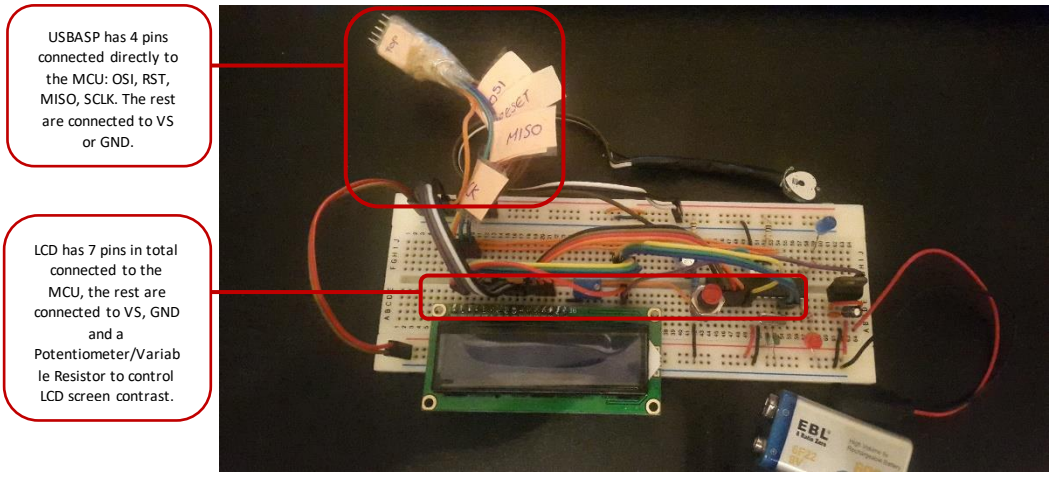
**Step 4:** Adopt components required to power-up and test the MCU: Crystal Oscillator circuit, MCU Voltage Supply, MCU GND and other components, such as: capacitor/s, resistors, LEDs and reset button.




**Step 5:** Include the Sensor, DC Blocker and Potential Divider design.




**Step 6:** Implement the LCD design and Programmer adapter. Soldering pins to the LCD was required – Appendix 6.



USBASP adapter cables were attached accordingly; resulting in a male socket with easy access.

**Step 7:** Testing.

**Cycle 1 – Implementation: Partial Software Design in ©**

A simple program was designed for testing purposes – Figure 5.1.7.

**Figure 5.1.7 – Software Implementation**

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB = 0x01; //set Data Direction Reg of PortB as O/P
    PORTB = 0;   //set PortB as Output

    while(1){
        PORTB = 0x01;       //Set PortB1
        PORTB = ~0x02;      //Clear PortB2
        _delay_ms(1000);    //Delay 1 Sec

        PORTB &= ~0x01;     //Clear PortB1
        PORTB &= 0x02;      //Set PortB2
        _delay_ms(1000);    //Delay 1Sec
    }
}
```

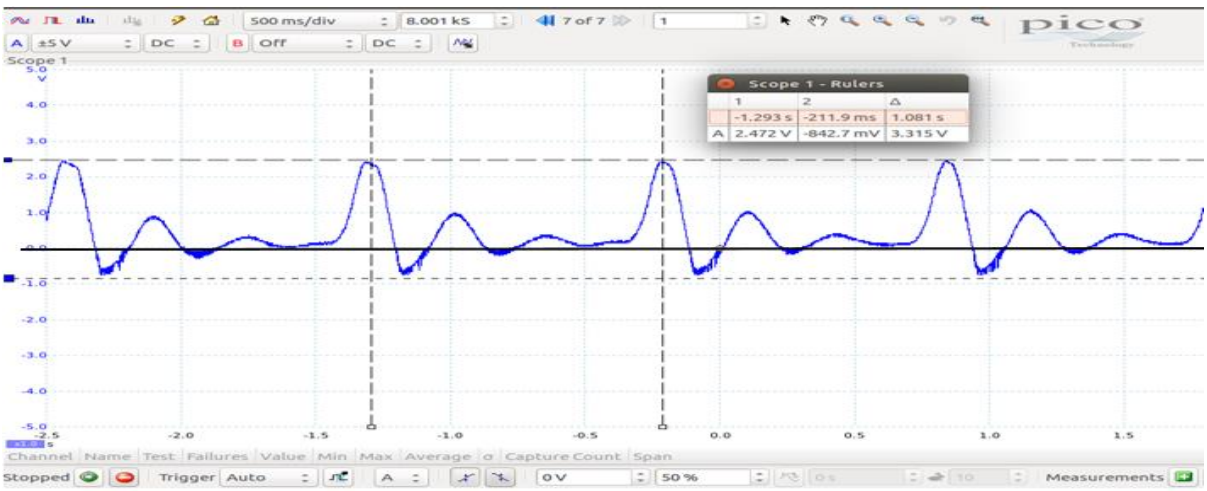
**Cycle 1 – Testing: Heart Pulse Sensor**

The heart pulse sensor was tested in isolation.

The capacitor acts as a DC blocker/filter, as well as a signal controller. The pulsatile signal drops to a **0Vref** (voltage reference), alternating between **+/- 2.5V** – Figure 4.1.8. The MCU's ADC restricts signal readings to positive values, hence, a potential divider circuit was designed using the formula: **Voltage Supply or VS (5V) * (R1/R1+R2)**.

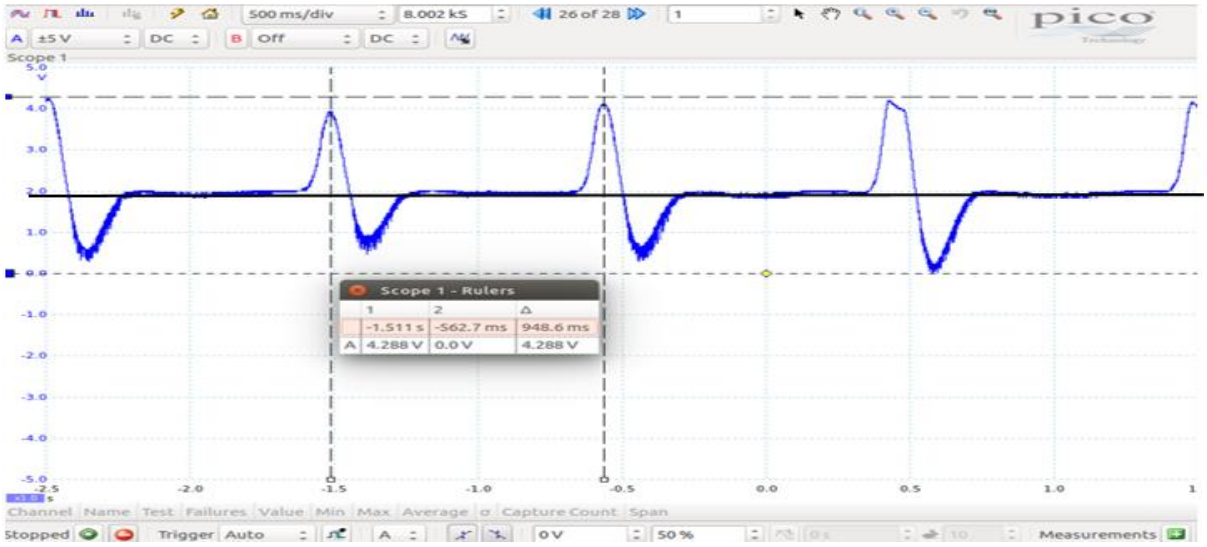
Equal values in both resistors initiates a DC signal free from interference of half the VS (**2.5V**); DC and AC together create a composite signal, which means that the signal rises to the DC voltage examined. Therefore, the signal would rise to **2.5V;** alternating between **+0-5V**. Consequently, with good manipulation of the formula the signal can be altered to suit the MCU restrictions – Figure 5.1.9.

**Figure 5.1.8 – Testing Signal Representation (Before)**



Exemplified, the signal generated from a Pico Scope is read as the following: **3.315V** Amplitude and **1.081s** is the time taken from peak-to-peak. The graph generated by Pico Technology exemplified an efficient signal diagnosis. The signal is not asymmetric and contains unreliable peaks, in this case, a DC blocker and potential divider is used to control and improve the signal, as well as its performance.

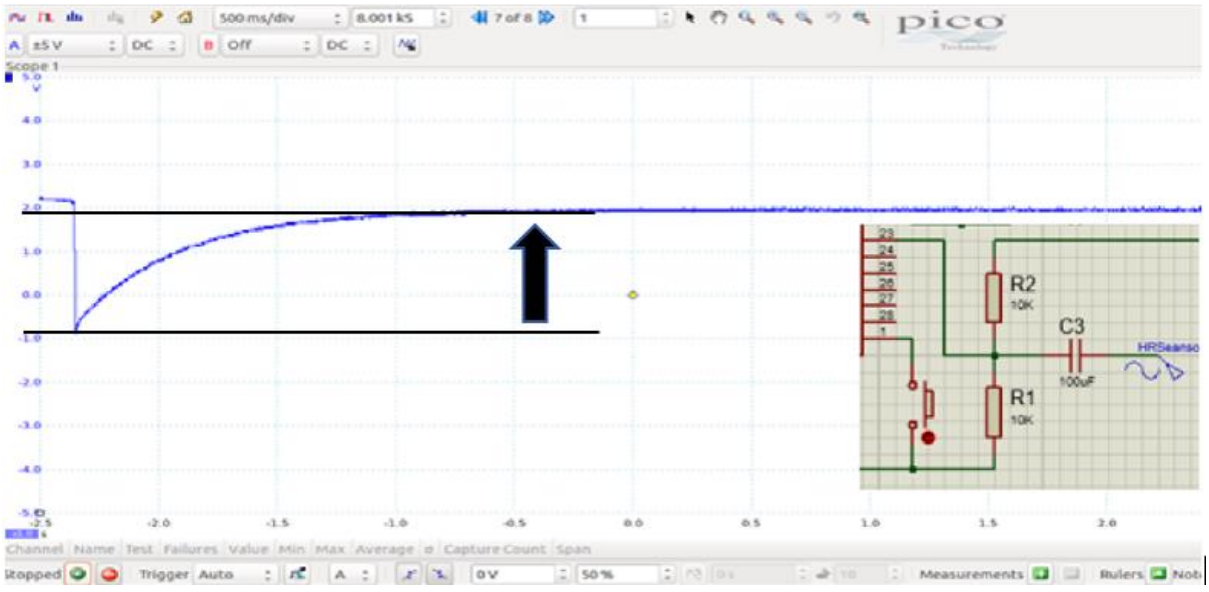
**Figure 5.1.9 – Testing Signal Representation (After)**




The 'after' graph was generated using the following values in the potential divider formula: VS(**5V**) * (R1 (**6.3K**) / ( R1(**6.3K**) + R2(**10K**) ) = **1.932515V**. The graph clearly demonstrates a controlled environment and improved signal and performance; resulting from the filter/DC blocker and signal controller. In addition, the signal rose to **1.9Vref** as a response from the potential divider design.

Figure 5.1.10 displays the transition from a pure analogue signal (non-DC) to a composite signal (AC+DC).

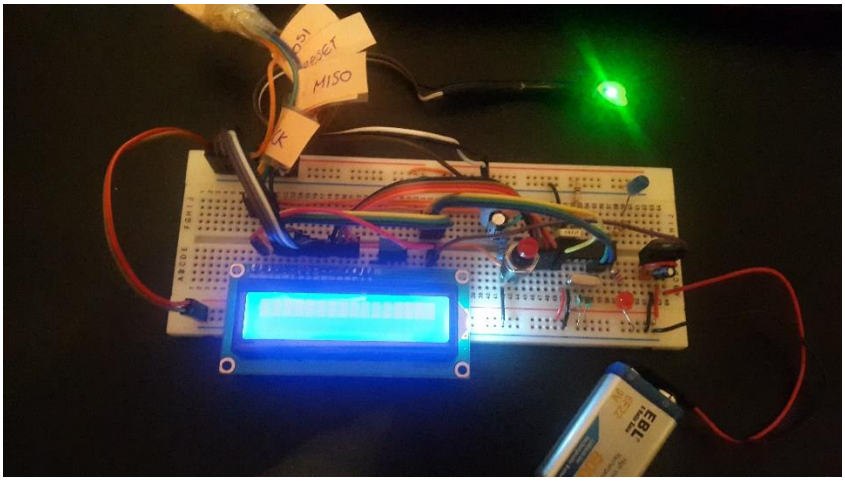
**Figure 5.1.10 – Pure to Composite Transition**



### Cycle 1 – Testing: Circuit Design

The circuit design was tested and powered as expected; the LCD and the sensor both lit up – Figure 5.1.11

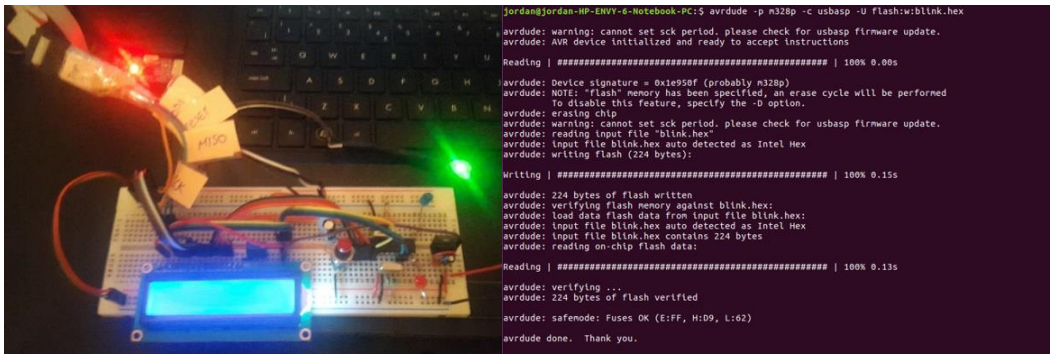
**Figure 5.1.11 – Circuit Design Test**




### Cycle 1 – Testing: Programmer (Programming MCU)

When connecting the programmer with the circuit's plug, the USBASP programmer lit-up and programming the MCU proved to be simple – Figure 5.1.12.

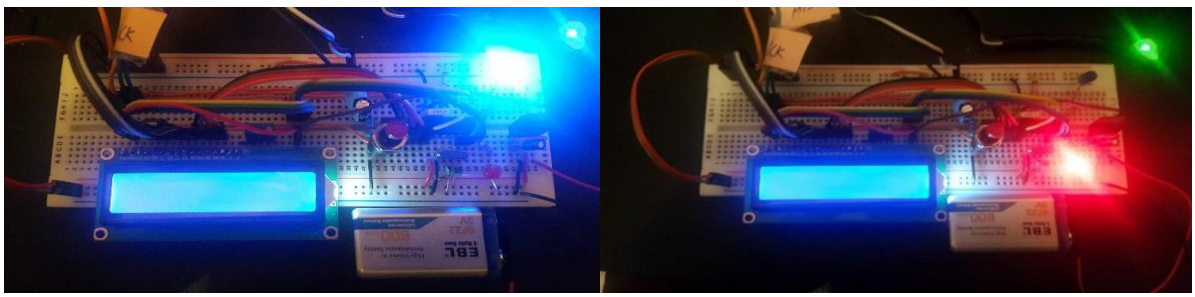
**Figure 5.1.12 – MCU programmed**




### Cycle 1 – Testing: Results (LED Blink)

As soon as programming was completed the LEDs begun to blink repeatedly – Figure 5.1.13.

**Figure 5.1.13 – LED Blink**



## Cycle 1 – Evaluation

Cycle one was initially pragmatic and the design ad hoc. Finally, the most crucial aspect of the project; having a prototype to work with, was complete. This cycle was considerably the most time-consuming phase, in conjunction with the upcoming cycle. Each phase was clearly demonstrated, thoroughly defined and efficiently tested.

Results satisfied the fundamental purpose of cycle one, the pulsatile signal was efficiently analysed and controlled, and the setup of components was implemented in the exact way it was designed. The LCD demonstrated good connectivity and MCU was effectively tested (via LEDs) along with the USBASP adapter/programmer. Thus, a HRM prototype was developed.

Cycle one and Cycle two can be described as crucial foundations in this project. Success of the project would be impossible without the completion of both cycles.

CHAPTER 6

# ADA INTEGRATION & TUTORIALS

## 6.1 – AVR-ADA Framework Design & Implementation

Cycle 2 – AVR-ADA cross compiler setup

- Consists of the AVR-ADA Cross-Compiler, Library Set-up Tutorial and Installation. Tested using a program implemented in AVR-ADA.
- **Product**: System prototype running an AVR-ADA program that blinks LEDs.

Microcontrollers are not specifically designed to carry Ada code in them, thus, running them with Ada code is not that simple. This document was designed to inspire those interested in the fore-mentioned research and upcoming development. Therefore, a tutorial on how the framework is designed, built and installed was included, as well as tested in two separate OSs.

**Cycle 2 – Design: AVR-ADA cross compiler setup**

The author of this thesis had absolutely no experience in this field. Research is somewhat limited and Linux was not the operating system used on a daily basis prior to this project. The key was accepting the level of complexity that this procedure entailed and preparing for all sorts of problems arising. As a result, it provided a level of emphasis encompassing wide bandwidth.

Research conducted, acceptance and meticulous attention prescribed was the key to successfully completing Cycle two.

A flowchart was designed encompassing all fundamental steps that this framework entails – Figure 6.1.1. The design of this flowchart could have not been completed in the way it was if it had not been for Anton G. Setzer, Project Supervisor, ReadMe Files and the following references:

(Ebert, 2003), (Ebert, 2012), (Ebert, 2012), (Ebert, 2013), (Ebert, 2015) (Cobbaut, 2015-05-24)

Special Thanks to Rolf Ebert for being responsible for most of the sources available regarding AVR-Ada and ongoing contribution.

Special Thanks to Anton G Setzer for his assistance in problem solving.

**Figure 6.1.1 – Flowchart**


ReadMe Files are present in all packages and were used during the design of the upcoming tutorial.

**Cycle 2 – Design: AVRDude Installation**

AVRdude is used to program microcontrollers and was installed using the following reference: (M, 2014)

AVRdude is user friendly and easily installed, nonetheless a small tutorial was included in the implementation.

**Cycle 2 – Design: AVR-ADA LED_ON Program**

**Pseudo Code:**

```
Initiate PORTB as Output
While True
    PortB0, True
    PortB1, True
End While
```

The aim is to simply run a program using the AVR-ADA library and test it by turning LEDs on.

**Cycle 2 – Implementation & Tutorial: AVR-ADA Framework**

It is an absolute necessity to understand what it is that the project entails. In this project, we are building a framework to integrate Ada into an AVR 8-Bit Microcontroller. Thus, it entails building a toolchain with the tools required to cross-compile AVR-ADA efficiently.

The OS used in this tutorial is **Ubuntu 16.04LTS.**

**Phase 1:**

Research, learn and understand the requirements, such as tools and procedures specified in the aforementioned references.

To build the framework, visit the following sites (in-the-order) and have a read-through.

- (Ebert, 2015)
- (Ebert, 2013)
- (Cobbaut, 2015-05-24) – Linux Fundamentals

You will find that the required tools are the following:

- binutils (assembler, linker, etc.), 2.24
- gcc (compiler), 4.9.2
- avr-libc (C run time system, start-up code, linker scripts), 1.8.1
- avr-ada (Ada run time system, gcc and binutils patches), 1.2.2

The references above state that a GCC is required prior to building the cross-compiler. Reason being, some of the scripts are written in Ada.

It is fundamental to have realised that a build script is the first step after having laid the tools in a relevant file path and checking a GCC compiler is installed. In addition, when personally working on building the script (which creates/installs the toolchain, bare compiler and binutils), a common error stated that additional tools were required:

**PS.** Do not install GCC from this reference if not already installed.

Perform the instructions displayed to avoid errors from an early stage.

GMP

GMP is the GNU Multiple Precision Arithmetic Library.

```
wget ftp://gcc.gnu.org/pub/gcc/infrastructure/gmp-4.3.2.tar.bz2
bunzip2 gmp-4.3.2.tar.bz2
tar xvf gmp-4.3.2.tar
cd gmp-4.3.2
./configure --disable-shared --enable-static --prefix=/tmp/gcc
make && make check && make install
```

MPFR

MPFR is the GNU Multiple-precision floating-point rounding library. It depends on GMP.

```
wget ftp://gcc.gnu.org/pub/gcc/infrastructure/mpfr-2.4.2.tar.bz2
bunzip2 mpfr-2.4.2.tar.bz2
tar xvf mpfr-2.4.2.tar
cd mpfr-2.4.2
./configure --disable-shared --enable-static --prefix=/tmp/gcc --with-gmp=/tmp/gcc
make && make check && make install
```

MPC

MPC is the GNU Multiple-precision C library. It depends on GMP and MPFR.

```
wget ftp://gcc.gnu.org/pub/gcc/infrastructure/mpc-0.8.1.tar.gz
tar zxvf mpc-0.8.1.tar.gz
cd mpc-0.8.1
./configure --disable-shared --enable-static --prefix=/tmp/gcc --with-gmp=/tmp/gcc --with-mpfr=/tmp/gcc
make && make check && make install
```

Reference: (joelparkerhenderson, 2017)

Problem 2: AVR-ADA is based on gcc-4.7.2, we need gcc-4.9.x at least. After performing some research, a repository was found in github and will be included later in this tutorial.

Nonetheless, download **avr-ada-1.2.2** as it will satisfy the version of Ada when building the script.

Files can be found and downloaded for free from:

- http://linux.freesoftware0.com/download/gnu-binutils-224-4f6b83b2.html

or

- http://linux.freesoftware0.com/gnu-binutils-224-4f6b83b2.html
- https://ftp.gnu.org/gnu/gcc/gcc-4.9.2/
- http://download.savannah.gnu.org/releases/avr-libc/
- https://sourceforge.net/projects/avr-ada/?source=typ_redirect

If any of the above sites are outdated, multiple download sites can easily locate the tools.

Place the downloaded tools or packages in a new directory named **src**.

Ubuntu usually comes with a GCC compiler. However, check if one is already installed

**$ gcc -v**

```
root@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.9/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.9.4-2ubuntu1~16.04' --with-bugurl=file:///usr/share/doc/gcc-4.9/README.Bugs --
enable-languages=c,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.9 --enable-shared --enable-linker-build-id --libexecdir
=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.9 --libdir=/usr/lib --enable-nls --with-s
ysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-vtable-verify --enable-
plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.9-am
d64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.9-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-4.9
-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-a
rch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-lin
ux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.9.4 (Ubuntu 4.9.4-2ubuntu1~16.04)
```

If gcc-4.9.x branch is not installed, though, a gcc-5.x branch is, then it will have to be uninstalled. In addition, install the correct GCC version **4.9.x**.

If GCC has to be uninstalled, typing the following command in your terminal:

**$ sudo apt remove gcc-***

Type the following command to install GCC-4.9.x branch.

**$ sudo apt-get install gcc-4.9**

Once that is sorted, create a new and independent directory where your files will be stored.

The file path used in this tutorial is: **/installation/devel/base_avr_cross**

- Copy the file **src** into **base_avr_cross**

**$ cp -r file/path/to/downloads/src /installation/devel/base_avr_cross**

This will give you the file path**: /installation/devel/base_avr_cross/src**; where your downloaded tar files originate.

Now un-tar them:

**$ cd /installation/devel/base_avr_cross/src**
**$ tar xfvj avr-ada-1.2.2.tar.bz2 , tar xfvj avr-libc-1.8.1.tar.bz2 etc.**
**$ ls -l src**

```
jordan@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross$ ls -l sr
c
total 144284
drwxr-xr-x  7 root root      4096 Sep 22 23:00 avr-ada-1.2.2
-rw-r--r--  1 root root    696822 Sep 26 00:15 avr-ada-1.2.2.tar.bz2
drwxr-xr-x 11 root root      4096 Sep 22 22:59 avr-libc-1.8.1
-rw-r--r--  1 root root   4790340 Sep 26 00:16 avr-libc-1.8.1.tar.bz2
drwxr-xr-x 17 root root      4096 Sep 22 22:58 binutils-2.24
-rw-r--r--  1 root root  28317443 Sep 26 00:15 binutils-2.24.tar.bz2
drwxr-xr-x 43 root root      4096 Oct  4 16:47 gcc-4.9.2
-rw-r--r--  1 root root 113913284 Sep 26 00:16 gcc-4.9.2.tar.bz2
jordan@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross$
```

Another problem that should be avoided is file permissions, only root can access these files. Thus, we change them so that User can also access them:

**$ chown $USER -R /src**

Now, when executing the command ls -l src, {username} and {root} should be displayed. File path that we would be working from is now set and ready.

Perform last check on **GCC**, **GNATGCC** and **GNATLS** versions:

**$ gcc -v** --checks your gcc version, in this case it's version 4.9.4

```
root@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.9/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.9.4-2ubuntu1~16.04' --with-bugurl=file:///usr/share/doc/gcc-4.9/README.Bugs --
enable-languages=c,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.9 --enable-shared --enable-linker-build-id --libexecdir
=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.9 --libdir=/usr/lib --enable-nls --with-s
ysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-vtable-verify --enable-
plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.9-am
d64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.9-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-4.9
-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-a
rch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-lin
ux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.9.4 (Ubuntu 4.9.4-2ubuntu1~16.04)
```

**$ gnatgcc -v** --checks your gnatgcc version, and if its present

```
jordan@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross$ gnatgcc
-v
Using built-in specs.
COLLECT_GCC=gnatgcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.9/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.9.4-2ubuntu1~16
.04' --with-bugurl=file:///usr/share/doc/gcc-4.9/README.Bugs --enable-languages=
c,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.9 --enabl
e-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gette
xt --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.9 --libdir=/
usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-de
bug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-vtable-veri
fy --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt
=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.9-amd64/j
re --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.9-amd64
--with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-4.9-amd64 --with-arch-dir
ectory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --e
nable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multi
lib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=re
lease --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.9.4 (Ubuntu 4.9.4-2ubuntu1~16.04)
```

**$ gnatls -v** --checks GNATLS version and if its present

```
jordan@jordan-HP-ENVY-6-Notebook-PC:/installation/devel/base_avr_cross$ gnatls -
v

GNATLS 4.9.3
Copyright (C) 1997-2014, Free Software Foundation, Inc.

Source Search Path:
   <Current_Directory>
   /usr/lib/gcc/x86_64-linux-gnu/4.9/adainclude/


Object Search Path:
   <Current_Directory>
   /usr/lib/gcc/x86_64-linux-gnu/4.9/adalib/


Project Search Path:
   <Current_Directory>
   /usr/share/ada/adainclude
```

**Phase 2**:

- Building the cross-compiler tool-chain for binutils, avr-libc and avr-gcc with a recognizable file path.

**PS**. **sudo** is used to execute files as root, this allows you to perform certain commands which you may not be permitted usually as an original user. It may ask for your password if it is set-up, and will recognize you for the next 15 minutes before you have to enter it again. Using **sudo** is not always essential as it restricts access. Nonetheless, access can be granted by changing file permissions as was performed earlier.

**Step1:** The build-script originating at **avr-ada-1.2.2/tools/build** directory requires editing.

Copy and paste the build-script somewhere so the original remains untouched, in this tutorial the script was copied to the Desktop, edited and copied back to the base_avr_cross directory.

```
$ cp avr-ada-1.2.2/tools/build/build-avr-ada.sh ~/Desktop
```

Open it on your usual text editor and get ready to edit. **Do not** make mistakes.

If you know how to edit using command line editor – **vi,** you can work directly from the base_avr_cross directory.

```
$ sudo vi build-avr-ada.sh
```

- Delete what needs editing first using **delete** button.
- Press **i** to enable editing and **esc** to disable editing.
- Once finished, exit the editor and save changes – **:wq!**
- Or Quit without saving using – **:q!**
- Undo something by pressing **'u'**

Moving on, at **line 72** change prefix to:

**PREFIX="/opt/avr_492_gnat"**

Or any other file path which does not yet exist.

```
BASE_DIR=$PWD
OS=`uname -s`
case "$OS" in
    "Linux" )
        PREFIX="/opt/avr_492_gnat"
        WITHGMP="/usr"
        WITHMPFR="/usr";;
    "Darwin" )
        PREFIX="/opt/avr_492_gnat"
        WITHGMP="/opt/local"
        WITHMPFR="/opt/local";;
    * )
        PREFIX="/mingw/avr_492_gnat"
        WITHGMP="/mingw"
        WITHMPFR="/mingw";;
```

This will be known as the installation directory, the file path that the environment variable **$PATH** should include when compiling software.

At **line 95** change the version of tools to match your downloaded version:

```
VER_BINUTILS=2.24
VER_GCC=4.9.2
VER_MPFR=3.1.0
VER_MPC=0.8.2
VER_GMP=4.3.2
VER_LIBC=1.8.1
VER_AVRADA=1.2.2
```

At **line 124** set the script to download any necessary tarbals and patches:

```
# Download necessary tarbals and patches using wget and cvs
download_files="no"
delete_obj_dirs="no"
delete_build_dir="yes"
delete_install_dir="no"
build_binutils="yes"     #build cross avr-bin
build_gcc="yes"          #build cross avr-gcc
build_mpfr="no"
build_mpc="no"
build_gmp="no"
build_libc="yes"         #build cross avr-libc
build_avrada="no"        #we build avr-ada ourselves
```

At **line 115** of the build-script, set the AVRADA_PATCHES to your file path; **installation/devel/base_avr_cross/avr-ada/patches** – both binutils and gcc-492 should be available. This file path is where the new updated avr-ada package will be installed (later).

```
#AVRADA_PATCHES=$AVR_BUILD/avr-ada-$VER_AVRADA/patches
AVRADA_PATCHES=/installation/devel/base_avr_cross/avr-ada/patches
AVRADA_GCC_DIR="$AVRADA_PATCHES/gcc/$VER_GCC"
AVRADA_BIN_DIR="$AVRADA_PATCHES/binutils/$VER_BINUTILS"
AVRADA_LIBC_DIR="$AVRADA_PATCHES/avr-libc/$VER_LIBC"
```

Now we need to save and copy the build script back to our file:

**/installation/devel/base_avr_cross/** – This is the main folder we work from – Easy access

$ **cp ~/Desktop/build-avr-ada.sh /installation/devel/base_avr_cross/**

Before we proceed to build the script, a common error that arises is usually locating the binutils and gcc patches – it is significant that the patches obtained are the latest ones. Therefore, let's make a clone to obtain the latest **avr-ada** package with the latest patches.

Access the directory which you will use to place the git repository previously mentioned.

$ **cd installation/devel/base_avr_cross/**

Make sure **git** is installed first, otherwise, do so by typing the following command:

$ **sudo apt-get install git**

Clone the latest version of avr-ada using the following git repository.

$ **git clone git://git.code.sf.net/p/avr-ada/code avr-ada**

**PS. When testing this tutorial a second time on a different OS, the git repository was updated to; git://git.code.sf.net/p/avr-ada/code avr-ada-code**

Add permissions to allow execution of the script by everyone or user. If you would like the permissions assigned to user – add **u+x** instead of just **+x** (for all) on the following command:

$ **sudo chmod +x build-avr-ada.sh**

Next, create an installation directory (**/opt/avr_492_gnat**) and assign permissions to it.

```
$sudo mkdir /opt/avr_492_gnat
$ sudo chmod 777 /opt/avr_492_gnat
```

**PS**. chmod 777 is used to overwrite/write permissions to all content within the specified directory. Provides read – write – execute permissions. 775 or 773 can also be used. However, before assigning them, become familiar with them.

Finally, run the script

```
$ sudo ./build-avr-ada.sh
```

When is it successful?

It is successful when the build script ends correctly.

```
------------------------------------------------------------
2016-09-26_17:22:36  Build end
------------------------------------------------------------


Build logs are located in /installation/devel/base_avr_cross/build
Programs are in the /opt/avr_492_gnat hierarchy
You may remove /installation/devel/base_avr_cross/build directory
```

If any errors arise within the log files (see PS below) that are to do with permissions, perform **chown** command as previously executed to the directory it can't access.

At this stage, we have the binutils and the bare compiler setup.

**PS**. Always read and try to understand what the scripts are programmed to do, as well as, what the readme.txt documents specify. Meticulous attention to detail is required for the solution of errors. Furthermore, in the case of errors during this build script, a build directory is created with log files that display where and why the script has failed, try to identify the problem through them. Hopefully no errors should arise.

**Phase 3**:

In our last phase, we require the AVR run-time system (rts) to be set up.

Thus, access new avr-ada directory:

```
$ cd /installation/devel/base_avr_cross/avr-ada (-code)
```

Before proceeding we need to let the **PATH** environment variable know which path to search. Hence, let's add the PATH.

```
$ export PATH=/opt/avr_492_gnat/bin:$PATH      # adds path to our environment variable
```

In the AVR-ADA directory there is a configure file and a ReadMe.txt.

```
$ ./configure                    #run configure
```

As a result, the configure file generates a config file with the version of gcc and few other properties. If any errors arise, open the configure file and ensure that the **/dev/null** has no '**/**' at the end of null. If permissions are denied, remember **chown $USER -R /file**.

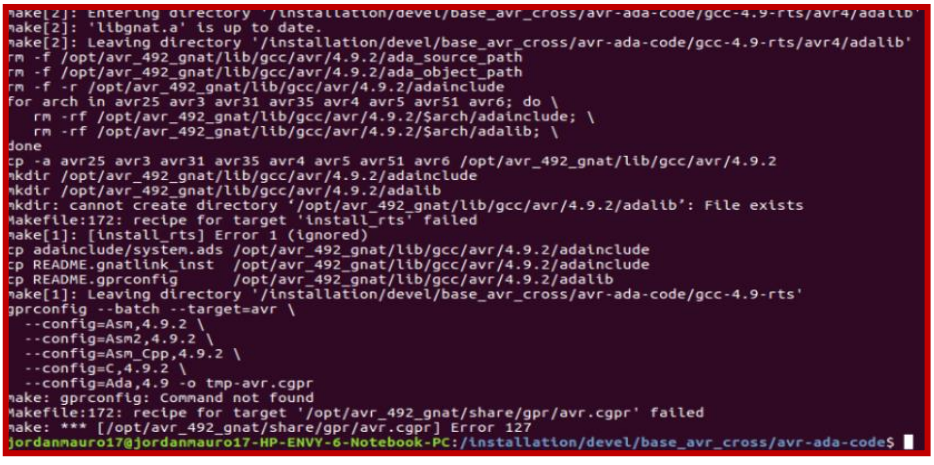
When successfully completed, navigate to **gcc-4.9.x-rts** (runtime system) file

```
$ cd gcc-4.9-rts
```

As stated in the ReadMe.txt file, we make, build and install our runtime system using the following script:

```
$ make build_rts

$ make install_rts
```

At this stage, some errors may arise at completion, but if make install_rts ends as the following script, proceed with the instructions.



The gcc runtime system has been installed.

In addition, following the avr directory and ReadMe File insinuates that the avr library (MCUs, etc) needs to be setup. Thus, navigate to the avr directory to build and install the avr library.

```
$ cd avr/
$ make
```

The script at this point ends with an error: **Makefile: 90: recipe for target 'build_libs' failed**

Indeed, this was an error that was both stressing and time consuming, among a few others not mentioned. The following command however fixes the problem.

```
$ make -k make -C avr_lib build_libs
```

The avr_library was built successfully. Thus, now we install it.

```
$ make -k install
```

When completed, you may realise a few extra files in the installation path set previously.

```
$ ls /opt/avr_492_gnat
```

To check the success of the avr cross-compiler installation, type the following command

```
$ avr-gnatls -v
```

**Cycle 2 – Implementation & Tutorial: AVRdude and AVR-ADA program setup**

AVRdude is used for programming software into MCUs via the use of a USBASP or ISP programmer.

AVRdude can be easily installed by typing the following command:

```
$ sudo apt-get install avrdude
```

**Cycle 2 – Implementation: AVR-ADA LED_ON Program**

avr_workSpace\test\led_on.adb

```
with AVR;       use AVR;    -- 1: make available general type definitions and names
with AVR.MCU;               -- 2: make use of the controller specific ports and pins

procedure LED_On is
   LED : Boolean renames MCU.PortB_Bits (1); -- 3: rename controller pin 0 and 1 of port B to a name of
   LED1 : Boolean renames MCU.PortB_Bits (0);-- 4: the problem domain
begin
   MCU.DDRB_Bits := (others => DD_Output);    -- 5: set data direction of all pins of port B to output
   LED := High;
   LED1 := High;                              -- 6: draw output pin to High, which turns on the LED
end LED_On;
```

**Cycle 2 – Testing: AVR-ADA setup**

Furthermore, testing the compiler can be done by compiling the apps using make. Navigate to apps directory.

```
$ cd /installation/devel/base_avr_cross/avr-ada/apps

$ make clean          #gets rid of unused/unnecessary files

$ make
```

All targets failed, however, proof of compiler working is witnessed.

Navigate to uart_echo and run make command; Another test proving compiler working.

```
$ cd uart_echo/

$ make
```

Run LS command, certain files have been created by the compiler; the '.HEX' file is used to program the MCU.

```
$ ls
```

Personally, displayed are unused files such as, in_out.*. Thus, they were removed also.

```
$ rm in_out.*
```

By taking a peak at the Makefile, it targets main, thus, it is all that is required along with **obj** and **build.gpr**.

Nonetheless, after taking a look at the Makefile, parts of it need editing. However, now that the avr-ada cross compiler and libraries are successfully built, installed and tested, set up a work space to work with and test the circuit.

```
$ mkdir ~/avrWorkspace
```

Now we need to copy the uart_echo directory in to avrWorkspace to use and edit the Makefile and build.gpr without affecting the original.

```
$ cp -r /installation/devel/base_avr_cross/avr-ada/apps/uart_echo/ ~/avr_workspace/
```

Open the **Makefile** and modify the following:

```
MCU:= atmega328p                    #select one that AVR library has available

GPR:= build.gpr

ADA_TARGET := led_on                #If project file is: led_on.adb; simply put led_on

AVRDUDE_PORT := usb

AVRDUDE_PROGRAMMER := uspasp        #or isp (the programmer  used to program MCU)
```

Once you have completed the modification to suit your specifications, navigate to the directory where the make file is located

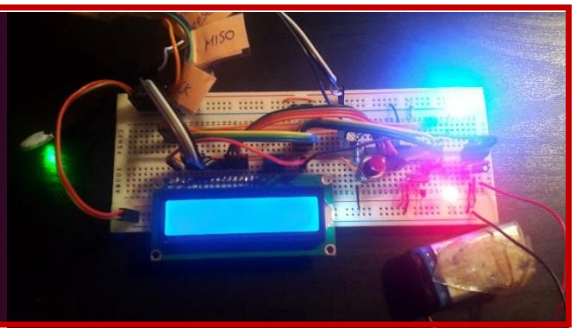
```
$ cd ~/avr_workspace/uart_echo

$ make all
```

```
jordan@jordan-HP-ENVY-6-Notebook-PC:~/avr_workSpace$ make all

-------- begin --------
avr-gcc (GCC) 4.9.2
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


Compiling C: blink.c
avr-gcc -c -mmcu=atmega328p -I. -gdwarf-2 -DF_CPU=8000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict
-prototypes -Wa,-adhlns=./blink.lst  -std=gnu99 -MMD -MP -MF .dep/blink.o.d blink.c -o blink.o

Linking: blink.elf
avr-gcc -mmcu=atmega328p -I. -gdwarf-2 -DF_CPU=8000000UL -Os -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-pr
ototypes -Wa,-adhlns=blink.o  -std=gnu99 -MMD -MP -MF .dep/blink.elf.d blink.o --output blink.elf -Wl,-Map=blink.map,--cref     -lm

Creating load file for Flash: blink.hex
avr-objcopy -O ihex -R .eeprom -R .fuse -R .lock blink.elf blink.hex

Creating load file for EEPROM: blink.eep
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 --no-change-warnings -O ihex blink.elf blink.eep || exit 0

Creating Extended Listing: blink.lss
avr-objdump -h -S -z blink.elf > blink.lss

Creating Symbol Table: blink.sym
avr-nm -n blink.elf > blink.sym

Size after:
AVR Memory Usage
----------------
Device: atmega328p

Program:     224 bytes (0.7% Full)
(.text + .data + .bootloader)

Data:          0 bytes (0.0% Full)
(.data + .bss + .noinit)
```

A set of new files have automatically been created, AVRdude requires the hex file to program the MCU. In addition, the last section marked proves the compiler is working, should be similar in previous tests.

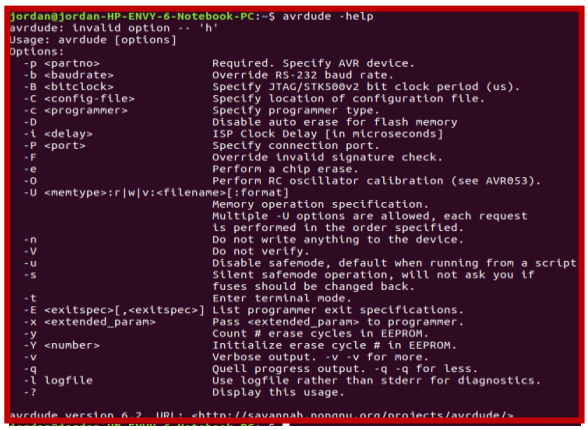
```
$ avrdude -p m328p -c usbasp -U flash:w:led_on.hex
```

```
jordannauro17@jordannauro17-HP-ENVY-6-Notebook-PC:~/avrWorkspace/test$ avrdude -c usbasp -p m328p -U flash:w:led_on.hex

avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: reading input file "led_on.hex"
avrdude: input file led_on.hex auto detected as Intel Hex
avrdude: writing flash (160 bytes):

Writing | ################################################## | 100% 0.15s

avrdude: 160 bytes of flash written
avrdude: verifying flash memory against led_on.hex:
avrdude: load data flash data from input file led_on.hex:
avrdude: input file led_on.hex auto detected as Intel Hex
avrdude: input file led_on.hex contains 160 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.13s

avrdude: verifying ...
avrdude: 160 bytes of flash verified

avrdude: safemode: Fuses OK (E:FF, H:D9, L:FF)

avrdude done.  Thank you.
```

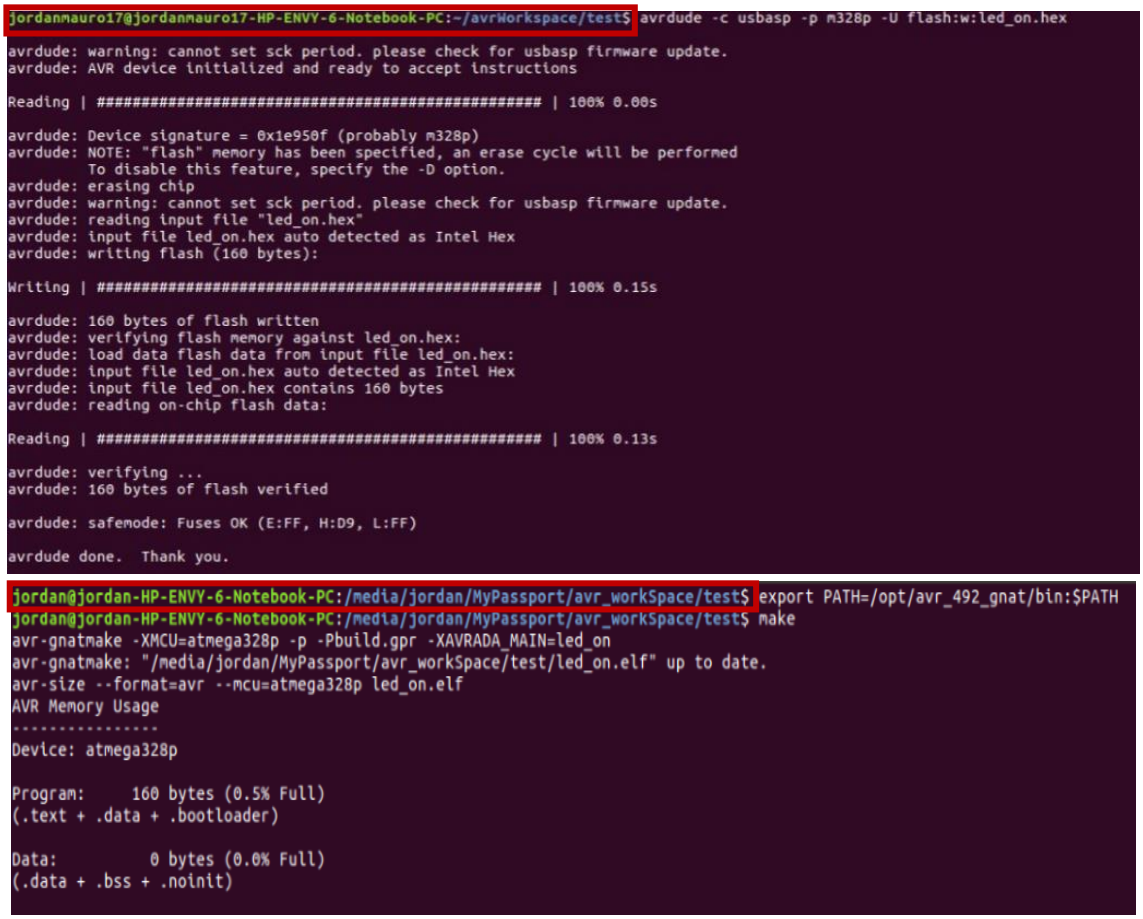


AVRDude -help provides information regarding **-p** or **-c** and more. Ex. **c = programmer**, **p = part number** – i.e. Atmega328p part number is **m328p** and programmer is **usbasp**.

**$ avrdude -help**




**Proof of tutorial being tested in two different OS:**







Further knowledge on AVR-ADA programming can be obtained by reviewing examples available in the AVR library.

**Cycle 2 – Evaluation**

The tutorial has been tested in two separate OSs, both run-throughs had their differences regarding problem-solving. A solution for each was found and adapted to the tutorial; making it extremely easy for people with no experience in either Linux or Cross-compilers. This tutorial may encourage people whom may have approached this setup and given up along the way. It has been thoroughly analyzed and designed to cover all possible errors that may arise, providing a smooth and systematic approach to begin programming using Ada on microcontrollers. This cycle has been the most tedious challenge that the author/developer has encountered, not to mention time consuming. Nonetheless, it was an amazing accomplishment and well-inclined learning experience.

CHAPTER 7

# SYSTEM INTELLIGENCE

The software design symbolises that the main foundations have been successfully accomplished and tested using an extremely simple program on the prototype. The following cycles are purely based on enhancing system intelligence to a sophisticated level.

## 7.1 – Software Characteristics

Design of software signifies that a formal analysis is required. In Object-Oriented Design, the usual analysis followed equates to performing a list of steps. However, since the Ada infrastructure is constructed using software packages instead of objects, certain approaches will vary. In addition, the design approach usually considered also changes when designing a device that uses an MCU to perform tasks in real-time.

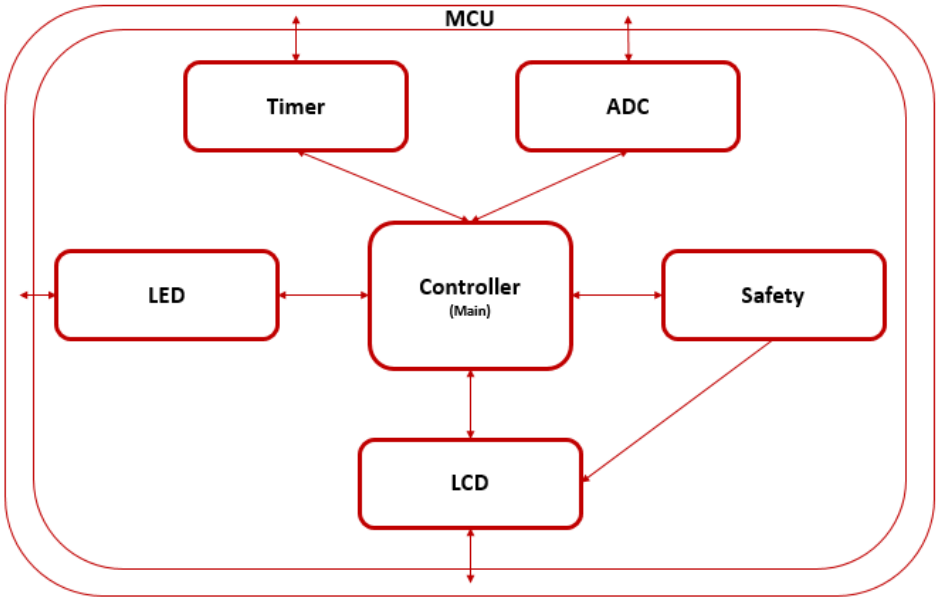
**Common Design Approach**

- Identifying Objects also known as nouns in the requirements specification.
- Identifying Actions also known as verbs in the requirements specification.
    - Developing CRC cards for each object.
- Identifying collaborations between objects.
- Constructing hierarchies from those objects and identifying abstract classes.
- Identifying subsystems.
    - Redesigning collaborations to conform with subsystems.
- Developing protocols in conformance with the above prior to implementation.

Consequently, since this project varies on infrastructure, the common design approach was altered. The Main Program will be treated as a subsystem; everything will interact with it and services will originate from it. The subsystem will be called “controller”, and will be where the pulse-detection algorithm will reside. The rest of the packages will be used by the *controller*, which constitutes to the implementation file withholding most of the system’s intelligence.

Thus, identifying packages concludes to Figure 7.1.1.

**Figure 7.1.1 – Identifying Packages**



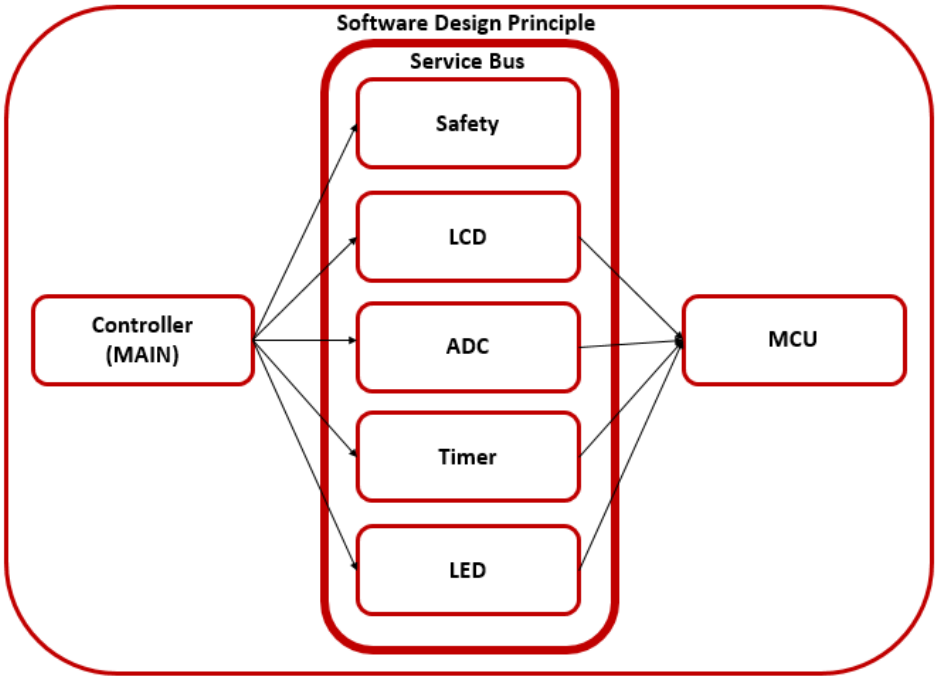

With reference to the software requirements, five main packages were identified. In addition, by performing practical and logical observations, the above diagram was formed; based on class (package) intelligence.

The controller knows about all packages; however, does not know how to communicate with the MCU directly. Therefore, it uses the packages to do so. Safety uses LCD to print useful information at the end of a BPM result. LCD, LED and Safety do not really provide the controller with useful information; however, all packages are bi-directional as somehow the system lets the controller know the request was fulfilled. In contrast, both Timer and ADC packages do provide the controller with useful data when requested; such as, when timing events occur (Timer-Controller) and pulsatile data (ADC-Controller). Thus, the controller (main) program is the collaborator, asking for distinct services from all the other '*Objects*'.

Integrating SPARK Tools requires a disparate mechanism or infrastructure from the current software design. However, only when the current design is fully implemented and tested, will SPARK Tools be integrated or attempted.

Furthermore, the software design principle is not aimed to produce the usual class hierarchy. Instead, it formulates to a service bus – Figure 7.1.2.

**Figure 7.1.2 – Software Design Principle**




Hierarchies will not form part of the software design, each package will be designed in isolation and tested with reference to the MCU to distinguish effective functionality. Subsequently, package design will be employed to the controller's algorithm when applicable.

The following cycle is based on implementing a simple delay to perform a crucial timing test.

# 7.2 – Iterative Design Principle Continued

Cycle 3 – One Second Delay

- Refined software design with improvements – 1 Second Delay
- **Product:** System flashing LEDS every 1 second.

**Cycle 3 – Design: One Second Delay**

Delays are usually straight forward, however, testing the MCU's clock is crucial. The MCU may need some alterations, as by default the ATMega328P and possibly others operate at 1MHz frequency. Having an external clock source (CS) or crystal oscillator (CO) signifies that the MCU may need to be programmed to oscillate at the same speed as the external CS. Perhaps, the AVRDude programmer could send the signal directly.

The best way to approach this test is by using the previously implemented program and use AVR-Ada's delay function. Test if there is a 1 second delay exactly between each LED activation/deactivation phase. If there isn't, the MCU will require an additional setup procedure based on MCU Fuse Bits.

**Pseudo Code: One Second Delay**

```
Rename LED1 and LED2 ports and declare it as Boolean
begin
instantiate PortB as Output
infinate loop
   LED1 ON
   LED2 OFF
   delay 1 second

   LED1 OFF
   LED2 ON
   delay 1 second
end loop
```

In the case of the MCU having the default setup, a blog states that the fuse bits can be set by the programmer itself, simply by adding the high fuse byte and low byte to the command used to program the MCU – Figure – 7.2.1.

**Figure 7.2.1 – Fuse Bits**

```
hfuse: 0b11011001 (0xD9), so no change here
lfuse: 0b11111111 (0xFF)
```

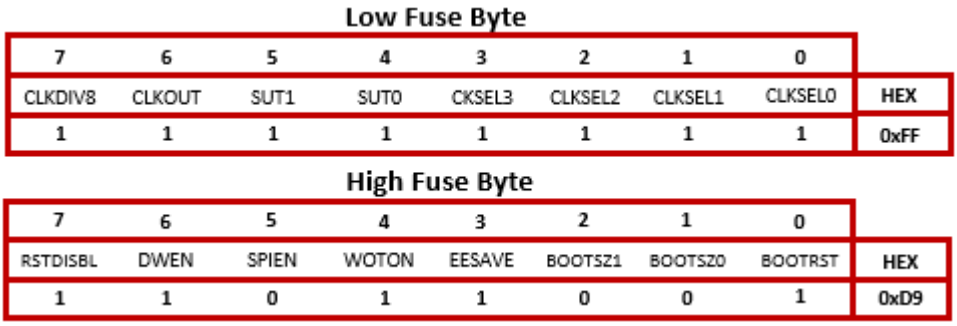
And the avrdude command becomes:

```
avrdude -c myavr -p m8 -P /dev/parport0 -U lfuse:w:0xFF:m
```

(Louic, 2008)

Fuse Bits explained by Louic are based on the ATMega8, however, they are both very similar. Nonetheless, the ATMega328P datasheet page 49-53 explains the external clock requirements and have been formulated in Figure – 7.2.2.

**Figure 7.2.2 – ATMega328P Fuse Bytes**



**Low Fuse Byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| CLKDIV8 | CLKOUT | SUT1 | SUT0 | CKSEL3 | CLKSEL2 | CLKSEL1 | CLKSEL0 | HEX |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0xFF |

**High Fuse Byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| RSTDISBL | DWEN | SPIEN | WOTON | EESAVE | BOOTSZ1 | BOOTSZ0 | BOOTRST | HEX |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0xD9 |

(Corporation, 2017)

Thus, with reference to the ATMega8 and ATMega328P, Fuse Bytes do not change regarding a 16MHz setup.

### Cycle 3 – Implementation: Programming

The above pseudo code was transformed and implemented to the following Ada program – Figure 7.2.3

**Figure 7.2.3 – Led_On: One Second Delay**

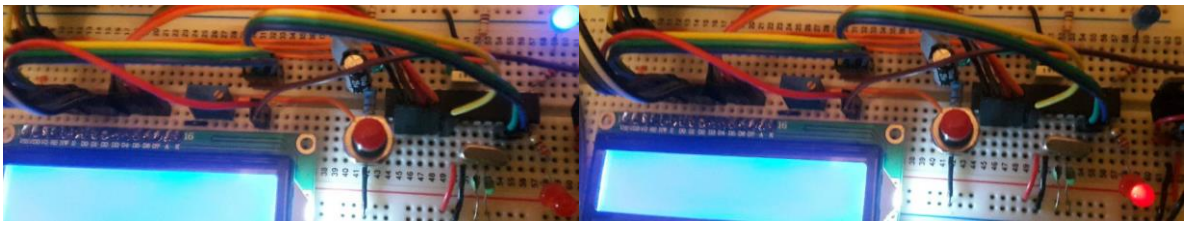
```
with AVR.MCU;
use AVR;
with AVR.Real_Time.Delays;              --function delay comes from this library

procedure Led_on is
    LED : Boolean renames MCU.PortB_Bits(1);    --Rename PortB Bit1 set as Bool
    LED1 : Boolean renames MCU.PortB_Bits(0);   --Rename PortB Bit0 set as Bool
begin

    MCU.DDRB_Bits (0) := DD_Output;     --Set as output
    MCU.DDRB_Bits (1) := DD_Output;

    loop
        LED := True;                    --LED On
        LED1 := False;                  --LED Off
        delay 1.0;                      --1 Sec Delay

        LED := False;                   --LED Off
        LED1 := True;                   --LED On
        delay 1.0;                      --1 Sec Delay
    end loop;
```

The program was successfully compiled and is ready for testing.

### Cycle 3 – Testing: One Second Delay

When testing the program without the fuse bits, LEDs were rapidly flashing. However, including the fuse bits to the AVRDude programming command, the LEDs flashed effectively every second – Figure 7.2.4.

**Figure 7.2.4 – One Second Delay**



Fortunately, one second delay was accomplished.

Cycle 3 – Evaluation

Previous assumptions were correct, the MCU by default was set to operate at 1MHz. The MCU needs to be told what clock source (CS) is being used via fuse bytes, those bytes require setup depending on the frequency of the crystal. The LED was flashing extremely fast during the first test, demonstrating timing deficiencies. It was clear that both the internal and external CS was unsynchronized. The internal CS was oscillating at 1MHz per clock cycle, and the external CS was oscillating at 16MHz per clock cycle. Setting-up and applying the fuse bytes meant that both the internal and external CS were operating synchronously.

Now that the MCU has been carefully tested and adjusted, the system is ready to begin software design.

The following cycle is purely based on setting up the MCU to intercept and read the pulsatile signal.

Cycle 4 – ADC Setup

- Refined software design with improvements – ADC Setup
- **Product:** System reading ADC pulsatile signal and tested using LEDs

**Cycle 4 – Design: ADC Setup**

Actions that this ADC package should include are listed on a PRC Card (Package, Responsibility and Collaborations – Figure 7.2.5.

**Figure 7.2.5 – PRC Card**

| **PRC Card** |
| --- |
| **Package:** ADC |
| **Responsibilities and Collaborations:**<br>➢ Initialize ADC – Collaborates directly with MCU<br>➢ Start Conversion – Collaborates directly with MCU<br>➢ Identify when conversion has completed – Collaborates with Controller (main)<br>➢ Get the result from that conversion – Collaborates with ADC<br>➢ Convert the data into voltage values – Collaborates with Controller |

Protocols for all actions are necessary to provide a direct approach towards implementation and is an effective approach used in common software designs. However, Pseudo code was taught by Mr Gary Tam, and provided an effective vision towards implementation of any algorithm. Hence, the reason it has been thoroughly used in previous cycles and why it shall continue to be used.

**Pseudo Code Requirements – Initialize ADC**

According to the ATMega328P Datasheet, initializing the ADC requires the following bits to be manipulated from the ADMUX Register – Figure 7.2.6 and ADCSRA Register – Figure 7.2.7. However, prior to initializing them the MCU ADC ports are set to inputs.

**Figure 7.2.6 – ADMUX Register Byte**

**ADMUX Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit No. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REF1 | REF0 | ADLAR | | MUX3 | MUX2 | MUX1 | MUX0 | HEX |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0x40 |

Bit 6 and 7 represent the voltage reference for the MCU. In this case, it would set it to AVcc with external capacitor at ARef Pin on the MCU.

Bit 5 represents the ADC Left Adjust Result. This is cleared, by adjusting this in an algorithm ensures that the result is assigned appropriately, and no conversion starts until the result is obtained.

Bits 3:0 represents Analogue Channel Selection. All cleared selects ADC0, thus, the one used by the sensor.

**Figure 7.2.7 – ADCSRA Register Byte**

ADCSRA Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit No. |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | HEX |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0x8F |

The above register should be set to that exact representation upon initialization.

Bit 7 represents ADC Enable. Thus, ADEN set to 1 turns on the ADC.

Bit 6 represents ADC Start Conversion. This bit needs to be set when a conversion is required. Hence, ADC package procedure Start Conversion is crucial.

Bit 5 represents Auto Trigger Enabled. Since the software decides when to start conversion, this is disabled.

Bit 4 represents the interrupt flag, and it is set as soon as a conversion is complete automatically. Thus, prior to starting a new conversion, this flag needs to be reset by writing a logical 1 to it. Furthermore, another procedure that checks whether the conversion has completed or not is crucial. This procedure will be called by Start Conversion and will only start the conversion on the return of a True state.

Bit 3 represents ADC Interrupt Enable. Thus, ADIE is set to be able to run more than 1 conversion.

Bit 2:0 represents the ADC Prescaler Select. It determines the division factor between the system clock frequency and the input clock to the ADC. All set formulates to a 128-division factor.

The ADC being used is a 10bit value. Hence, 1024. To calculate voltage from an ADC result, a divisive approach is used. **Max voltage (5.0) / 1024 = lowest significant bit (LSB)**. Thus, rearranging the formula: **LSB * ADC result = Voltage**.

**Pseudo Code – Initialize ADC:**

```
procedure Init_ADC
        Initialise ADC Port(C) as Input.
        Set ADMUX Register to "01000000" or 0x40
        SET ADCSRA Register to "10001111" or 0x8F
end Init_ADC
```

**Pseudo Code – Start ADC Conversion**

```
procedure Start_Conversion
        Set ADSC to True
        Set Results Completed to False
end Start_Conversion
```

**Pseudo Code – Conversion Complete**

```
function Conversion_Complete returns boolean
       return ADC interrupt flag not equal to false
end Conversion_Complete
```

**Pseudo Code – Get Conversion Result**

```
function get_result returns result
        if Conversion is complete
                reset interrupt flag by writing a logical one
                get ADC Low Byte result
                get ADC High Byte result
                set result completed
        else
                return (previous) result
        end if
        return result
end get_result
```

**Pseudo Code – Convert result to Volts**

```
function Convert_ADC_Result_to_Volts returns Volts
   declare ADC_reading of type result
   declare ADC_reading_in_Volts of type volts
   declare low_significant_bit as constant with lowest resolution (5Volts/1024) : 0.004882812
        ADC_reading becomes result
        declare ADC_10_Bit of type 16_bit_Unsigned
        ADC_10_Bit becomes ADC High Byte and ADC Low Byte concatenated appropriately (Shifted)
        calculate_reading
          ADC_reading_in_Volts becomes LSB * ADC_10_Bit converted to type volts
        end calculate_reading
        return ADC_Reading_In_Volts
end Convert_ADC_Result_to_Volts
```

As a result, the design was completed. The implementation will follow the above pseudo codes and will be adapted in conformance with the Ada compiler and best practice.

## Cycle 4 – Implementation: ADC Setup

### Figure 7.2.8 – Bits of Interest & Procedure Initialize

```
with AVR.MCU;                           use AVR;

--ADC Package used by Main
package body ADC is

        --Bits of interest

        ADC_Start_Conversion    : Boolean renames MCU.ADCSRA_Bits (MCU.ADSC_Bit);
        ADC_Interrupt_Flag              : Boolean renames MCU.ADCSRA_Bits (MCU.ADIF_Bit);

-------------------------------------------------------------------------------------------------------------------
        procedure Initialize is
        begin
                --set Analogue Port to Input
                        MCU.DDRC_Bits := (others => DD_Input);

                --ADC Multiplexer Selection Register := ADMUX:
                --Select Reference Source : AVcc := 5v : REFS1:=0 REFS0:=1
                --Select ADC Channel to Read : MUX3:0 := 0000 Channel0 PortC0
                --Set_ADC Left Adjust Result : ADLAR := 1 or 0 to Right_Adjust ---> A328p.ADMUX := 2#0100_0000#
                        MCU.ADMUX :=  MCU.ADMUX and 2#0100_0000#;

                --ADC Status and Control Register := ADCSRA:
                --Select Prescaler 128 := ADPS2:0 : 111 --> 16MHZ/128 = 125KHz
                --Set: ADENable:= 1 : ADSC:= 0 (StartConversion when required)
                --Set: ADInteruptFlag : ADIF:= 0 : ADIE:= 0 ---> A328P.ADCSRA := 2#1000_0111#
                        MCU.ADCSRA := 2#1000_0111#;

                --Disable Digital Input for the ADC Pins: A328P.DIDR0 := 2#1111_1111# --reduces power consumption during buffer
                        MCU.DIDR0 := 2#1111_1111#;

        end Initialize;

-------------------------------------------------------------------------------------------------------------------
```

### Figure 7.2.9 – Procedure Start Conversion

```
-------------------------------------------------------------------------------------------------------------------
        procedure Start_Conversion is

                Result : Result_Record;

        begin
                -- The Start_Convertion () routine is called whenever the application needs an ADC conversion.

                -- Set the Start Conversion bit (ADSC:6) in ADCSRA to start a single conversion.

                 ADC_Start_Conversion := True;

                -- Poll (wait) for the Interrupt Flag (ADIF) bit in the ADCSRA register to be set,
                -- indicating that a new conversion is completed.
                -- At this instance conversion completion is active.. i.e.

                Result.Completed := False;

        end Start_Conversion;
-------------------------------------------------------------------------------------------------------------------
```

### Figure 7.2.10 – Procedure Conversion Complete

```
-------------------------------------------------------------------------------------------------------------------
        function Conv_Complete return Boolean is

                --  Once the conversion is over (ADIF bit ADCSRA:4) becomes high,
                --  then read the ADC data register pair (ADCL/ADCH) to get the 10-bit result.
        begin

                return ADC_Interrupt_Flag /= False;             -- ADIF is set

        end Conv_Complete;

-------------------------------------------------------------------------------------------------------------------
```

**Figure 7.2.11 – Procedure Get Result**

```
function Get_Result return Result_Record is
        Result : Result_Record;
begin

        if Conv_Complete then -- ADIF is set

                --DS:ADIF is cleared by writing a logical one to the flag.

                ADC_Interrupt_Flag := True;

                Result.Low_Byte         := MCU.ADCL; -- ADC Data Register Low Byte
                Result.High_Byte        := MCU.ADCH; -- ADC Data Register High Byte
                Result.Completed        := True;

        else

                return Result;

        end if;

        return Result;

end Get_Result;
```

**Figure 7.2.12 – Procedure Convert to Volts**

```
function Compute_Volts_From_ADC return ADC.To_Volts is
    -- Declare Variables of type Result_Record, To_Volts and Lowest Significant bit of a 10Bit Unsigned value
    ADC_Reading                 : Result_Record;
    ADC_Reading_in_Volts        : ADC.To_Volts;
    -- LSB -- Specify Resolution of 10Bit: 5.0/1024. Since delta is 0.005, some precision is lost. however, it is minor.
    LSB10                       : constant Float := 0.004882812;

begin
    -- Get ADC Result
    ADC_Reading := ADC.Get_Result;

    declare         -- Code Block
        -- new 16Bit type to store our value
        ADC_10_Bit :  ADC_16bit_Unsigned;
    begin
            --  Shift Left 8 spaces our ADCH Byte to create our 10 bit by concatenating our ADCL byte on the lower 0 - 7 bits
        ADC_10_Bit  := Shift_Left(ADC_16bit_Unsigned(ADC_Reading.High_Byte),8)
                                        + ADC_16bit_Unsigned(ADC_Reading.Low_Byte);
        Calculate_Reading_10:   -- New Code Block to perform our final calculation

        declare
            -- Perform checks using Compiler Directive pragma Suppress:
            pragma SUPPRESS(ALL_CHECKS);
            -- This will give permission to this implementation to omit certain language-defined checks
        begin

            -- Multiply our LSB by the 10 bit value to get reading in volts. Convert to the required type: To_Volts

            ADC_Reading_in_Volts   := ADC.To_Volts(LSB10 * Float(ADC_10_Bit ));

        end Calculate_Reading_10;
    end;

    return ADC_Reading_in_Volts;
end Compute_Volts_From_ADC;
```

Pseudo Code was successfully employed to the above implementation. However, Ada requires two packages as previously discussed: implementation and specification file. Thus, the Specification file was implemented – Figure 7.2.13. Pragma Suppress All Checks was used in Procedure Convert to Volts as it allows the implementation to omit certain language-defined checks, making a more efficient conversion and calculation, as stated in the Ada manual.

**Figure 7.2.13 – Specification File**

```
with Interfaces;                use Interfaces;

package ADC is
        type ADC_16bit_Unsigned is new Unsigned_16 range 0 .. 2**10-1;

        -- Record Type for Values of interest
        type Result_Record is
          record
                High_Byte : Unsigned_8 := 0; -- ADCH
                Low_Byte  : Unsigned_8 := 0; -- ADCL
                Completed : Boolean := False;
         end record;

        --Delta specified must be a power of 2, so it is used directly,
        --Unless using a specific range - delta must be a power of 10 - ARM Page 81
        --And must be larger than the lowest significant bit 5.0 / 1024 = 0.0048828125;
        type To_Volts is delta 0.005 range 0.0 .. 5.0; -- 5.0 volts
        for  To_Volts'Size use 16;

        procedure Initialize;

        procedure Start_Conversion;

        function Conv_Complete return Boolean;

        function Get_Result return Result_Record;

        function Compute_Volts_From_ADC return To_Volts;
end ADC;
```

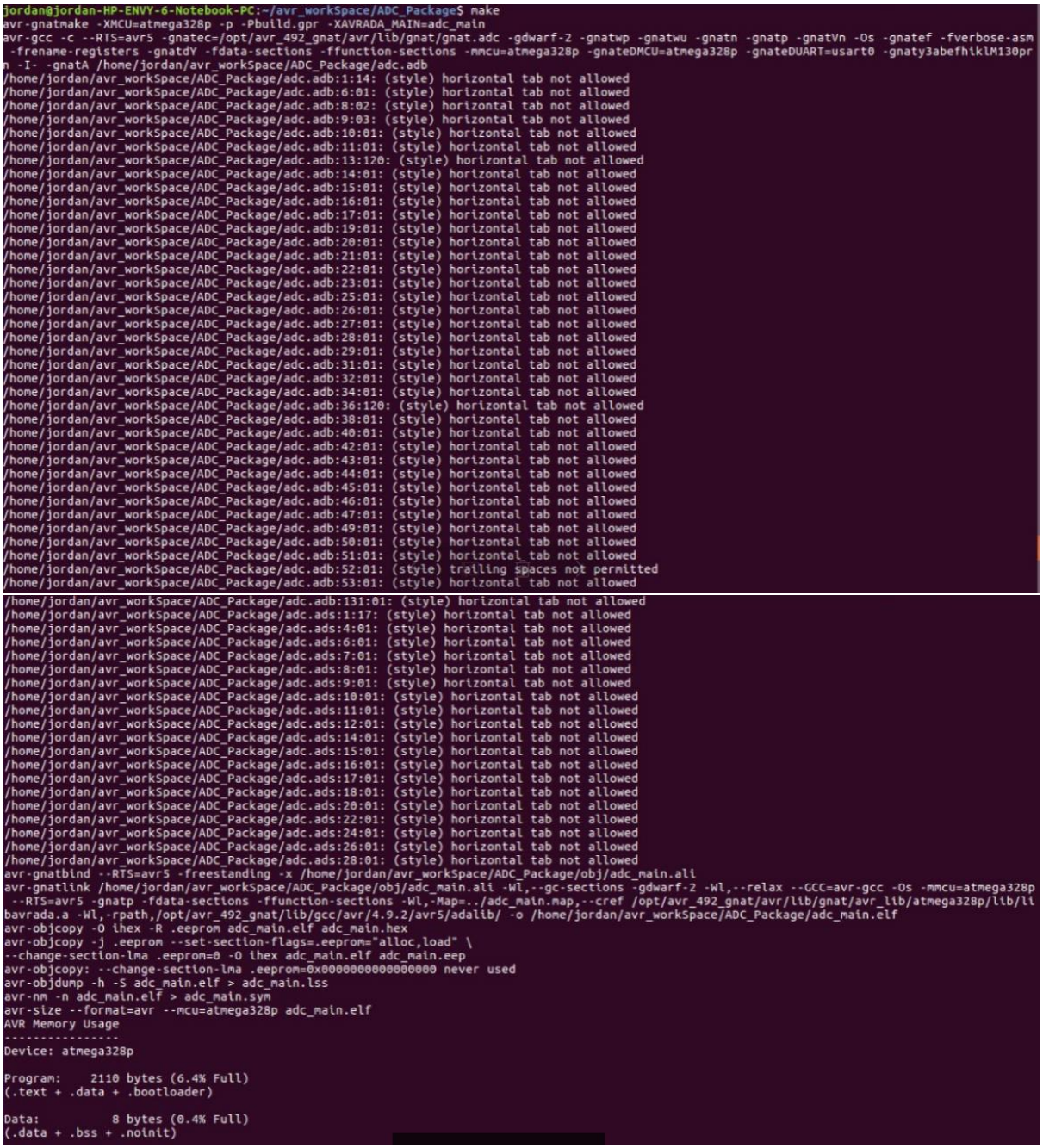
Moreover, an algorithm was implemented for testing purposes – Figure 7.2.14.

**Figure 7.2.14 – ADC_Main**

```
with AVR;
with AVR.MCU;                           use AVR;
with ADC;                               use ADC;
procedure ADC_Main is
        LED : Boolean renames MCU.PortB_Bits(1);
        LED0 : Boolean renames MCU.PortB_Bits(0);
        ADC_Val : ADC.To_Volts;
begin
        MCU.DDRB_Bits (1) := DD_Output;
        MCU.DDRB_Bits (0) := DD_Output;

        LED := False;
        LED0 := False;
        ADC.Initialize;
        loop
                ADC.Start_Conversion;
                if ADC.Conv_Complete then
                        ADC_Val := ADC.Compute_Volts_From_ADC;
                        if ADC_Val > 4.0 then -- 4volts
                                LED := True;
                                LED0 := False;
                                delay 0.1;
                        elsif ADC_Val > 3.0 then --3volts
                                LED0 := True;
                                LED := False ;
                                delay 0.1;
                        elsif ADC_Val > 2.0 then --2volts
                                LED := False;
                                LED0 := False;
                                delay 0.1;
                        elsif ADC_Val > 1.0 then --1volt
                                LED := True;
                                LED0 := True;
                                delay 0.1;
                        end if;
                end if;
        end loop;
end ADC_Main;
```

Both ADC Package and ADC_Main was successfully compiled – Figure 7.2.15 and tested.

## Figure 7.2.15 – ADC Package Compiled



**Cycle 4 – Testing: ADC Setup**

**Figure 7.2.16 – Testing ADC_Main (1 - 2V)**

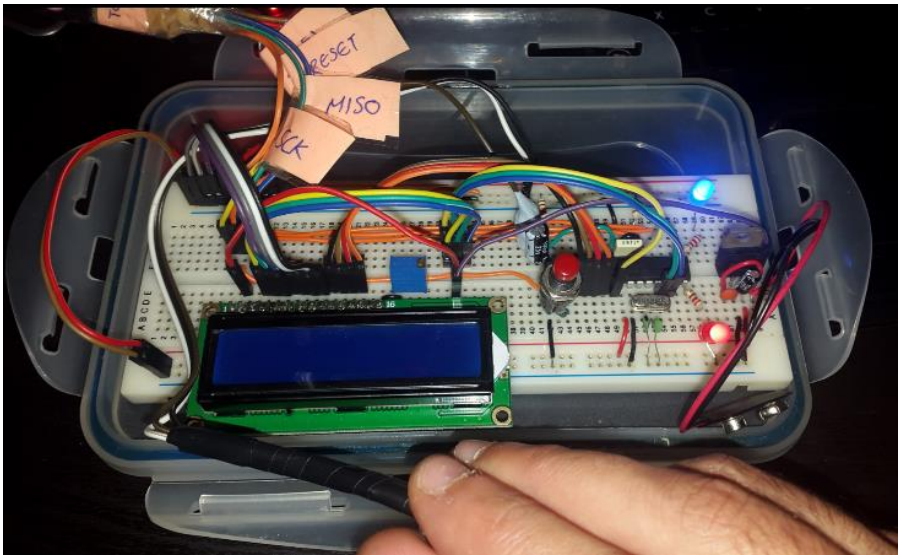
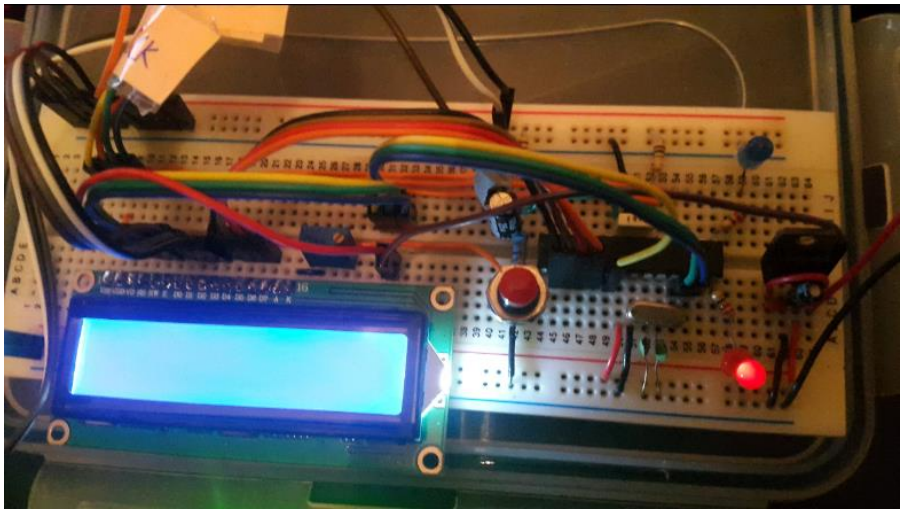



**Figure 7.2.17 – Testing ADC_Main (3 - 4V)**



Cycle 4 – Evaluation

The ADC setup was successfully employed and tested using LEDs. The testing method used is known as Unit Testing, as it purely focuses on the ADC Package's functionality. The main concern was testing the voltage conversion, as it is what defines the state of the signal. Thus, using a different logical state on each defined voltage value provided a visual representation of the signal. The signal was as expected, the LEDs flashed mainly between 1 and 4 volts and concluded to an efficient and complete algorithm. The system is now ready for Cycle 5.

Cycle 5 – LCD Setup

- Refined software design with improvements – LCD Setup.
- **Product:** System running and tested by displaying strings on LCD.

**Cycle 5 – Design & Implementation: LCD_Setup**

In this cycle an objective was to be proven, sometimes all that is required to design a working solution is a good understanding of programs, knowing where to find them, how to read them, how they work and how to use them to one's advantage. Thus, after pursuing a thorough search and coming across the LCD example package within the AVR-Ada library, how the whole LCD package operated, executed and what files/packages were being used was revised– Appendix 7A:C. Subsequently, a separate package was created and the code adapted to fulfil the system's requirements.

Design & Implementation are both concatenated in this section as it purely focuses on developing a new functionality from reading, understanding and employing code to the system's advantage. Furthermore, library examples provide an inclined learning curve towards the programmer's knowledge as it uses best practice programming and distinct programming techniques, useful in multiple programming environments. Moreover, the LCD datasheet is paramount as it provides a thorough understanding on the LCD requirements and why are things programmed the way they are.

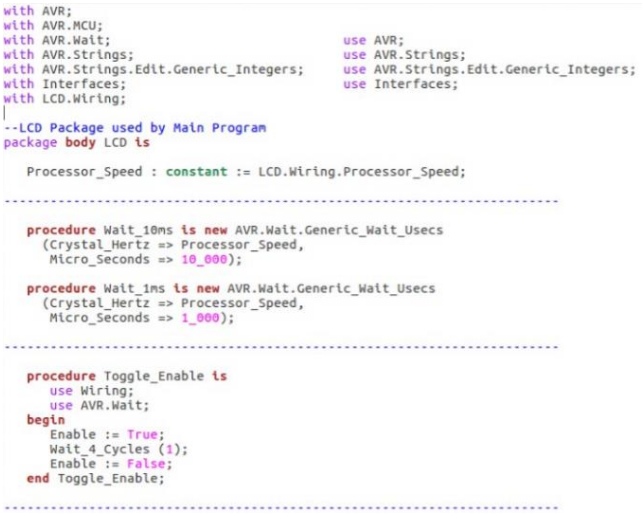
The following figures represent the original LCD package altered to the system's advantage.

**Figure 7.2.18 – LCD Wiring**

```
with AVR;                          use AVR;
with AVR.MCU;                      use AVR.MCU;
with AVR.Atmega328P;
with Interfaces; use Interfaces;
private package LCD.Wiring is
   pragma Preelaborate;
   --LCD data buses
   Data0             : Boolean renames MCU.PORTD_Bits (PORTD0_Bit);
   Data1             : Boolean renames MCU.PORTD_Bits (PORTD1_Bit);
   Data2             : Boolean renames MCU.PORTD_Bits (PORTD2_Bit);
   Data3             : Boolean renames MCU.PORTD_Bits (PORTD3_Bit);
   Data0_DD          : Boolean renames MCU.DDRD_Bits (DDD0_Bit);
   Data1_DD          : Boolean renames MCU.DDRD_Bits (DDD1_Bit);
   Data2_DD          : Boolean renames MCU.DDRD_Bits (DDD2_Bit);
   Data3_DD          : Boolean renames MCU.DDRD_Bits (DDD3_Bit);

   --LCD control bits

   RegisterSelect    : Boolean renames MCU.PORTD_Bits (PORTD4_Bit);
   RegisterSelect_DD : Boolean renames MCU.DDRD_Bits (DDD4_Bit);
   ReadWrite         : Boolean renames MCU.PORTD_Bits (PORTD5_Bit);
   ReadWrite_DD      : Boolean renames MCU.DDRD_Bits (DDD5_Bit);
   Enable            : Boolean renames MCU.PORTD_Bits (PORTD6_Bit);
   Enable_DD         : Boolean renames MCU.DDRD_Bits (DDD6_Bit);

   --processor

   Processor_Speed   : constant := 16_000_000;

end LCD.Wiring;
```

The LCD wiring relates to LCD wires connected on the MCU. Furthermore, the setup deliberates that the LCD is connected in 4 Bit Mode (4 Data Busses). The control bits are used to Read/Write, Select Registers and Enable/Disable bits when performing operations. The Processor speed is initiated in this package.

**Figure 7.2.19 – LCD Package Body**



```
with AVR;
with AVR.MCU;
with AVR.Wait;                              use AVR;
with AVR.Strings;                           use AVR.Strings;
with AVR.Strings.Edit.Generic_Integers;     use AVR.Strings.Edit.Generic_Integers;
with Interfaces;                            use Interfaces;
with LCD.Wiring;

--LCD Package used by Main Program
package body LCD is

   Processor_Speed : constant := LCD.Wiring.Processor_Speed;

-------------------------------------------------------------------------

   procedure Wait_10ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 10_000);

   procedure Wait_1ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 1_000);

-------------------------------------------------------------------------

   procedure Toggle_Enable is
      use Wiring;
      use AVR.Wait;
   begin
      Enable := True;
      Wait_4_Cycles (1);
      Enable := False;
   end Toggle_Enable;

-------------------------------------------------------------------------
```

To perform an operation, a command is written to the register and the Enable bit must be toggled for the command to be sent successfully. Hence, Toggle Enabled and specific delays (Wait_Xms) are required.

**Figure 7.2.20 – LCD Specification File**

```
with Interfaces;                use Interfaces;
with AVR.Strings;                       use AVR.Strings;

package LCD is
        pragma Preelaborate;

        type Line_Position is new Unsigned_8 range 1 .. 2;
        type Char_Position is new Unsigned_8 range 1 .. 16;
        type Command_Type is new Unsigned_8;

-------------------------------------------------------------------------

        procedure Toggle_Enable;

        procedure Output (Cmd : Unsigned_8; Is_Data : Boolean := False);

        procedure Initialize;

        procedure Put (C : Character);

        procedure Put (S : AVR_String);

        procedure Command (Cmd : Command_Type);

        procedure Clear_Screen;

        procedure Home;

        procedure GotoXY (X : Char_Position; Y : Line_Position);

        procedure Display_Int(Result : in Integer);
-------------------------------------------------------------------------

        package Commands is
                Clear                   : constant Command_Type := 16#01#;
                Home                    : constant Command_Type := 16#02#;

                --  interface data width and number of lines
                Mode_4bit_1line         : constant Command_Type := 16#20#;
                Mode_4bit_2line         : constant Command_Type := 16#28#;
                Mode_8bit_1line         : constant Command_Type := 16#30#;
                Mode_8bit_2line         : constant Command_Type := 16#38#;

                --  display on/off, cursor on/off, blinking char at cursor position
                Display_Off             : constant Command_Type := 16#08#;
                Display_On              : constant Command_Type := 16#0C#;
                Display_On_Blink        : constant Command_Type := 16#0D#;
                Display_On_Cursor       : constant Command_Type := 16#0E#;
                Display_On_Cursor_Blink : constant Command_Type := 16#0F#;

                --  entry mode
                Entry_Inc               : constant Command_Type := 16#06#;
                Entry_Dec               : constant Command_Type := 16#04#;
                Entry_Shift_Inc         : constant Command_Type := 16#07#;
                Entry_Shift_Dec         : constant Command_Type := 16#05#;

                --  cursor/shift display
                Move_Cursor_Left        : constant Command_Type := 16#10#;
                Move_Cursor_Right       : constant Command_Type := 16#14#;
                Move_Display_Left       : constant Command_Type := 16#18#;
                Move_Display_Right      : constant Command_Type := 16#1C#;
        end Commands;

private
        pragma Inline (Command);
```

Command Package located within the LCD specification file resembles all commands displayed in the LCD Datasheet.

**Figure 7.2.21 – LCD Package Procedure Initialize**

```
procedure Initialize is
        use LCD.Wiring;
begin
        --Make PortD as Output using LCD-Wiring
        Enable_DD         := DD_Output;
        ReadWrite_DD      := DD_Output;
        RegisterSelect_DD := DD_Output;
        Data0_DD := DD_Output;
        Data1_DD := DD_Output;
        Data2_DD := DD_Output;
        Data3_DD := DD_Output;

        --  power up delay required 16ms
        Wait_10ms;
        Wait_10ms;

        --  send command 0x30 3 times to reset the LCD and tell it: it will be used in 4 bit mode.
        Data0 := True;
        Data1 := True;
        Toggle_Enable;
        Wait_10ms;
        --  send last command again (is still in register, just toggle E)
        Toggle_Enable;
        Wait_1ms;
        --  send last command a third time
        Toggle_Enable;
        Wait_1ms;
        --  select bus width (0x30 - for 8-bit and 0x20 for 4-bit) in this case 0x20.
        Data0 := False;
        --  send new command to the command register to enable 4bit.
        Toggle_Enable;
        Wait_1ms;
        --  select modes for operation
        Command (Commands.Mode_4bit_2line);
        Command (Commands.Display_On);
        Clear_Screen;
        Command (Commands.Entry_Inc);
end Initialize;
```

In the above illustration, Port D on the MCU is initialised as Output. The LCD requires a 16ms time frame to power up effectively and 4 Bit Mode, 2 lines, Display On and Entry Increment by 1 requires initialisation. Toggle Enabled is used to send the command stored in the register to the LCD, Toggle Disabled indicates end of operation.

The LCD Datasheet specifically states that the command 0x30 must be written three times to enable 4 Bit Mode. While the command is written in the register, the procedure toggle enable and a millisecond delay in between sends the command. Once 4 Bit Mode has been initialized, a 4 Bit bus must be set up using 0x20 hex command. Thus, Data0 is the only one cleared as Data1 remains true in the register – Figure 7.2.22. In addition, entry modes can be called directly as all modes use the higher nibble only – Figure 7.2.23.

**Figure 7.2.22 – Data Register Operation**

| 3 | 2 | 1 | 0 | DataNo. |
|---|---|---|---|---|
| 8 | 4 | 2 | 1 | Bit Value |
| F | F | T | T | |
| | | 0x30 | | Register |

| 3 | 2 | 1 | 0 | DataNo. |
|---|---|---|---|---|
| 8 | 4 | 2 | 1 | Bit Value |
| F | F | T | F | |
| | | 0x20 | | Register |

**Figure 7.2.23 – Entry Modes: Higher Nibble**

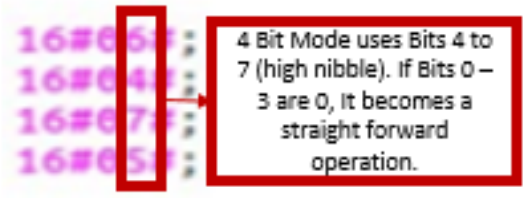
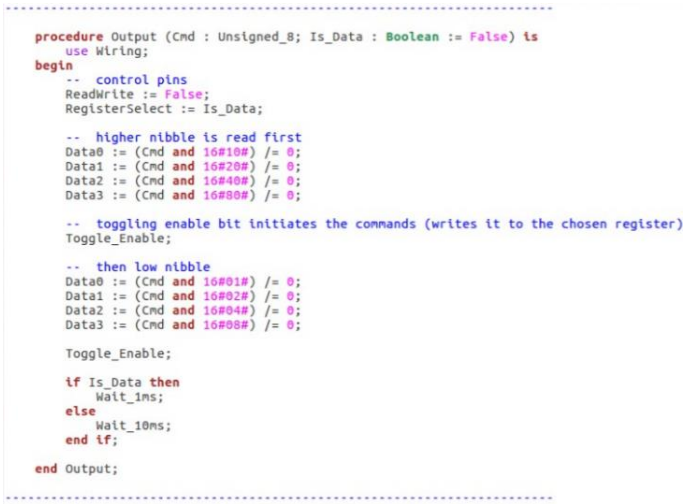


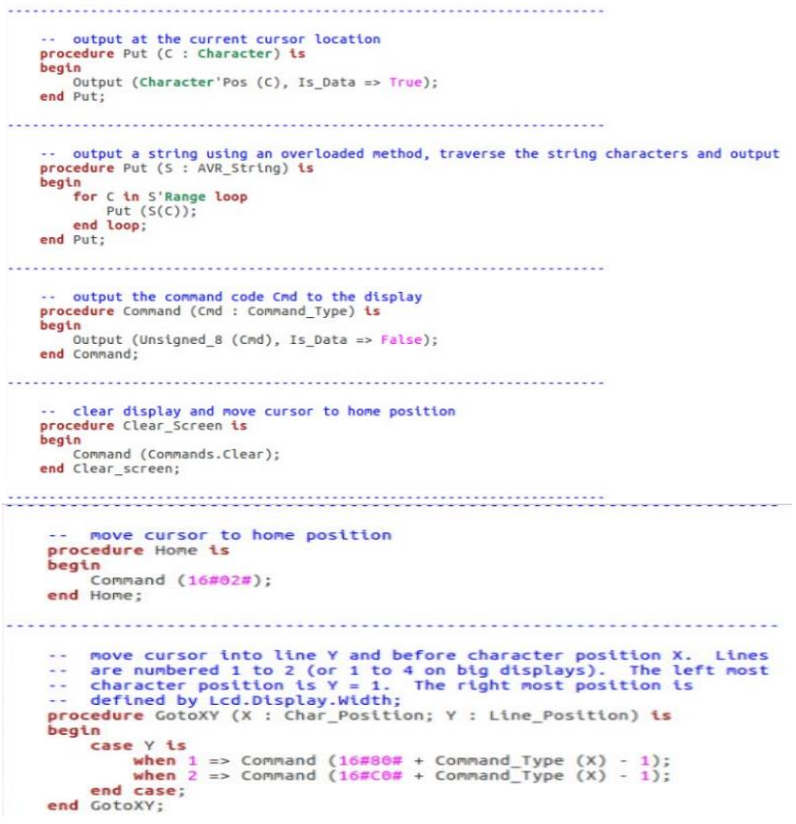
**Figure 7.2.24 – LCD Package Procedure Output**



```
procedure Output (Cmd : Unsigned_8; Is_Data : Boolean := False) is
    use Wiring;
begin
    --  control pins
    ReadWrite := False;
    RegisterSelect := Is_Data;

    --  higher nibble is read first
    Data0 := (Cmd and 16#10#) /= 0;
    Data1 := (Cmd and 16#20#) /= 0;
    Data2 := (Cmd and 16#40#) /= 0;
    Data3 := (Cmd and 16#80#) /= 0;

    --  toggling enable bit initiates the commands (writes it to the chosen register)
    Toggle_Enable;

    --  then low nibble
    Data0 := (Cmd and 16#01#) /= 0;
    Data1 := (Cmd and 16#02#) /= 0;
    Data2 := (Cmd and 16#04#) /= 0;
    Data3 := (Cmd and 16#08#) /= 0;

    Toggle_Enable;

    if Is_Data then
        Wait_1ms;
    else
        Wait_10ms;
    end if;

end Output;
```

The most crucial procedure within the LCD package (disregarding initialisation) is Output. Output is responsible for reading data brought to the procedure and assigning it to the register. The toggle enable procedure as previously explained, sends out the address. It is an incredible methodology used for such implementation/execution. In addition, as stated on the LCD Datasheet, register select used to write data requires less waiting time than register select used to write commands.

**Figure 7.2.25 – Procedures Put, Command, Clear**



```
--  output at the current cursor location
procedure Put (C : Character) is
begin
    Output (Character'Pos (C), Is_Data => True);
end Put;

--  output a string using an overloaded method, traverse the string characters and output
procedure Put (S : AVR_String) is
begin
    for C in S'Range loop
        Put (S(C));
    end loop;
end Put;

--  output the command code Cmd to the display
procedure Command (Cmd : Command_Type) is
begin
    Output (Unsigned_8 (Cmd), Is_Data => False);
end Command;

--  clear display and move cursor to home position
procedure Clear_Screen is
begin
    Command (Commands.Clear);
end Clear_screen;

--  move cursor to home position
procedure Home is
begin
    Command (16#02#);
end Home;

--  move cursor into line Y and before character position X.  Lines
--  are numbered 1 to 2 (or 1 to 4 on big displays).  The left most
--  character position is Y = 1.  The right most position is
--  defined by Lcd.Display.Width;
procedure GotoXY (X : Char_Position; Y : Line_Position) is
begin
    case Y is
        when 1 => Command (16#80# + Command_Type (X) - 1);
        when 2 => Command (16#C0# + Command_Type (X) - 1);
    end case;
end GotoXY;
```

The above procedures were used directly and remain unchanged. However, the LCD Package had one concerning aspect. Integers could not be converted into a String generically, thus, a new method was deliberated to solve the issue – Figure 7.2.26.

**Figure 7.2.26 – Procedure Display Integer**

```
procedure Display_Int(Result : in Integer) is
    L : Unsigned_8;
    T : AVR_String(1 .. L) := (others => ' ');
begin
        Put_U32 (Value  => Unsigned_32(Result),
            Target => T,
            Last   => L);
        Put(T(2..L)); --displays the integer into a string
end Display_Int;
```

Display Integer takes in an integer as input, converts it to unsigned 32 and uses a method to convert it into an AVR.String. Subsequently, it displays the integer as a string using a "put" method within the same AVR.Strings.X package.

In addition, an LCD Main package was developed for testing purposes, using the previous ADC Package, in conjunction to the new LCD Package. Deliberating an incremental development on each cycle of the iterative design model – Figure 7.2.27.

**Figure 7.2.27 – LCD_Main**

```
with AVR;
with AVR.MCU;                                   use AVR;
with ADC;                                       use ADC;
with LCD;

procedure LCD_Main is
        LED : Boolean renames MCU.PortB_Bits(1);
        LED0 : Boolean renames MCU.PortB_Bits(0);
        --ADCInt : Unsigned_8 := 1000_0000;
        ADC_Val : ADC.To_Volts;
begin
        MCU.DDRB_Bits (1) := DD_Output;
        MCU.DDRB_Bits (0) := DD_Output;
        LED := False;
        LED0 := False;
        ADC.Initialize;
        LCD.Initialize;
        LCD.Clear_Screen;
         loop
                ADC.Start_Conversion;
                if ADC.Conv_Complete then
                        LCD.Clear_Screen;
                        ADC_Val := ADC.Compute_Volts_From_ADC;
                        LCD.GotoXY(1, 2);
                        LCD.Put("HR Monitor");
                        LCD.Home;
                        LCD.Put ("Pulse Voltage: ");
                        LCD.GotoXY(15, 1);
                        if ADC_Val > 4.0 then -- 4volts
                                LED := True;
                                LED0 := False;
                                delay 0.1;
                        elsif ADC_Val > 3.0 then --3volts
                                LED0 := True;
                                LED := False ;
                                delay 0.1;
                        elsif ADC_Val > 2.0 then --2volts
                                LED := False;
                                LED0 := False;
                                delay 0.1;
                        elsif ADC_Val > 1.0 then --1volt
                                LED := True;
                                LED0 := True;
                                delay 0.1;
                        end if;
                        LCD.Display_Int(Integer(ADC_Val)); --displays the integer into a string
                        LCD.GotoXY(16, 1);
                        LCD.Put("v");
                        delay 0.15;
                end if;
        end loop;
end LCD_Main;
```

The LCD methods being tested are methods that are relevant on future cycles. Hence, Initialize, Home, Go to XY position, Put, Clear, Display Integer are all crucial methods. This testing method is known as integration testing as two packages are involved, though considered as a unit test, as ADC was already tested in isolation.

**Figure 7.2.28 – LCD Main Compile**

```
jordan@jordan-HP-ENVY-6-Notebook-PC:~/avr_workSpace/LCD_Package$ make
avr-gnatmake -XMCU=atmega328p -p -Pbuild.gpr -XAVRADA_MAIN=lcd_main
avr-gcc -c --RTS=avr5 -gnatec=/opt/avr_492_gnat/avr/lib/gnat/gnat.adc -gdwarf-2 -gnatwp -gnatwu -gnatn -gnatp -gnatVn -Os -gnatef -fverbose-asm
 -frename-registers -gnatdY -fdata-sections -ffunction-sections -mmcu=atmega328p -gnateDMCU=atmega328p -gnateDUART=usart0 -gnaty3abefhiklM130pr
n -I- -gnatA /home/jordan/avr_workSpace/LCD_Package/lcd_main.adb
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:2:14: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:3:10: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:7:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:7:36: (style) bad casing of "PORTB_Bits" declared at avr-atmega328p.ads:2449
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:8:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:8:37: (style) bad casing of "PORTB_Bits" declared at avr-atmega328p.ads:2449
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:9:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:10:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:12:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:13:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:14:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:15:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:16:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:17:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:18:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:19:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:20:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:21:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:22:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:23:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:24:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:25:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:26:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:27:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:28:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:29:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:30:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:31:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:32:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:33:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:34:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:35:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:32:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:33:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:34:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:35:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:36:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:37:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:38:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:39:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:40:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:41:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:42:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:43:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:44:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:45:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:46:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:47:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:48:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:49:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:50:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/LCD_Package/lcd_main.adb:51:01: (style) horizontal tab not allowed
avr-gnatbind --RTS=avr5 -freestanding -x /home/jordan/avr_workSpace/LCD_Package/obj/lcd_main.ali
avr-gnatlink /home/jordan/avr_workSpace/LCD_Package/obj/lcd_main.ali -Wl,--gc-sections -gdwarf-2 -Wl,--relax --GCC=avr-gcc -Os -mmcu=atmega328p
 --RTS=avr5 -gnatp -fdata-sections -ffunction-sections -Wl,-Map=../lcd_main.map,--cref /opt/avr_492_gnat/avr/lib/gnat/avr_lib/atmega328p/lib/li
bavrada.a -Wl,-rpath,/opt/avr_492_gnat/lib/gcc/avr/4.9.2/avr5/adalib/ -o /home/jordan/avr_workSpace/LCD_Package/lcd_main.elf
avr-objcopy -O ihex -R .eeprom lcd_main.elf lcd_main.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 -O ihex lcd_main.elf lcd_main.eep
avr-objcopy: --change-section-lma .eeprom=0x0000000000000000 never used
avr-objdump -h -S lcd_main.elf > lcd_main.lss
avr-nm -n lcd_main.elf > lcd_main.sym
avr-size --format=avr --mcu=atmega328p lcd_main.elf
AVR Memory Usage
----------------
Device: atmega328p

Program:    3090 bytes (9.4% Full)
(.text + .data + .bootloader)

Data:         48 bytes (2.3% Full)
(.data + .bss + .noinit)
```

**Cycle 5 – Testing: LCD Setup**

Integration/Unit testing was performed using the previously tested unit (ADC), in conjunction with the new LCD design – Figure 7.2.29.

**Figure 7.2.29 – LCD Integration Testing**

The system responded as expected, both LEDs off in response to a signal between 1 and 2 volts. Voltage is rounded off to the nearest integer when displayed. Although, this shows a deficiency on precision, it does not affect the system as primarily, the display will be formulated from a counter of pulses or peaks using round numbers. Thus, this illustrates an effective result regarding LCD testing.

Cycle 5 – Evaluation

The LCD Package has followed a distinct approach that encompassed substantial programming techniques. The LCD is extremely vital in this project, as it is the only interface providing crucial information to the user. The program design is extremely efficient and the approach provides users with an enforcement towards build-up of programs when using library examples. Although, in occasions, certain aspects of learning unfamiliar code can become tremendously tedious. Revising the requirements specified in a datasheet or specification can provide a sense of interpretation that fulfils the comprehension necessary for pursuing certain manipulations. Thus, having tested the LCD Package and accomplishing successful results, the ADC and LCD are both ready for Cycle 6; Pulse Detection Algorithm.

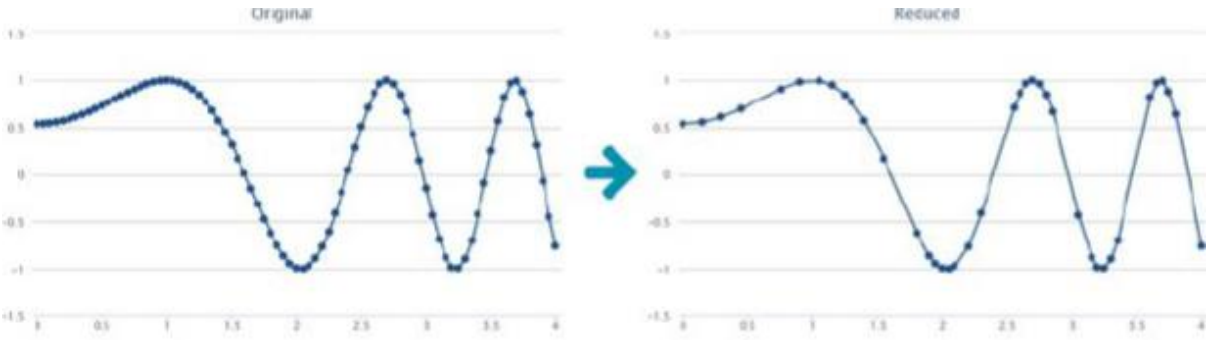
Cycle 6 – Pulse Detection Algorithm

- Refined software design with improvements – Pulse Detection Algorithm.
- **Product:** System detecting peaks on an estimated timeframe of 10 seconds, tested by displaying BPM on LCD.

In the past, research was conducted on algorithms regarding peak detection as part of CSCM10 – Coursework 2 – Medical Devices.

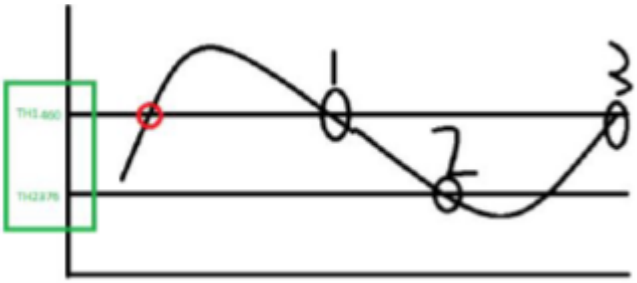
Algorithm techniques revised were known as:

**Figure 7.2.30 – Rammer-Douglas-Peucker (RDP) Algorithm**




The RDP Algorithm concept consists of dodging points which prove to be irrelevant. As a result, minimizes number of readings taken.

**Figure 7.2.31 – Threshold Analysis Algorithm**



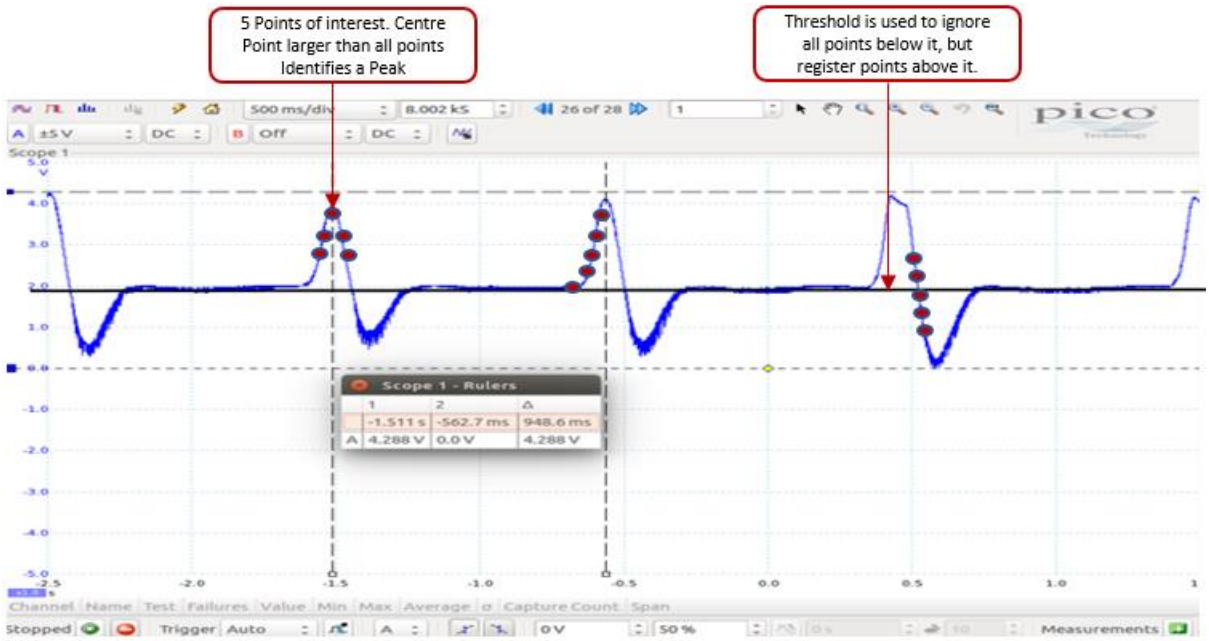
The Threshold Analysis Algorithm uses two thresholds that identifies when the signal is above the threshold, signalling a peak in every cycle.

**Cycle 6 – Design: Pulse Detection Algorithm**

The analysis on both algorithms brought a new and improved algorithm to light. One introduced and named as part of this thesis – Thresh Peak Algorithm.

The Thresh Peak Algorithm concept focuses on points of interest originating above a set threshold – Figure 7.2.32.

**Figure 7.2.32 – Thresh Peak Algorithm**




Point of interest are classified as the following:

- NextNext
- Next
- Current – Focal Point
- Prev
- PrevPrev

Current is the main point of interest; the Centre Point. On every peak detected, a counter named pulse is incremented and detection is performed throughout a 10 second time frame. Since a timer has not yet been implemented, an estimated time frame will be calculated using delays; 10 seconds / 2ms delay equals 5000 cycles. Thus, keeping a counter that increments on every cycle subsequent to a 2ms delay issues a 10 second delay on the 5000$^{th}$ iteration. In addition, the time lost between the execution of commands, 10 seconds should be estimated upon every 4000 – 4500 cycles. Trial and error will be issued to induce an accurate 10 second estimation before results are displayed. Results will then be multiplied by 6 to formulate results in *Beats Per Minute - BPM.*

Points of interest should be assigned upon a completed ADC conversion, otherwise, the original points already assigned remains.

**Pseudo Code – Pulse Detection Algorithm**

```
Pulse Peak Detection Algorithm
   Declare points of interest as ADC.To_Volts Type assigned to 0.0.
   Declare Counters and HR or BPM as Integers
   Initialize ADC and LCD.
   Clear LCD Screen
   Calibrate LCD - new procedure that adds an introduction to prepare sensor
   Start infinite_loop
      Start ADC Conversion
      if ADC Conversion has Completed
        assign NextNext with ADC Voltage From ADC Result
        delay 2ms
        counter becomes counter +1
        if NextNext is greater or equal to Threshold then
            PrevPrev becomes Prev
            Prev becomes Current
            Current becomes Next
            Next becomes NextNext
            if Current is greater than PrevPrev, Prev, Next and NextNext
                 Pulse becomes Pulse + 1
            Turn LED BLue on
            Clear LCDScreen
            if Counter is greater or equal to 4000
                 HR becomes pulse multiplied by 6
                 LCD Put ur HRate is
                 LCD GotoXY(7,2)
                 LCD Display Integer(HR)
                 Turn LED Blue off
                 Turn LED Red On
                 exit
             else
                 Clear LCD
                 LCD Put Calculating HR
     else
        PrevPrev becomes PrevPrev
        Prev becomes Prev
        Current becomes Current
        Next becomes Next
        NextNext becomes NextNext
end Pulse Detection Algorithm
```

The Thresh Peak Algorithm has been employed and now awaits implementation and testing.

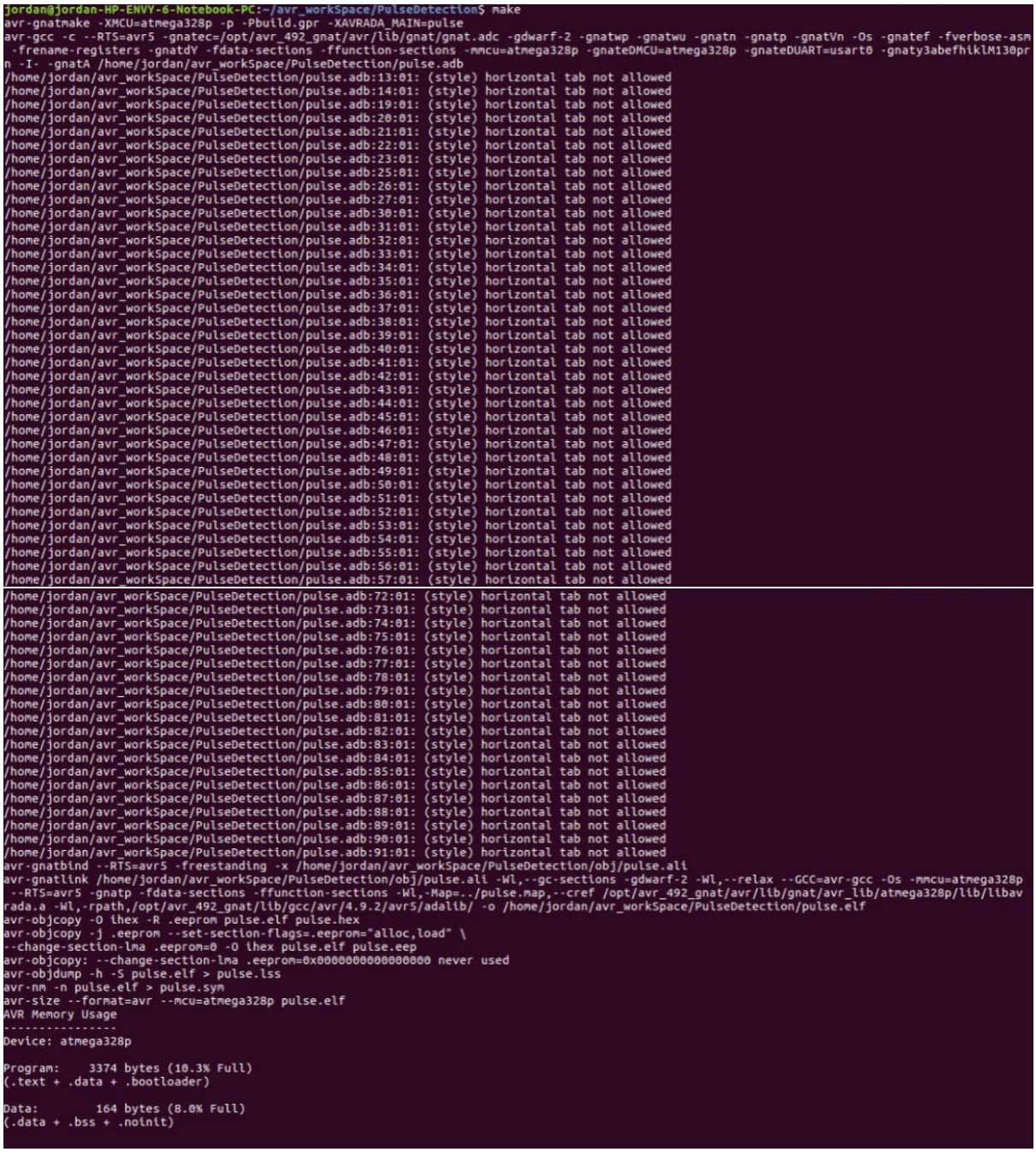
### Cycle 6 – Implementation: Pulse Detection Algorithm

The Pseudo Code Pulse Detection Algorithm was successfully implemented and compiled. The algorithm was employed to the Main method, also known as Controller and named Pulse for testing purposes – Figure 7.2.33.

**Figure 7.2.33 – Pulse Detection Algorithm Implementation**

```
with AVR;
with AVR.MCU;
use AVR;

with ADC;
use ADC;

with LCD;
use LCD;

--Pulse Program makes use of ADC and LCD packages (Timer Package will be added if testing of its functionality is accomplished)
procedure Pulse is
    LED  : Boolean renames MCU.PORTB_Bits(1);
    LED0 : Boolean renames MCU.PORTB_Bits(0);

-- Declare Values of interest globally
-- Initialize values of interest to a default value

    PrevPrev    : ADC.To_Volts := 0.0;
    Prev        : ADC.To_Volts := 0.0;
    Current     : ADC.To_Volts := 0.0;
    Next        : ADC.To_Volts := 0.0;
    NextNext    : ADC.To_Volts := 0.0;

    pulse   : Integer := 0;
    counter : Integer := 0;
    HR : Integer := 0;

begin
    MCU.DDRB_Bits (1) := DD_Output;
    MCU.DDRB_Bits (0) := DD_Output;
    --Initialize
    ADC.Initialize;
    LCD.Initialize;
    LCD.Clear_Screen;
    LCD.Calibrate_System_UI;
        loop
                -- Start Conversion
                ADC.Start_Conversion;
                -- We test if conversion is completed, if True (result is stored into the array)
                -- We increase the counter every time a conversion is complete and stored
                if ADC.Conv_Complete then

                    NextNext := ADC.Compute_Volts_From_ADC;

                    -- 2ms delay
                    delay 0.002;
                    counter := counter +1;

                    --threshold of 2.7 volts anything below is ignored
                    if (NextNext >= 2.7) then
                        -- Assign Values of Interest
                        PrevPrev := Prev;
                        Prev     := Current;
                        Current  := Next;
                        Next     := NextNext;

                        --if current is bigger than all values of interest then we have located a peak/pulse
                        if (Current > Prev and Current > PrevPrev and Current > Next and Current > NextNext) then
                            --pulse detected so we increment number of pulses
                            pulse := pulse + 1;
                        end if;

                       -- blue LED On
                       LED := True;

                       LCD.Clear_Screen;

                       --counter 4000 is the delay calculated for a 10 second cycle
                       --due to missed compares or unexpected increment counter ==4000 condition may never be true.
                       --Thus to remain on the safer side, i use '>=' instead of '=='.
                       if (counter >= 4000) then
                           -- calculate BPM and display it
                           HR := pulse * 6;
                           LCD.Put("ur HRate is:");
                           LCD.GotoXY(7,2); --7,2
                           LCD.Display_Int(HR);
                           LED := False;
                           LED0:= True;
                           exit;
                       else
                           LCD.Clear_Screen;
                           LCD.Put("Calculating HR..");
                       end if;
                   end if;
               else
                       -- Values remain the same
                       PrevPrev := PrevPrev;
                       Prev     := Prev;
                       Current  := Current;
                       Next     := Next;
                       NextNext := NextNext;

               end if;
        end loop;
end Pulse;
```

**Figure 7.2.34 – Pulse Detection Compiled**

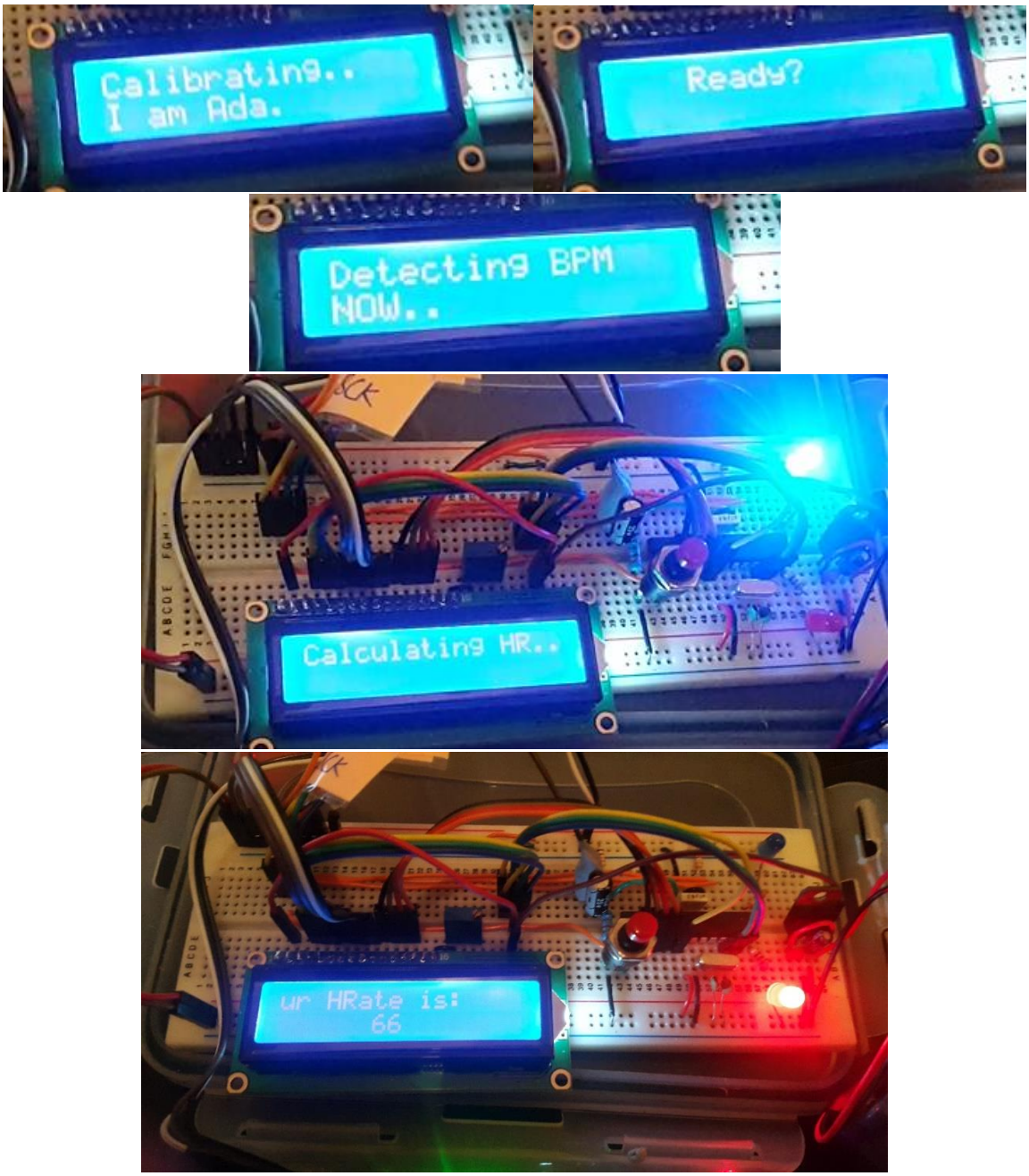



The Pulse Detection Algorithm was successfully employed and compiled, thus, the program is ready for testing.

**Cycle 6 – Testing: Pulse Detection Algorithm**

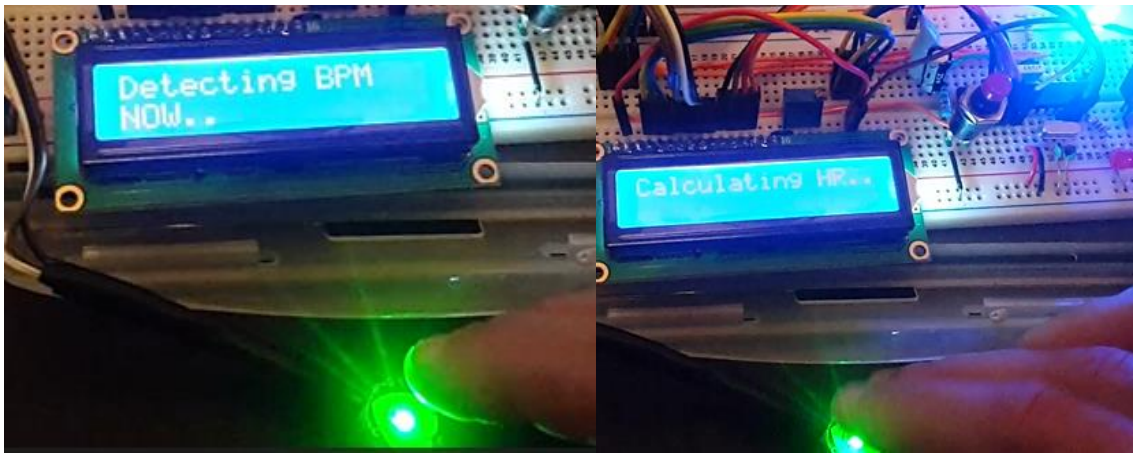
The Pulse Peak Detection Algorithm was programmed to the MCU and Tested – Figure 7.2.35.

**Figure 7.2.35 – System Running Pulse Detection Algorithm**

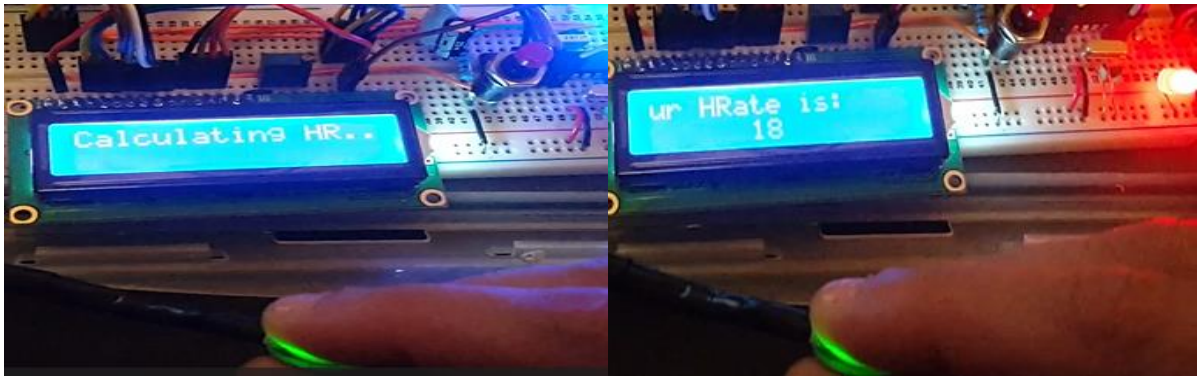



The system was executing the pulse detection algorithm. The concept of Thresh Peak Algorithm previously introduced operated effectively and the estimated 10 seconds using delays was tested on different cycles between 4000 and 4500. At 4500, the time frame was approximately 11 - 12 seconds, additionally, the time frame at 4000 was approximately 10 to 11 seconds. Thus, 4000 iterations were selected and BPM was successfully deliberated. Malfunction was only witnessed when sensor was not positioned on time. The program would start and end on that estimated 10 seconds regardless if pulses are being read – Figure 7.2.36.

**Figure 7.2.36 – Possible Malfunction**




Sensor not positioned on time.




Sensor positioned late and not all pulses were detected. Thus, the algorithm requires slight alterations. At this stage, alterations were made to the LCD Calibration procedure, providing the user with more instructions and time prior to performing BPM calculation. In addition, when all cycles have been completed, a sensor clip or strap can be mounted to improve its preparation/position. While the sensor was positioned on time, the system responded efficiently.

Cycle 6 – Evaluation

The Thresh Peak Algorithm operated competently, peaks were consistent. Furthermore, the design approach to the system has benefitted from using an incremental development. The system experienced a malfunction when purposely struggling to position sensor on time. Nevertheless, it operates profoundly otherwise. However, accuracy in timing events is paramount, especially when dealing with HRMs. Thus, the most concerning aspect at this stage is time. By initiating a timer that triggers accurate timing events, the system may improve substantially – Cycle 7 – Timer Setup.

Cycle 7 – Timer Setup

- Refined software design with improvements – Timer Setup.
- **Product:** System running a timer on CTC mode, tested using LEDs and adopted to Pulse Detection Algorithm.

The system operating on a timer enhances the system's quality and accuracy throughout execution and formulisation of results. Timers can be integrated on multiple modes through initialization of MCU registers. The idea of implementing a timer on CTC mode i.e. Clear Timer on Compare match, signifies that a 1 second delay can be initiated by using a formulated counter value that takes 1 second for the timer to reach.

The Timer Counter Register 1 (TCNT1) is an incrementing counter of 2 bytes and the Output Compare Register is a 2 Byte register where a formulated counter value is assigned. TCNT1 compares its counting values with the OCR and when a match is found, overflow occurs and an interrupt is set. Thus, for the timer to restart, that interrupt needs to be cleared. A PRC Card was designed for these tasks – Figure 7.2.37.

**Figure 7.2.37 – PRC Card: Timer**

| **PRC Card** |
|---|
| **Package:** Timer |
| **Responsibilities & Collaborations**<br>➢ Initialize Timer – Collaborates with MCU Directly<br>➢ Check Interrupt Flag – Collaborates with MCU and Controller<br>➢ Reset Flag – Collaborate with MCU |

In addition, the following setup is required to develop the Timer Package and initialize the MCU – Figure 7.2.38.

**Figure 7.2.38 – Timer Counter Register 1 B**

**TCCR1B Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit No. |
|---|---|---|---|---|---|---|---|---|
| ICNC1 | ICES1 | | WGM13 | WGM12 | CS12 | CS11 | CS10 | HEX |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0x0C |

Bit 7 – Not Important: Input Capture Noise Canceller

Bit 6 – Not Important: Input Capture Edge Select

Bit 3:4 – Wave Form Generation Mode: Combined with the WGM1[1:0] bits found in the TCCR1A Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes.

| Mode | WGM13 | WGM12 (CTC1)[1] | WGM11 (PWM11)[1] | WGM10 (PWM10)[1] | Timer/ Counter Mode of Operation | TOP | Update of OCR1x at |
|---|---|---|---|---|---|---|---|
| 3 | 0 | 0 | 1 | 1 | PWM, Phase Correct, 10-bit | 0x03FF | TOP |
| 4 | 0 | 1 | 0 | 0 | CTC | OCR1A | Immediate |

Thus, initiating CTC mode results in WGM12 being set; as illustrated above.

Bit 2:0 – Resemble the mode of operation regarding the Prescaler, also known as the three clock-select bits.

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | clkI/O/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |

The formula in Figure7.2.39 allows wise decisions to be made on what Prescaler should be used.

**Figure 7.2.39 – Timer Count Formula**

$$Timer\ Count = \frac{Required\ Delay}{Clock\ Time\ Period} - 1$$

The required delay is 1 second and the clock time-period is 16MHz/Prescaler – Figure 7.2.40.

**Figure 7.2.40 – Timer Count**

| | Freq/Pres | Req Delay/CTP -1 |
|---|---|---|
| Prescaler | Clock Time Period | Timer Count |
| 1 | 16MHz | **15999999** |
| 8 | 2MHz | **1999999** |
| 64 | 250KHz | **249999** |
| 256 | 62.5KHz | **62499** |
| 1024 | 15.625KHz | **15624** |

Prescaler 256 and 1024 are the only two that could count a 1 second delay using a 16bit register. Smaller Prescalers provide better resolution.

According to Timer Count table, the Prescaler that best suits the system for a 1 second delay formulated to 256 – **62499**. As a result, Timer Package is complete for pseudo code design.

**Pseudo Code – Timer Initialization Procedure**

```
Procedure Init
  CS12 set to 1
  WGM12 set to 1
  TCNT1 initialized to 0
  OCR1A initialized to Timer Counter 62499
end procedure Init
```

As explained previously, an interrupt is set upon overflow and to restart timer it requires being cleared. However, clearing the flag is done by writing a logical one to it.

**Pseudo Code – Timer Check Interrupt Function**

```
Function CounterCompare_InterruptSet
  return Interrupt_Flag not equal to False
end Function CounterCompare_InterruptSet
```

**Pseudo Code – Timer Reset Flag Procedure**

```
Procedure Reset_Flag
  Interrupt_Flag becomes True (reset by writing a logical 1)
end Procedure Reset_Flag
```

The Timer Setup is ready for implementation.

**Cycle 7 – Implementation: Timer Setup**

The implementation was adopted conforming with the above-stated Pseudo Codes – Figure 7.2.41.

**Figure 7.2.41 – Timer Setup Implementation**

```
with AVR;
use AVR;
with AVR.MCU;
use AVR.MCU;

--ATMEGA328P Timer1 Setup CTC Mode
package body Timer is

        CS12 : Boolean renames MCU.TCCR1B_Bits (MCU.CS12_Bit);
        WGM12 : Boolean renames MCU.TCCR1B_Bits (MCU.WGM12_Bit);
        Interrupt_Flag : Boolean renames MCU.TIFR1_Bits (MCU.OCF1A_Bit);
        TCNT1 : Unsigned_16 renames MCU.TCNT1;

        procedure Init is
        begin
                --Initializes Prescaler 256 and CTC Mode (TCCR1B (CS12 ---> 1) WGM12 ---> 1)
                CS12 := True;  WGM12 := True;

                --Initialize Counter
                TCNT1 := 0;

                --Initialize Compare Value: 16MHz/256 = 1/62500 = (1000msDelay/0.016ms) -1 = 62499
                MCU.OCR1A := 62499;
        end Init;

        function CounterCompare_InterruptSet return Boolean is
                --Check the TIFR Register for the OCF1A Flag
                --When a match is found,  the comparison Flag becomes active
                --So return true if flag is active
        begin
                return Interrupt_Flag /= False;         -- OCF1A is set
        end CounterCompare_InterruptSet;

        procedure ResetFlag is
        begin
                --write a logical one to reset i.e.flag becomes true (1)
                Interrupt_Flag := True;
        end resetflag;
end Timer;
```

Timers can be tedious regarding algorithm mistakes; thus, unit testing was performed to isolate issues completely. A main program was implemented and the Timer Setup implementation was tested through LEDs – Figure 7.2.42.

**Figure 7.2.42 – Timer Unit Test Program**

```
with AVR;
use AVR;

with AVR.MCU;
use AVR.MCU;

with Timer;
use Timer;

procedure Main is
        LED0 : Boolean renames MCU.PORTB_Bits(0);
        LED  : Boolean renames MCU.PORTB_Bits(1);
        Counter : Integer := 0;
begin
        Timer.Init;
        loop
                if (Timer.CounterCompare_InterruptSet) then

                        Counter := Counter +1;
                        if(Counter =10) then
                                LED := False;
                                LED0 := True;
                                delay 1.0;
                                Counter :=0;
                        end if;
                        Timer.ResetFlag;
                else
                        LED := True;
                        LED0 := False;
                end if;
        end loop;
end Main;
```

For testing purposes, the counter was originally tested with a counter value of 1, resulting in a 1 second delay. Upon success, a counter of 10 was instantiated to achieve a 10 second delay instead. The LEDs alternating every 1 second successively proves a working timer. The program was compiled and subsequently tested – Figure 7.2.43 and tested.

**Figure 7.2.43 – Timer Setup Compiled**

```
jordan@jordan-HP-ENVY-6-Notebook-PC:~/avr_workSpace/TimingTest$ make
avr-gnatmake -XMCU=atmega328p -p -Ptest.gpr -XAVRADA_MAIN=main
avr-gcc -c --RTS=avr5 -gnatec=/opt/avr_492_gnat/avr/lib/gnat/gnat.adc -gdwarf-2 -gnatwp -gnatwu -gnatn -gnatp -gnatVn -Os -gnatef -fverbose-asm
 -frename-registers -gnatdY -fdata-sections -ffunction-sections -mmcu=atmega328p -gnateDMCU=atmega328p -gnateDUART=usart0 -gnaty3abefhiklM130pr
n -I- -gnatA /home/jordan/avr_workSpace/TimingTest/timer.adb
/home/jordan/avr_workSpace/TimingTest/timer.adb:9:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:10:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:11:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:12:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:14:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:15:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:16:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:17:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:19:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:20:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:22:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:23:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:24:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:26:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:27:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:28:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:29:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:30:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:31:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:32:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:34:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:35:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:36:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:37:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:38:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.adb:38:13: (style) bad casing of "ResetFlag" declared at timer.ads:4
/home/jordan/avr_workSpace/TimingTest/timer.ads:2:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.ads:3:01: (style) horizontal tab not allowed
/home/jordan/avr_workSpace/TimingTest/timer.ads:4:01: (style) horizontal tab not allowed
avr-gnatbind --RTS=avr5 -freestanding -x /home/jordan/avr_workSpace/TimingTest/obj/main.ali
avr-gnatlink /home/jordan/avr_workSpace/TimingTest/obj/main.ali -Wl,--gc-sections -gdwarf-2 -Wl,--relax --GCC=avr-gcc -Os -mmcu=atmega328p --RT
S=avr5 -gnatp -fdata-sections -ffunction-sections -Wl,-Map=../main.map,--cref /opt/avr_492_gnat/avr/lib/gnat/avr_lib/atmega328p/lib/libavrada.a
 -Wl,-rpath,/opt/avr_492_gnat/lib/gcc/avr/4.9.2/avr5/adalib/ -o /home/jordan/avr_workSpace/TimingTest/main.elf
avr-objcopy -O ihex -R .eeprom main.elf main.hex
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
--change-section-lma .eeprom=0 -O ihex main.elf main.eep
avr-objcopy: --change-section-lma .eeprom=0x8000000000000000 never used
avr-objdump -h -S main.elf > main.lss
avr-nm -n main.elf > main.sym
avr-size --format=avr --mcu=atmega328p main.elf
AVR Memory Usage
----------------
Device: atmega328p

Program:    1080 bytes (3.3% Full)
(.text + .data + .bootloader)

Data:          8 bytes (0.4% Full)
(.data + .bss + .noinit)
```

Consequently, integration testing was implemented on the Controller by adopting the exact same formulisation used during unit testing. Proving further incremental development amongst cycles – Figure 3.2.44.

**Figure 7.2.44 – Timer Setup Integration Testing Procedure**

```
with ADC;
use ADC;

with LCD;
use LCD;

with TIMER;
use TIMER;

--Main Program makes used of ADC and LCD packages (Timer Package will be added if testing of its functionality is accomplished)
procedure Controller is
    LED0 : Boolean renames MCU.PORTB_Bits(0);
    LED  : Boolean renames MCU.PORTB_Bits(1);
-- Declare Values of interest globally
-- Initialize values of interest to a default value

    PrevPrev    : ADC.To_Volts := 0.0;
    Prev        : ADC.To_Volts := 0.0;
    Current     : ADC.To_Volts := 0.0;
    Next        : ADC.To_Volts := 0.0;
    NextNext    : ADC.To_Volts := 0.0;

    pulse   : Integer := 0;
    Counter : Integer := 0;
    HR : Integer := 0;
begin
    MCU.DDRB_Bits (1) := DD_Output;
    MCU.DDRB_Bits (0) := DD_Output;

    LED0 := True;
    --Initialize
    ADC.Initialize;
    LCD.Initialize;
    LCD.Clear_Screen;
    LCD.Calibrate_System_UI;
        loop
                -- Start Conversion
                ADC.Start_Conversion;
                 if ADC.Conv_Complete then
                     NextNext := ADC.Compute_Volts_From_ADC;
                     --threshold of 2.7 volts anything below is ignored
                     if (NextNext >= 2.7) then
                         -- Assign Values of Interest
                         PrevPrev := Prev;
                         Prev     := Current;
                         Current  := Next;
                         Next     := NextNext;
                         --if current is bigger than all values of interest then we have located a peak/pulse
                         if (Current > Prev and Current > PrevPrev and Current > Next and Current > NextNext) then
                             --pulse detected so we increment number of pulses
                             pulse := pulse + 1;
                             if (pulse = 1) then
                                 Timer.Init;
                                 LED := True;
                                 LED0 := False;
                             end if;
                         end if;
                         LCD.Put("Detecting BPM..");
                         LCD.Clear_Screen;
                         --Interrupt is set when the TimerCounter1 Hits 62499 (which is the value set on the output compare register)
                         --When Flag is reset by writing a logical one to it, the timerCounter1 resets automatically and restarts counting.
                         --Every interrupt is triggered after 1 second, thus, we wait for the interrupt to be triggered 10 times.
                         if (Timer.CounterCompare_InterruptSet) then
                             Counter := Counter +1;
                             if(Counter =10) then
                                 -- calculate BPM and display it
                                 HR := pulse * 6;
                                 LCD.Put("BPM:");
                                 LCD.GotoXY(7,1); --7,2
                                 LCD.Display_int(HR);
                                 delay 2.0;
                                 LED := False;
                                 LED0 := True;
                                 exit;
                             else
                                 LCD.Clear_Screen;
                                 LCD.Put("...");
                             end if;
                             Timer.ResetFlag;
                         end if;
                     else
                         -- Values remain the same
                         PrevPrev := PrevPrev;
                         Prev     := Prev;
                         Current  := Current;
                         Next     := Next;
                         NextNext := NextNext;
                     end if;
                 end if;
        end loop;
end Controller;
```

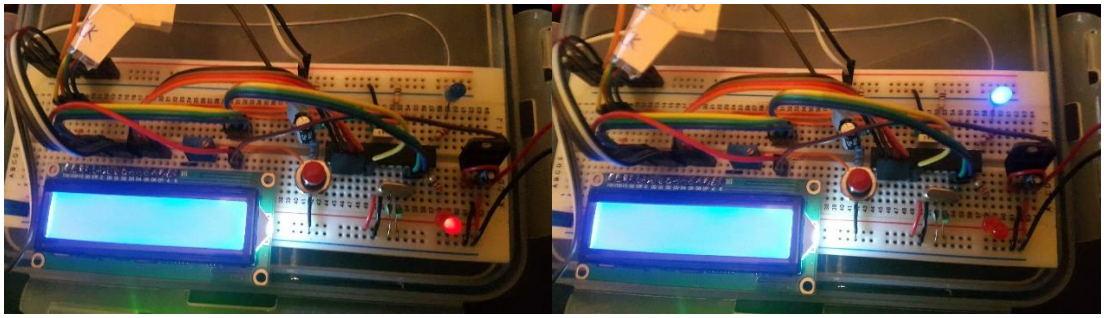
Timer begins after the 1st pulse has been found.

Checks for an Interrupt Flag and Calculates BPM after 10 seconds.

Alterations were made to the Controller as integration tests were being employed. Changes made resolved the malfunction previously encountered, allowing users to place the sensor when ready. The system now awaits for the first pulse to be detected prior to initiating the timer, this avoids a running timer if sensor has not yet been properly positioned.

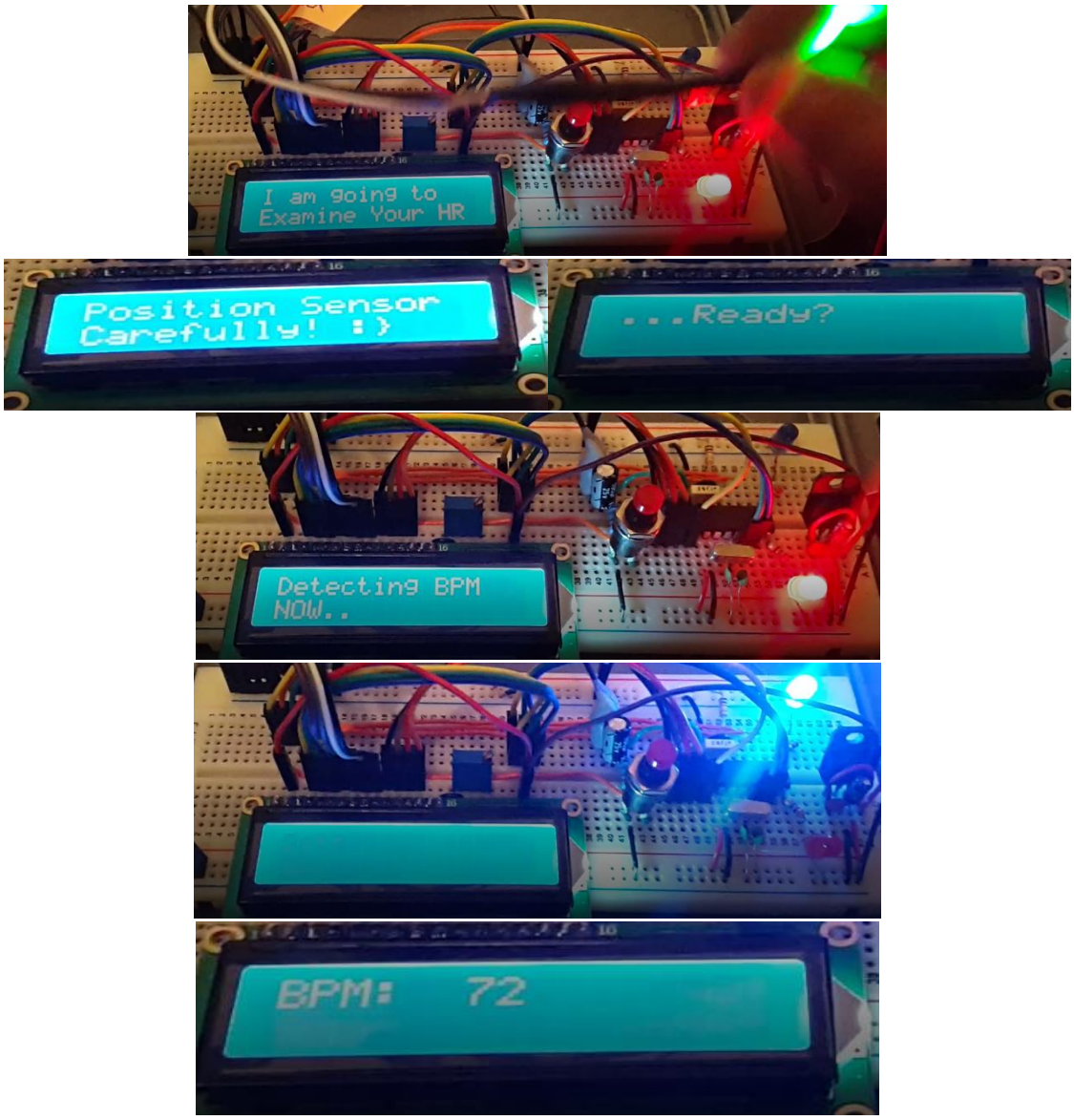
**Cycle 7 – Unit Testing: Timer Setup**

**Figure 7.2.45 – System Running Timer Setup Unit Tests**



**Cycle 7 – Integration Testing: Incremental Development**

**Figure 7.2.46 – System Running Timer Setup Integration Tests**



Cycle 7 – Evaluation

Employing a timing event on the system has incremented its overall efficiency and accuracy. The system proved being user friendly and extremely accurate for its 10 second time frame. Overall, Timer Counter 1(Timer1) on the MCU was used in CTC mode. Timer1, a 16bit register counts to the formulated value of 62499, causing a 1 second delay. A counter was employed to count 10 overflows before calculating the BPM. As a result, the system became extremely accurate. The system proved to operate efficiently and demonstrated a complete system. However, to fulfil all system requirements, a Safety procedure must be included to the overall software design to provide advice regarding the state of each BPM calculation. In addition, the LED should be initialized from a separate package to isolate the controller from having direct access to the MCU.

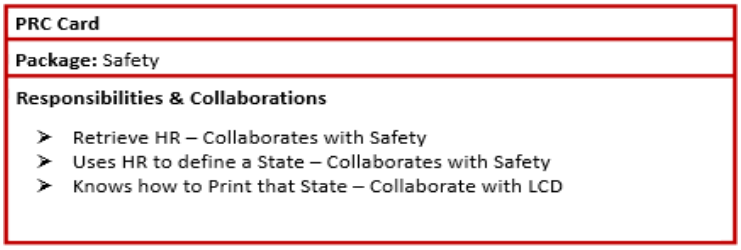
Cycle 8 – Safety & LED

- Refined software design with improvements – Safety & LED
- **Product:** System running successfully and tested by displaying useful advice on completion. In addition, an LED Package will be included to avoid the controller from accessing the MCU directly.

**Cycle 8 – Design: Safety**

A PRC Card was developed for this procedure to define the main responsibilities associated with the Safety Package – Figure 7.2.47.

**Figure 7.2.47 – PRC Card: Safety**




The system is contained with all packages required to calculate and provide a BPM result effectively. However, the system can be enhanced further by providing users with crucial data regarding their physical state with reference to their BPM. In theory, BPM varies immensely between users of distinct ages and gender. The aim of the system is to target users of all ages, gender and background. Thus, an average range was calculated from all averages previously illustrated in Research – HRM functionalities – Figure 3.2.1.D1/D2. Results are displayed in Figure 7.2.48.

**Figure 7.2.48 – Resting Heart Rate (RHR) Averages**

| | Men Average | | Women Average | | Total Average | |
|---|---|---|---|---|---|---|
| State | From | To | From | To | From | To |
| Athlete | 49.83333 | 55.5 | 54 | 59.33333 | 52 | 58 |
| Excellent | 56.5 | 61.5 | 60.33333 | 64.33333 | 58.5 | 63 |
| Good | 62.5 | 65.83333333 | 65.33333 | 68.5 | 64 | 67 |
| Above Average | 66.83333 | 70 | 69.5 | 72.66667 | 68.5 | 71.5 |
| Average | 71 | 74.33333333 | 73.66667 | 77 | 72.5 | 75.5 |
| Below Average | 75.33333 | 81.16666667 | 78 | 83.33333 | 76.5 | 82 |
| Poor | 82.16667 | + | 84.16667 | + | 83 | + |

A total average was formulated and states can now be initiated. However, there are too many states that define a good RHR and may pose confusion. Furthermore, RHR of above 83 states poor, even-though a RHR of above 120 may require medical attention. Hence, states were slightly altered to suit a wider range and avoid confusion – Figure 7.2.49.

**Figure 7.2.49 - States**

| State | Condition | RHR |
|---|---|---|
| Red | At Risk | 120+ |
| Amber | Risky | 100+ |
| Poor | No Risk | 83+ |
| Green | Excellent | 52+ |
| Poor | No Risk | 40+ |
| Red | At Risk | Else |

The formulated alterations go by colour,

- Red; representing seek medical attention.
- Amber; representing seek medical advice.
- Green; representing excellent RHR and no medical attention or advice is necessary.
- Poor literally represents a poor RHR, though, not at risk.

In conclusion, excellent RHR has a large swing, there is no need to elaborate the exact state if the user contains a good RHR.

The software design will use an enumerator type representing each state and the Controller will feed the Safety package with BPM results. Additionally, regarding the BPM representation, the Safety package should be intelligent enough to act upon it and print the status efficiently.

**Safety Package Requirements**

**Pseudo Code – Identify HR**

```
procedure HR_is (HR of type Integer)
    enumerator type status initialized to worst case
    status becomes HR
    status is fed to function Check HR and stored in status
    print status
end procedure HR_is
```

**Pseudo Code – Perform Check**

```
function HRCheck takes a HR of Integer, returns medical state enumerator type
    if HR is greater or equal to 120, return red alert high
    else if HR is greater or equal to 100, return amber alert high
    else if HR is greater or equal to 83, return poor high
    else if HR is greater or equal to 52, return excellent
    else if HR is greater or equal to 40, return poor low
    else if HR is less than 40, return red alert low
end function HRCheck
```

**Pseudo Code – Print HR Status**

```
procedure print HR status (status of type enumerator)
    if status, print RED: At RISKK!
    else if  status, print Amber: Risky!
    else if  status, print Poor: No Risk
    else if  status, print RHR Excellent!
    else if  status, print Poor: No Risk
    else if  status, print RED: At RISKK!
end procedure print HR status
```

The safety package is ready for implementation.

Furthermore, to isolate the Controller from accessing the MCU, an LED package was designed and implemented using the LED code from the Controller.

### Cycle 8 – Implementation: Safety Package

The pseudo code was implemented. In addition, a specification was included to the package body as one of Ada's standard or safety features and the LED package was also tackled.

**Figure 7.2.50 – Safety Package (Body)**

```
with LCD;
use LCD;

package body Safety is
        procedure HR_is (HR : in Integer) is
                status : Medical_State := Red_Alert_Low;
        begin
                status := HRCheck (HR);
                PrintHRStatus(status);
        end HR_is;

        function HRCheck (HR : Integer) return Medical_State is
        begin
                if (HR >= 120) then
                        return Red_Alert_High;
                elsif (HR >=100) then
                        return Amber_Alert_High;
                elsif (HR >= 83) then
                        return Poor_High;
                elsif (HR >= 52) then
                        return Excellent;
                elsif (HR >= 40) then
                        return Poor_Low;
                elsif (HR < 40) then
                        return Red_Alert_Low;
                end if;
        end HRCheck;

        --Knows how to print its result!
        procedure PrintHRStatus (status : in Medical_State) is
        begin
                if (status = Red_Alert_High) then
                        LCD.GotoXY(1,2);
                        LCD.Put("RED: At RISKK!");
                elsif (status = Amber_Alert_High) then
                        LCD.GotoXY(1,2);
                        LCD.Put("Amber: Risky!");
                elsif (status = Poor_High) then
                        LCD.GotoXY(1,2);
                        LCD.Put("Poor: No Risk");
                elsif (status = Excellent) then
                        LCD.GotoXY(1,2);
                        LCD.Put("RHR Excellent");
                elsif (status = Poor_Low) then
                        LCD.GotoXY(1,2);
                        LCD.Put("Poor: No Risk");
                elsif (status = Red_Alert_Low) then
                        LCD.GotoXY(1,2);
                        LCD.Put("RED: At RISKK!");
                end if;
        end PrintHRStatus;
end Safety;
```

**Figure 7.2.51 – Safety Package (Specification)**

```
with AVR.Strings;
use AVR.Strings;

package Safety is

        type Medical_State is ( Red_Alert_High,
                                Amber_Alert_High,
                                Poor_High,
                                Excellent,
                                Poor_Low,
                                Red_Alert_Low);

        procedure HR_is (HR : in Integer);
        function HRCheck (HR : Integer) return Medical_State;
        procedure PrintHRStatus (status : in Medical_State);
```

**Figure 7.2.52 – LED Package (Body)**

```
with AVR.MCU;
use AVR;

package body LED is

        LED  : Boolean renames MCU.PORTB_Bits(1);
        LED0 : Boolean renames MCU.PORTB_Bits(0);

        procedure Setup_LED_Ports is
        begin

                MCU.DDRB_Bits (1) := DD_Output;
                MCU.DDRB_Bits (0) := DD_Output;

        end Setup_LED_Ports;

        procedure Turn_Red_On (Item : in Boolean) is
        begin
                LED0 := Item;
        end Turn_Red_On;

        procedure Turn_Blue_On (Item : in Boolean) is
        begin
                LED := Item;
        end Turn_Blue_On;

end LED;
```

**Figure 7.2.53 – LED Package (Specification)**

```
with interfaces;
use interfaces;

package LED is

procedure Setup_LED_Ports;
procedure Turn_Red_On (Item : in Boolean);
procedure Turn_Blue_On (Item : in Boolean);

end LED;
```

Subsequently, the packages were all applied to the Controller – Figure 7.2.54.

**Figure 7.2.54 – Controller Complete**

```
with ADC;
use ADC;

with LCD;
use LCD;

with Timer;
use Timer;

with Safety;
use Safety;

with LED;
use LED;

--Main Program makes used of ADC and LCD packages (Timer Package will be added if testing of its functionality is accomplished)
procedure Controller is
-- Declare Values of interest globally
-- Initialize values of interest to a default value
    PrevPrev    : ADC.To_Volts := 0.0;
    Prev        : ADC.To_Volts := 0.0;
    Current     : ADC.To_Volts := 0.0;
    Next        : ADC.To_Volts := 0.0;
    NextNext    : ADC.To_Volts := 0.0;
--Declare and Initialize Counter, Pulse and HR
    pulse   : Integer := 0;
    Counter : Integer := 0;
    HR : Integer := 0;
begin
    Setup_LED_Ports;
    Turn_Red_On(True);
    --Initialize
    ADC.Initialize;
    LCD.Initialize;
    LCD.Clear_Screen;
    LCD.Calibrate_System_UI;
    loop
        -- Start Conversion
       ADC.Start_Conversion;
       -- We test if conversion is completed, if True (result is stored into the array)
       -- We increase the counter every time a conversion is complete and stored
       if ADC.Conv_Complete then
           NextNext := ADC.Compute_Volts_From_ADC;
           --threshold of 2.7 volts anything below is ignored
           if (NextNext >= 2.7) then
               -- Assign Values of Interest
               PrevPrev := Prev;
               Prev     := Current;
               Current  := Next;
               Next     := NextNext;
               --if current is bigger than all values of interest then we have located a peak/pulse
               if (Current > Prev and Current > PrevPrev and Current > Next and Current > NextNext) then
                   --pulse detected so we increment number of pulses
                   pulse := pulse + 1;
                   if (pulse = 1) then
                       Timer.Init;
                       Turn_Blue_On(True);
                       Turn_Red_On(False);
                   end if;
               end if;
               LCD.Put("Detecting BPM...");
               --Interrupt is set when the TimerCounter1 Hits 62499 (which is the value set on the output compare register)
               --When Flag is reset by writing a logical one to it, the timerCounter1 resets automatically and restarts counting
               --Every interrupt is triggered after 1 second, thus, we wait for the interrupt to be triggered 10 times.
               if (Timer.CounterCompare_InterruptSet) then
                   LCD.Clear_Screen;
                   Counter := Counter +1;
                   if(Counter =10) then
                       -- calculate BPM and display it
                       HR := pulse * 6;
                       LCD.Put("BPM:");
                       LCD.GotoXY(7,1);
                       LCD.Display_Int(HR);
                       Turn_Blue_On(False);
                       Turn_Red_On(True);
                       Safety.HR_is(HR);
                       exit;
                   else
                       LCD.Clear_Screen;
                   end if;
                   Timer.ResetFlag;
               end if;
               LCD.Clear_Screen;
           else
               -- Values remain the same
               PrevPrev := PrevPrev;
               Prev     := Prev;
               Current  := Current;
               Next     := Next;
               NextNext := NextNext;
           end if;
       end if;
    end loop;
end Controller;
```

No Access to MCU

LED package adopted.

The timer is initialised when the first peak is detected. This allows the system to wait for a user.

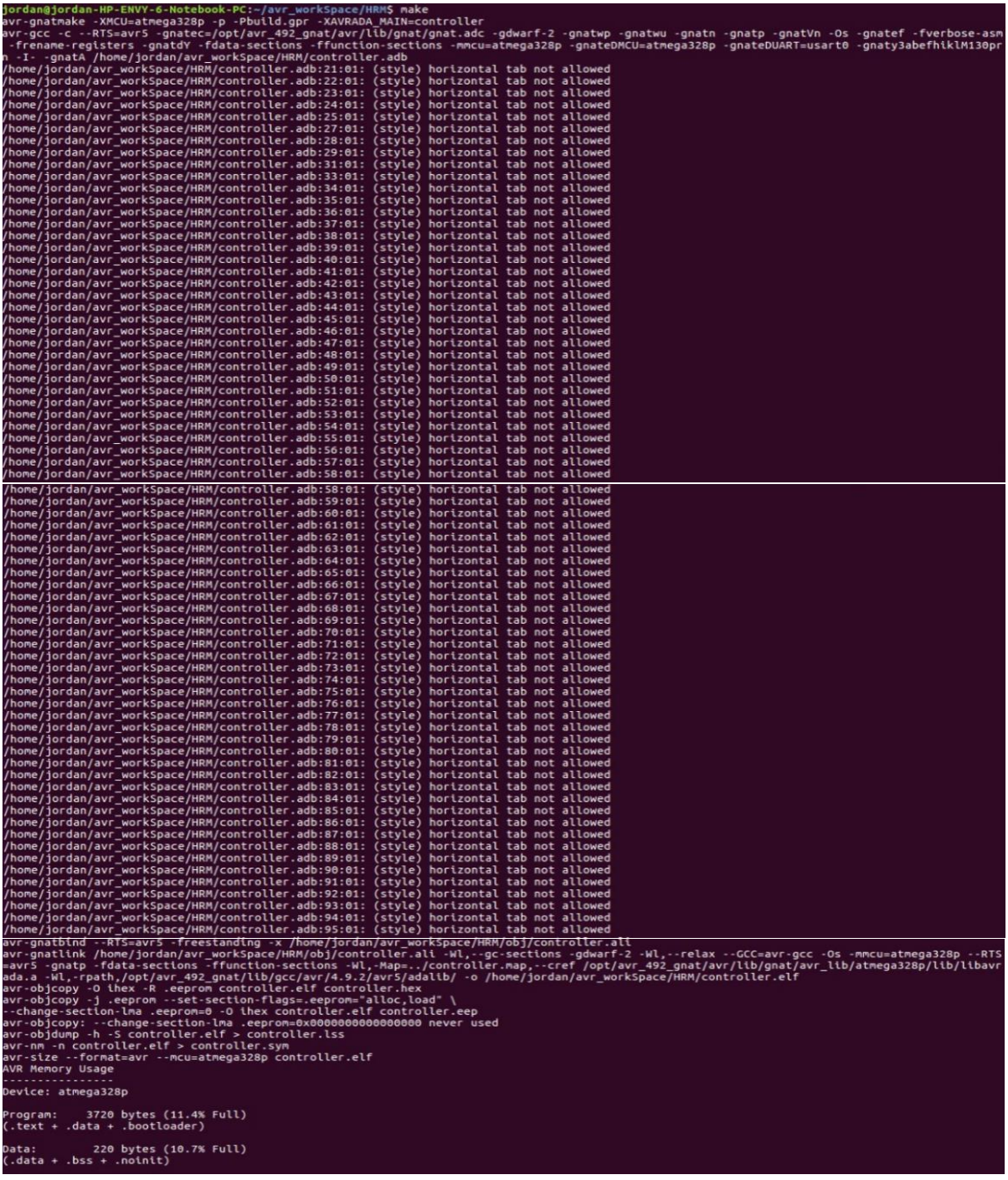
LED package adopted.

Safety & LED package adopted.

Timer Checks Interrupt Flag within the Threshold Acceptance. If a Peak is not detected, the flag is not cleared and timer pauses. Thus, the system waits for the user to reposition sensor, detect the peak again and resume with its detection. As a result, it demonstrates consistency and reliability.

The Controller has no direct access to the MCU, it uses the packages containing with clauses to satisfy services. They compiled successfully – Figure 7.2.55, and is now ready for further integration testing. Moreover, the Calibrate_System_UI procedure in the LCD package was altered to provide a good user experience – Figure 7.2.56.

**Figure 7.2.55 – Safety and LED Package Compiled via Controller**




All warnings have been eliminated to reassure the system's safety. There is nothing that can be done about the tabs as it is a layout (style) rule. Instead of using tabs, use of spaces was applied for testing purposes. This action resolved "tab not allowed" by substituting it with "bad indentation". Fortunately, they do not represent warnings.

**Figure 7.2.56– Calibrate_System_UI Procedure**

```
procedure Calibrate_System_UI is
begin
        LCD.Put("Greetings :)");
        LCD.GotoXY(1,2);
        LCD.Put("I am Ada.");
        delay 2.0;
        LCD.Clear_Screen;
        LCD.Put("I am going to");
        LCD.GotoXY(1,2);
        LCD.Put("Examine Your HR");
        delay 2.0;
        LCD.Clear_Screen;
        LCD.Put("Position Sensor");
        LCD.GotoXY(1,2);
        LCD.Put("Carefully! :)");
        delay 3.0;
        LCD.Clear_Screen;
        LCD.Put(".");
        delay 2.0;
        LCD.GotoXY(2,1);
        LCD.Put(".");
        delay 2.0;
        LCD.GotoXY(3,1);
        LCD.Put(".");
        delay 2.0;
        LCD.GotoXY(4,1);
        LCD.Put("Ready?");
        delay 1.0;
        LCD.Clear_Screen;
        LCD.Put("Detecting BPM");
        LCD.GotoXY(1,2);
        LCD.Put("NOW..");
        delay 1.0;
end Calibrate_System_UI;
```

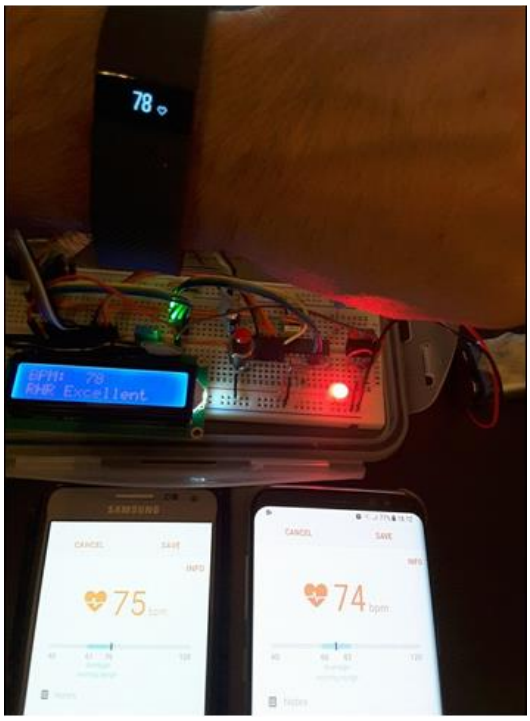
Cycle 8 and the system is now complete; ready for testing.

**Cycle 8 – Integration Testing: Safety & LED**

**Figure 7.2.57 – System running all packages**


Testing proved that the system was effective and consistent. Therefore, several devices with HR monitoring capabilities were obtained, tested and compared – Figure 7.2.58.

**Figure – 7.2.58 – System Accuracy**




The system proved to operate efficiently and accurately in all tests.

Devices represent the listed brands:

- FitBit – Charge HR (wrist watch)
- Samsung Galaxy Alpha (bottom left)
- Samsung Galaxy S8 – Latest Model
- HRM Prototype

All devices formulated to similar results, the Charge HR and the HRM Prototype were the same. This proves that the Thresh Peak Algorithm introduced in this paper is effective and profound. The system has been tested in a dozen of occasions and has not failed to detect the peaks.

## Cycle 8 – Evaluation

Cycle 8 was considered as the final cycle regarding system functionality. Used to tie up loose ends and finally bring the system to completion. The Safety package was designed with regards to crucial research previously discussed and the LED package was implemented to limit the *controller*'s intelligence. A procedure was manipulated at the end of the implementation to greet the user and provide reliable instructions not previously addressed. The system was compared with other systems as a result of its performance and proved to be as accurate and efficient.

Cycle 9 – SPARK Tools Integration

- Verified software design using SPARK Tools
- **Product:** System fully verified and validated.

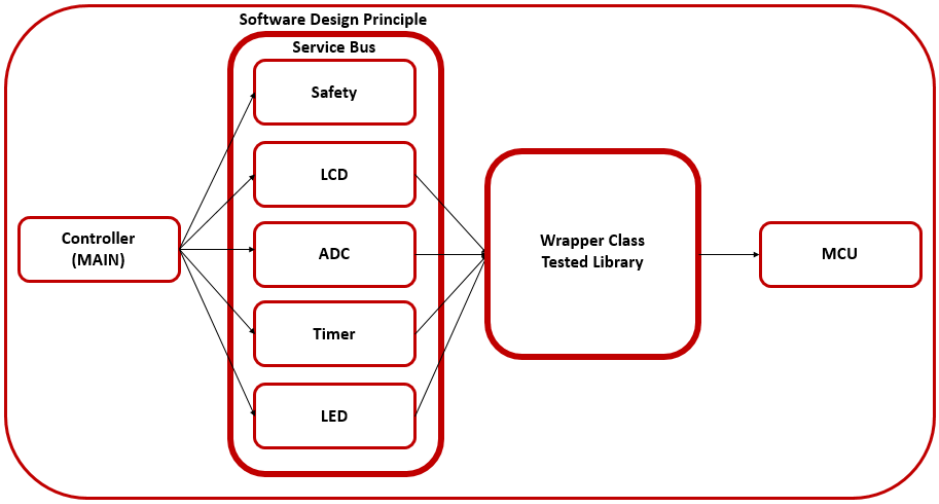
Integrating SPARK Tools is somewhat not compulsory, though, beneficial. Not one computer scientist or system's engineer have yet accomplished this task and thus, research is somewhat absent. This means that the success of this objective classifies as a unique and innovative solution towards formal verification and validation of 8-Bit Microcontrollers using SPARK Tools and Examiner.

**Cycle 9 – Design: SPARK Tools Integration**

The approach towards the completion of such task requires a new infrastructure towards the already existing one. Moreover, the main priority revolves around the way packages have direct access to the MCU.

The idea to accomplish this task is by creating a library or wrapper class that includes (setters and getters) procedures with direct access to the MCU registers and make the above-mentioned packages access the MCU via this library. Finally, using SPARK Tools to thoroughly test SPARK contracts included on the Library's specification file **only** – Figure 7.2.58.

**Figure 7.2.58 – SPARK Integration Library**




SPARK can be set to focus on checking specification file I/O contracts and disregard implementation details by setting SPARK Mode (On/Off). This is a benefit towards the project as AVR-Ada uses libraries not recognised by SPARK or the original Ada Compiler. Thus, compiling the implementation file with SPARK_Mode Off should resolve issues associated to those unknown libraries. Consequently, the specification file will have SPARK_Mode On – Figure 7.2.59.

**Figure 7.2.59 – SPARK Code Commented**

```
--#pragma SPAK_MODE (off);
```

Distinctive comments are required (--#) as two separate compilers will have to test the code; one that does not support SPARK and another that does not support AVR-ADA. Thus, AVR-ADA will have no issues in compiling the software when comments are present. An execution file will be implemented to make a copy of the new library, remove all distinctive comments and perform testing using SPARK Tools.

### Cycle 9 – Implementation: Wrapper Class

The Wrapper Class was implemented using standard procedures previously witnessed – Figure 7.2.60. This implementation was designed for testing purposes.

**Figure 7.2.60 – Wrapper Class Standard Procedures**

```
pragma SPARK_Mode (Off);

package body Wrapperclass
is

   procedure Init_Standard_Input   is
   begin
        SPARK.Text_IO.Init_Standard_Input;
   end Init_Standard_Input;

   procedure Init_Standard_Output   is
   begin
      SPARK.Text_IO.Init_Standard_Output;
   end Init_Standard_Output;

-->Initializes ADC
   procedure Set_Analog_As_Input (Item : in Unsigned_8) is
   begin
      --Code copied from ADC Package; uses avr Ada
      MCU.DDRC_Bits := (others => Item);
   end Set_Analog_As_Input;

-->start getting ADC result
   procedure get_Low_Byte (Item : out Boolean) is
   begin
      -- item becomes low byte
      Item := MCU.ADCL;
   end get_Low_Byte;

end Wrapperclass;
```

The code used to access the MCU are direct copies from the ADC Package. In addition, SPARK contracts were implemented within the specification – Figure 7.2.61.

**Figure 7.2.61 – Wrapper Class Specification**

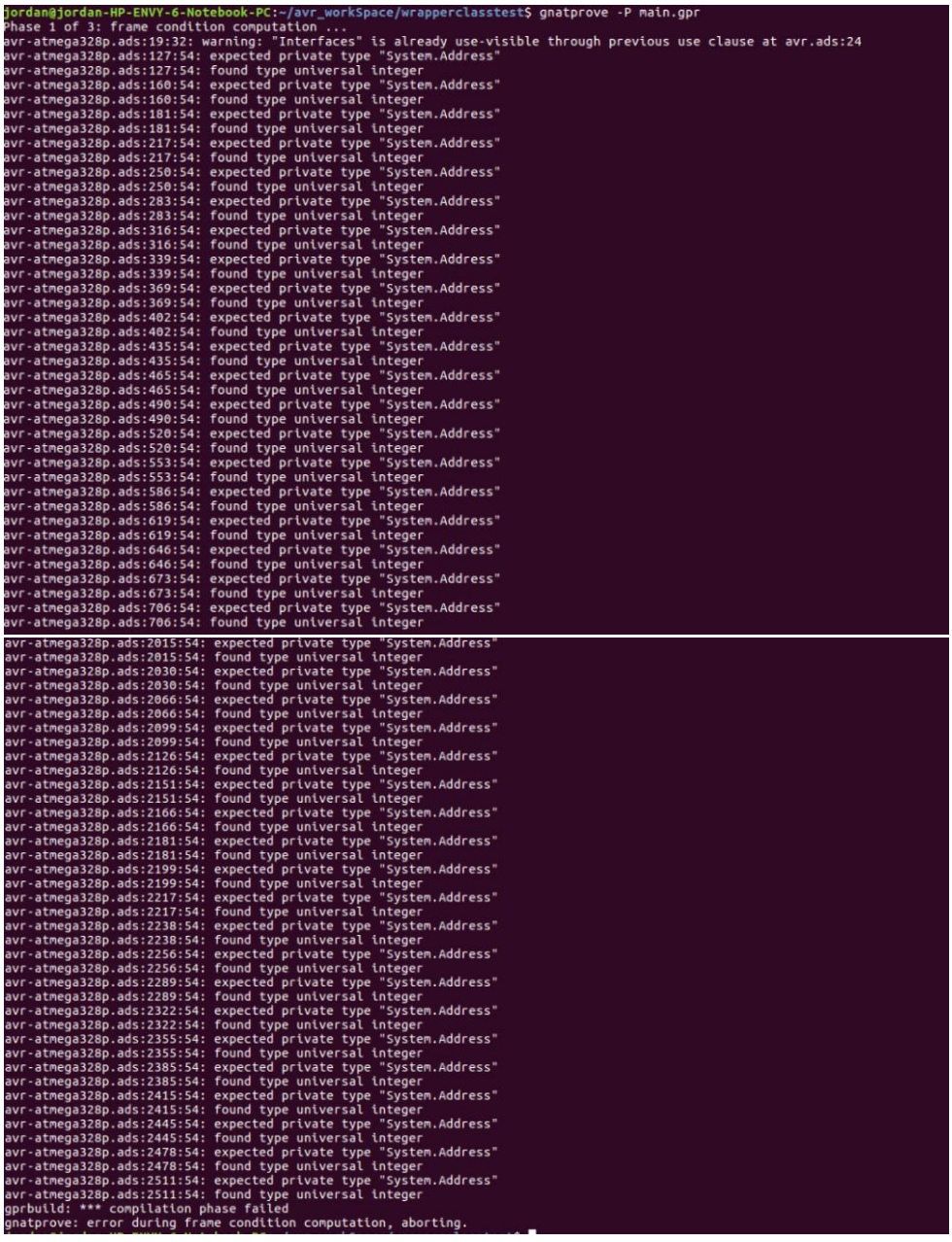
```
pragma SPARK_Mode (On);

package Wrapperclass
is
        -- this procedure initialises standard input
        procedure Init_Standard_Input
     with Global => (Output => Standard_Input),
          Depends => (Standard_Input  => null);

   -- this procedure initialises standard output
        procedure Init_Standard_Output
     with Global => (Output => Standard_Output),
          Depends => (Standard_Output  => null);

-->Init ADC Registers

        procedure Set_Analog_As_Input (Item : in Unsigned_8) --;--@
    with Global => (In_Out => Standard_Output),
    Depends=> (Standard_Output => (Item, Standard_Output));

-->Start getting ADC Result
        procedure get_Low_Byte (Item : out Boolean) --;--@
    with Global => (In_Out => (Standard_Input)),
    Depends=> (Standard_Input => Standard_Input, Item =>Standard_Input);

 end Wrapperclass;
```

SPARK contracts are those commands below each procedure, these two packages have been designed and altered with reference to Anton G Setzer's Wrapper Class, available on his personal webpage. The complete Wrapper Class/Library that would be used if the above tests are successful can be found in Appendix 8A:B.

**Cycle 9 – Unit Testing: Wrapper Class**

Testing the Wrapper Class is done through command line, using the following command: ***gnatprove -P file.gpr***. The GPR file has reference to the wrapper class package and will check the specification file's SPARK contracts using the Three Levels of Analysis followed by the SPARK Theorem Prover; GNATProve – Figure 7.2.62.

**Figure 7.2.62 – Wrapper Class Test**




Subsequently, various tests were performed and certain errors were resolved. However, due to lack of documentation the project manager was not able to hide AVR-ADA code from SPARK-Ada. In theory, SPARK_Mode Off should make GNATProve ignore subsequent code. Instead it performs checks on the complete implementation and detects unknown code and libraries. Unfortunately, SPARK Tools and Examiner could not be integrated into the above-mentioned infrastructure.

Cycle 9 – Evaluation

The infrastructure design to accomplish this objective was profound. Unfortunately, after having constructed the complete library and adjusting the code to suit all packages, an unsuspected technical error rose during unit testing. Had the system been accomplished, loss of efficiency would have been injected to the system. However, a system fully verified using SPARK-Ada would have led to an exceptional accomplishment and complete delivery. Nevertheless, the system has reached an increasingly satisfying and rewarding level with reference to the risk analysis assessment diagram – Green Zone.

CHAPTER 8

# EVALUATION

The project led from an understanding of the fundamental concepts imposed by Ada, SPARK and other software programming languages used pervasively in micro systems. Furthermore, distributing additional knowledge on Linux Ubuntu OS, MCUs, Project Management and Software Development Models. The selection process entitled to project models established its vitality, producing a fixed and settled outline towards the development process. In addition, the software model brought forward the accountability imposed by flexibility and scalability.

In essence, knowing the fact that producing software can go horribly wrong and possibly become a coliseum of unstructured data and undelivered results made prioritising and time management a major concern.

The realm encompassing project deliverables stands large on completion and as units during its development process. Thus, considering a development model that follows an incremental development process provided a reliable contrast towards real-world project facts. As a result, the iterative and incremental development model brought a soothing flow on each consecutive phase and cycle throughout the entire development process. In addition, employment of Unit and Integration Testing was made easy and fundamental.

The project's foundation representing the system prototype was constructed using research, past knowledge and experience; attained from a graduate's degree and Proteus professional; used to develop a visual representation or blueprint of the circuit design. As a result, the system was successfully developed and thoroughly tested.

Furthermore, the foundation representing the framework was developed with direct reference to well-structured documentations found online, assistance provided by the supervisor, Linux Fundamentals Booklet and the official avr-ada mailing list. As a result, the framework's infrastructure was tested in two separate operating systems. Additionally, a well-structured and informative tutorial was formulated to aid interested users and expand Ada's limitations and documentations.

Packages that made the project complete were designed in conformance to the ATMega328P and HD44780 Datasheet, RDP Algorithm and Threshold Theorem; which together brought Thresh Peak Algorithm to light and basic knowledge attained from self-motivation, as well as persistence towards programming practice. Moreover, the development of visual aids and infrastructures induced visions that formulated to Software Design aims, objectives and PRC Cards.

The integration of SPARK tools was unsuccessful due to the presence of unforeseen circumstances, resulting from SPARK-Ada's refusal to ignore or uncheck AVR-Ada code. Nevertheless, the system was verified and validated to the best of the developer's capability.

CHAPTER 9

# Conclusion

Consequently, the project was accomplished, providing an inclined learning curve establishing the foundations and progression regarding the author of this thesis' distributed background; Computer Systems and Electronics and Computer Science. The applicated knowledge of past and consecutive learning was of paramount importance throughout the course of this project.

The project was built in a heterogeneous environment, aimed to support AVR-8 Bit Microcontrollers, Ada programs, HRMs and users withholding similar interests.

When following this documentation, users with no experience should be able to undertake or implement all foundations established in the systems infrastructure, such as: integrating AVR-Ada programs into 8 Bit MCUs; Linux OS command line usage; research and project management, based on real-life contexts; risk analyses and risk assessments; time management; circuit and software design techniques; unit and integration testing methods; programming MCUs using Linux command line/AVRDude programmer; perform incremental developments towards large critical projects or systems and learn how a peak detection algorithm can be implemented to a HRM using the Thresh Peak Algorithm introduced by the author.

In conclusion, the system was directed via an incremental development where all cycles and phases undertook verification and validation testing performed using unit and integration testing methods. As a result, the system proved to operate exceptionally well, and for that reason it was compared with three other commercialised systems with HR monitoring capabilities, giving surprisingly accurate results.

Moreover, the HRM prototype was completed with the idea of enhancing its functional performance in the future via the adoption of additional cycles, and was delivered on time with the complete functionality addressed. As a result, negligence imposed by many software developer's experience or software crisis victims was avoided.

Finally, the thesis deliberates an effective design infrastructure towards the inclusion of SPARK Tools for verification and validation purposes. Though, the system was unable to implement SPARK Tools resulting from poor documentation, the design concept was reasonable and could possibly be adapted by an experienced Ada programmer.

CHAPTER 10

# REFLECTION

Undertaking this project was breath taking and the greatest accomplishment ever encountered. In this project, the most considerable learning aspect was to never underestimate your abilities, always be positive and prioritise large tasks into well-structured smaller elements. In addition, make decisions within reason.

My aim was to undertake a project that tested my complete abilities and that reflected my complete background. Fortunately, it was the perfect choice after pursuing a thorough discussion with my supervisor. He was the one who motivated me towards the idea of undertaking this project. In my mind I owe him much, as I would have never thought that testing both my past learning and experience, in conjunction with everything learnt in Computer Science recently was even possible.

Facing the project at first made me negative and terrified, always thinking "what if I can't make it". However, subsequent to several informal discussions with my supervisor I accepted the challenge and begun my journey.

Today, this journey has made me grow in both, confidence and as a person. As a result, I will never underestimate my abilities and will always accept and be positive regardless the task.

# REFERENCES


AB, N. M., 2002. *The electrocardiogram – looking at the heart of electricity.* [Online]
Available at: https://www.nobelprize.org/educational/medicine/ecg/ecg-readmore.html

AdaIC, n.d. *Boeing Flies on 99% Ada.* [Online]
Available at: http://www.aonix.com/content/products/objectada/success_stories/boe-777.html

Altran, A. a., 2017. *7.10. Examples in the Toolset Distribution.* [Online]
Available at: http://docs.adacore.com/spark2014-docs/html/ug/en/source/examples_in_the_toolset_distribution.html

Amey, P., 2002. Software Engineering Technology. *Correctness by Construction: Better Can Also Be Cheaper*.

Anderson, L. O., 1994. *Program Analysis and Specialization for the C Programming Language,* Denmark: http://www-ti.informatik.uni-tuebingen.de/~behrend/PaperSeminar/Program%20Analysis%20and%20SpecializationPhD.pdf.

Beeye, 2017. *Project risk analysis.* [Online]
Available at: https://www.mybeeye.com/en/management-tools/risk-analysis?utm_campaign=Outils%20de%20gestion&utm_source=ppc

Buhagiar, J. L. M., 2015. *Medical Devices ,* Swansea: Swansea University.

Capers, J., 2000. In: *Software Assessments, Benchmarks, and Best Practices.* Boston, United States : Pearson Education (US).

Carlisle, B. F. a. M., 2014. *Provably Secure DNS: A Case Study in Reliable Software.* Colorado, s.n., p. 9.

Clearinghouse, A. I., 2009 - 2017. *Case studies.* [Online]
Available at: http://www.adaic.org/advantages/case-studies/

Cobbaut, P., 2015-05-24. *Linux Fundamentals.* Linux Fun ed. s.l.:Free Software Foundatio.

Corporation, A., 2017. *ATMEGA328P Datasheet, PDF.* [Online]
Available at: http://www.alldatasheet.com/view.jsp?Searchword=ATMEGA328P

Corporation, L. M., 2014. *First C-130J Super Hercules 'Shimshon' Arrives In Israel.* [Online]
Available at: http://www.lockheedmartin.com/us/news/press-releases/2014/april/o409hq-1st-israel-c130j.html

David Cooper, J. B., 2008. *The Tokeneer System.* [Online]
Available at: http://www.adacore.com/sparkpro/tokeneer

Derby, U. o., 2009. *Research Onion Diagram (Based upon Saunders Et Al's Diagram).* [Online]
Available at: https://onion.derby.ac.uk/

Derby, U. o., 2009. *Research Onion Diagram (Based upon Saunders Et Al's Diagram).* [Online]
Available at: https://onion.derby.ac.uk/onion.pdf

Diagnostics, V., 2017. *Cardiology Services.* [Online]
Available at: http://vittalsdiagnostics.com/cardiology-services.php

Ebert, R., 2003. *avr-ada-devel.* [Online]
Available at: https://sourceforge.net/p/avr-ada/mailman/avr-ada-devel/

Ebert, R., 2012. *How to build and install AVR-Ada.* [Online]
Available at: https://sourceforge.net/p/avr-ada/wiki/Setup/

Ebert, R., 2012. *Tutorial for Programming AVR Devices in Ada.* [Online]
Available at: https://sourceforge.net/p/avr-ada/wiki/Introductory%20Tutorial/

Ebert, R., 2013. *Building and Installing an Ada Cross Compiler using the Build Script.* [Online]
Available at: https://sourceforge.net/p/avr-ada/wiki/BuildScript/

Ebert, R., 2015. *Building and Installing an Ada Cross Compiler Tool Suite.* [Online]
Available at: https://sourceforge.net/p/avr-ada/wiki/BuildCrossCompiler/

Ebrahim, A., 2010. *Assembly Language and Embedded Systems Development.* [Online]
Available at: http://mathscitech.org/articles/assembly-value

Electronics, L. A., 2015. *How to Program an AVR chip using a USBASP.* [Online]
Available at: http://www.learningaboutelectronics.com/Articles/Program-AVR-chip-using-a-USBASP-with-10-pin-cable.php

Etienne, V., 2013. *Vincent Etienne.* [Online]
Available at: https://plus.google.com/+VincentEtienneCFF

Google, 2017. *ECG Monitors.* [Online]
Available at:
https://www.bing.com/images/search?q=latest%20ECG%20monitor&qs=n&form=QBIR&pq=latest%20ecg%20monitor&sc=0-14&sp=-1&sk=

Hills, C., 2014. *Implementing MISRA-C:2012,* s.l.: Phaedrus Systems.

Hitachi, 2017. *HD44780 Datasheet (PDF).* [Online]
Available at: http://www.alldatasheet.com/datasheet-pdf/pdf/63673/HITACHI/HD44780.html

Institute, I. R., 1998. *Ada Used to Develop Medical Analytical Systems.* [Online]
Available at: http://archive.adaic.com/docs/flyers/medsys.html

Instruments, T., 1942 - 2017. *Reflective Optical Sensors.* [Online]
Available at: http://uk.rs-online.com/web/generalDisplay.html?id=aboutRS
[Accessed 15 June 2017].

Jake, 2014. *Log #1 - BP Monitor!.* [Online]
Available at: https://hackaday.io/project/1026/logs

joelparkerhenderson, 2017. *How to install GCC piece by piece with GMP, MPFR, MPC, ELF, without shared libraries?.* [Online]
Available at: https://stackoverflow.com/questions/9450394/how-to-install-gcc-piece-by-piece-with-gmp-mpfr-mpc-elf-without-shared-libra

John W. McCormick, P. C. C., 2015. Introduction and Overview. In: *Building High Integrity Applications With SPARK.* Cambridge: Library of Congress Cataloging.

Kline, M., 2017. *C++ On Embedded Systems.* [Online]
Available at: http://bitbashing.io/embedded-cpp.html

Leif, R. C., 1993. *A Biomedical Engineer's View of Ada.* [Online]
Available at: http://archive.adaic.com/docs/flyers/biomed.html

Louic, 2008. *Setting fuse-bits on Atmel Atmega8 AVR.* [Online]
Available at: http://blog.louic.nl/?p=161

Margi, 2017. *Pulsesensor Heart Rate Pulse Sensor Sensor Module For Arduino Raspberry pi Pop.* [Online]
Available at: http://www.ebay.co.uk/itm/Pulsesensor-Heart-Rate-Pulse-Sensor-Sensor-Module-For-Arduino-Raspberry-pi-Pop-/182449915927?hash=item2a7adcc817:g:zI8AAOSwB-1YnL-n

Margj17, 2017. *USB ISP USBASP Programmer AVR ATMEL ATMega8 Download Pin IDC Cable 3.3V 5V - UK.* [Online]
Available at: http://www.ebay.co.uk/itm/USB-ISP-USBASP-Programmer-AVR-ATMEL-ATMega8-Download-Pin-IDC-Cable-3-3V-5V-UK-/182084372891?var=&hash=item2a6513099b:m:mLrxFCStcL8qz_a6Mtsz_jg

Martin Carlisle, B. F., 2012. *IRONSIDES: DNS With No Single-Packet Denial of Service or Remote Code Execution Vulnerabilities ,* USAFA, CO USA: Department of Computer Science US Air Force Academy .

Mieiel6792014, 2017. *Serial IIC/I2C/TWI 2004 204 20X4 Character LCD Module Display For Arduino Blue.* [Online]
Available at: http://www.ebay.co.uk/itm/Serial-IIC-I2C-TWI-2004-204-20X4-Character-LCD-Module-Display-For-Arduino-Blue-/262317442560?hash=item3d13568600:g:BxwAAOSwubRXLALb

MikroElektronika, 2017. *Microcontroller – Device Overview.* [Online]
Available at: https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/chapter/pic16f887-microcontroller-device-overview/

M, J., 2014. *How to Install The Latest AVRDude 6.1 in Ubuntu 14.04.* [Online]
Available at: http://ubuntuhandbook.org/index.php/2014/09/install-avrdude-6-1-ubuntu-1404/

moore_estates, 2017. *White 128X64 OLED LCD LED Display Module For Arduino 0.96" I2C IIC Serial UK New.* [Online]
Available at: http://www.ebay.co.uk/itm/162042315981

Moy, Y., 2013. *MISRA-C 2012 vs SPARK 2014, the Subset Matching Game.* [Online]
Available at: http://www.spark-2014.org/entries/detail/misra-c-2012-vs-spark-2014-the-subset-matching-game

Nina V. Sviridova, K. S., 2015. *Human photoplethysmogram: New insight into chaotic characteristics.* [Online]
Available at: https://www.researchgate.net/figure/277027133_fig5_Components-of-PPG-signal-waveform-for-healthy-young-subjects

Ochem, Q., 2017. *The time has come for Ada.* [Online]
Available at: http://www.embedded.com/electronics-blogs/say-what-/4457545/The-time-has-come-for-Ada

Pflug, B., n.d. *Boeing Flies on 99% Ada.* [Online]
Available at: http://archive.adaic.com/projects/atwork/boeing.html

Point, T., 2017. *SDLC - Agile Model.* [Online]
Available at: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

Point, T., 2017. *SDLC - Iterative Model.* [Online]
Available at: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

Point, T., 2017. *SDLC - Spiral Model.* [Online]
Available at: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

Point, T., 2017. *SDLC - Waterfall Model.* [Online]
Available at: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Protostack, 2015. *USBASP Pinout – USBASP AVR Programmer.* [Online]
Available at: http://www.datasheetcafe.com/usbasp-pinout-avr-programmer/

Radcliffe, T., 2016. *python-vs-cc-embedded-systems.* [Online]
Available at: https://opensource.com/life/16/8/python-vs-cc-embedded-systems

Rowden, N., n.d. *ILTIS- Railway Traffic Control,* Germany: Aonix.

scooterboy101, 2017. *LCD Display Module White Blue 1602 16X2 HD44780 Arduino Raspberry PI 1x 3x 5x.* [Online]
Available at: http://www.ebay.co.uk/itm/LCD-Display-Module-White-Blue-1602-16X2-HD44780-Arduino-Raspberry-PI-1x-3x-5x-/162300749857?var=&hash=item25c9e0e821:m:mwLsfKFCx41y2i3nt5c-NZQ

Sinex, J., 1999. Pulse oximetry: Principles and limitations. *The American Journal.*

Singh, A. K., 2008. *Microcontroller and Embedded System.* Lucknow: Goel Institute of Technology.

Solutions, S. B., 1988 - 2017. *What is the Capability Maturity Model.* [Online]
Available at: http://www.selectbs.com/process-maturity/what-is-the-capability-maturity-model

SPARKFUN, 2003 - 2017. *Pulse Sensor.* [Online]
Available at: https://www.sparkfun.com/products/11574

Supervisor, A. G. S., 2017. *Discussion on effective programming languages - Weekly Meetings* [Interview] (March 2017).

Technology, M., 1998-2017. *Popular 8-Bit AVR Products.* [Online]
Available at: http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=30047

Tecom, 2017. *Stat Profile Prime.* [Online]
Available at: http://www.quickmedical.com/images/sku/large/40585.jpg

Teske, D., n.d. *Boeing flies on 99% Ada.* [Online]
Available at: http://archive.adaic.com/projects/atwork/boeing.html

Valencell, 2015. *Optical Heart Rate Monitoring: What you need to know.* [Online]
Available at: http://www.valencell.com/blog/2015/10/opticalheartratemonitoringwhatyouneedkno

Wallemburg, Y. M. a. A., 2012. *Tokeneer: Beyond Formal Program Verification.* Amstedam, Bath, Semantic Scholar, p. 8.

Wilson, R., 2016. *Is Tomorrow's Embedded-Systems Programming Language Still C?.* [Online]
Available at: http://systemdesign.altera.com/tomorrows-embedded-systems-programming-language-still-c/

Wood, R. J., 2010. *Resting Heart Rate Table.* [Online]
Available at: http://www.topendsports.com/testing/heart-rate-resting-chart.htm

Yannick Moy, S. M., 2013. *About SPARK 2014.* [Online]
Available at: http://www.spark-2014.org/about

# APPENDICES

## Appendix 1A – Ada Continues to revolutionise the world


**Eurofighter Typhoon**

BAE Systems are using the GNAT Pro development environment for host Ada compilation in the development of software for the Eurofighter's mission computers.

Read the case study

**Ship Self-Defense System (SSDS)**

Raytheon has delivered the Ship Self-Defense System (SSDS) Mk 2 using GNAT Pro for LynxOS within its multi-language software development environment. SSDS Mk 2 is a combat system that integrates and coordinates the sensors and weapons systems aboard a US Naval vessel to provide a coherent tactical picture for situational awareness, command and controls, and quick-reaction self-defense.

Read the press release  Read Raytheon's SDSS story

**Astute-Class Submarine Periscope**

This case study describes Thales UK's state-of-the-art non-hull-penetrating optronic mast for the United Kingdom Royal Navy's new Astute-class submarines, which provides greater flexibility in boat design and improved surface visibility while reducing the probability of detection. The optronic mast is powered by AdaCore partner, Wind River's VxWorks mission-critical real-time operating system (RTOS) submarine.

Read the case study *164 Kb (PDF)*

Reference to the above case studies – (Clearinghouse, 2009 - 2017).

# Appendix 1B – Ada continues revolutionising the world

## Air Traffic Control

**EuroControl** – European air traffic control organization (34 member states)
**Over 20 Government ATC Agencies** – including: Canada, China, UK and USA
**National Air Traffic Services (NATS)** – Interim Future Area Control Tools Support (iFACTS)
**NATS** – New En Route Centre (NERC) system
**Lockheed Martin** – En Route Automation Modernization (ERAM)
**Lockheed Martin** – User Request Evaluation Tool (URET)

## Aviation (Commercial)

**Airbus** – Aircraft: A320, A330, A340, A350 XWB
**Boeing** – Aircraft: 787, 777, 767,757, 747-400, 737-200, (and others)
**Canadian Air** – Aircraft: Regional Jet
**Fokker** – Aircraft: F 100
**Ilyushin** – Aircraft: Il-96M
**Raytheon** – Aircraft: Beechjet 400A (business jet)
**ULA** – Atlas and Delta Expendable Launch Vehicles
**Barco** – Advanced Jet Avionics Display

## Aviation (Defense)

**BAE Systems** – Aircraft: Harrier, Hawk
**Boeing** – K767 Tanker, C130 AMP
**EADS** – Aircraft: Tornado, A330 MRTT
**Eurocopter** – Aircraft: Tiger and NH90 helicopters
**Eurofighter GmbH** – Aircraft: Eurofighter
**Lockheed Martin** – Aircraft: F-16, F-22, C-130J
**Martin Baker** – Aircraft: F14, F18 and T-45

## Communications, Satellites and Receivers

**Inmarsat** – Voice/data communications to ships and mobile communications
**Hertz** – NeverLost GPS receiver
**RadarSat Intl.** – NeverLost GPS receiver
**Astrium in the UK** – Sentinel-1
**EADS CASA** – ARGOS 4
**CNES** – SPOT5/HELIOS2

## Defense Related Projects

**Honeywell** – Modular Azimuth Position System
**MBDA** – Missile systems
**Thales** – Astute-Class Submarine Periscope
**Raytheon** – Ship Self-Defense System (SSDS)
**Raytheon** – Extended Air Defense Test bed Simulation and Modeling Toolkit
**Raytheon** – Cobra Dane Radar System

## Rail Transportation

**Channel Tunnel**
**French High-Speed Rail System (TGV)**
**Hong Kong Suburban Rail**
**London Underground**
**Paris Metro and Suburban Rail**

## Financial Applications

**PostFinance**
**BNP Paribas**

## Others

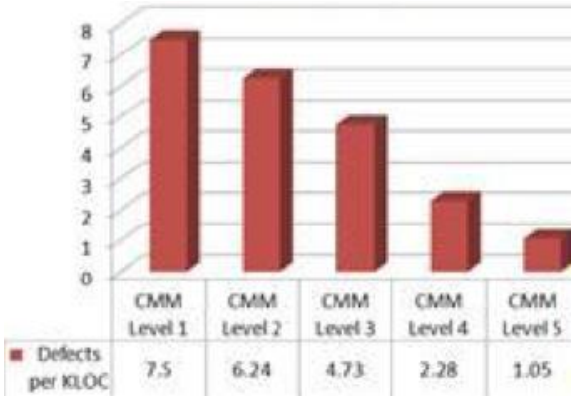
**Lawrence Livermore National Labs**
**National Ignition Facility**

# Appendix 2 – Software Capability Maturity Model

| | Level | Focus | Characteristics |
|---|---|---|---|
| 1 | Initial | None | Adhoc, chaotic or maturity. |
| 2 | Repeatable | Project Management | Software development successes are repeatable.<br>Basic project management may be used to track cost and schedule.<br>A process discipline reassures that existing practices are retained during stressful events.<br>Project status and service deliverables are visible (at major milestones or task completions).<br>Established processes are used to track cost, schedule and Functionality.<br>Significant risk of failing to meet cost and estimated time remains. |
| 3 | Defined | Software Engineering | Software process for management and engineering activities is documented, standardizedand integrated into a set of standard software processes for the organisation.<br>Standard process are established and improved.<br>Ensures that established objectives are appropriately addressed.<br>Repeats earlier successes from similar projects and scope.<br>Distinction of level 1 and 2 are scope of standards, process description and procedures. |
| 4 | Managed | Quality Process | Detailed measures of software process and product quality are collected.<br>Management can effectively control the software development effort.<br>Quantitative quality goal for software process and maintenance are set.<br>Software process, quality and maintenance are quantitatively understood.<br>Distinction between level 3 and 4 is the predictability of process performance. |
| 5 | Optimizing | Continuous Improvement | Focusing on continuous process enhancement and performance.<br>By process feedback and incremental and innovative technological improvements.<br>Quantitative process-improvement objectives are established, continually revised and used as criteria in management process improvements.<br>Distinction between level 4 and 5 is the type of process variation improvement addressed. |

(John W. McCormick, 2015, p. 3) (Solutions, 1988 - 2017)

Average Defect Density of Delivered Software




(Capers, 2000)

# Appendix 3 – SPARK 3 Level Analysis (Course Notes)

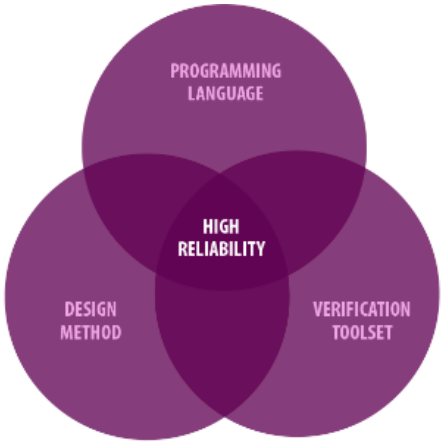

- Annotations to Ada.
  - Some required for data and information flow analysis.
  - Others allow to generate and prove verification conditions.
  - It is as well possible to integrate unchecked or even unimplemented Ada code.
- SPARK Ada code compiles with standard Ada compilers.


- **Level 1 Analysis**
  - Correct Ada syntax used.
  - SPARK-subset of Ada chosen, as described in the next subsection.
- **Level 2 Analysis**
  - **Data flow analysis.**
  - **Information flow analysis.**
- **Level 3 Analysis**
  - **Generation of verification conditions.**
  - **Proof of verification conditions** using automated and interactive theorem provers.
- When running Spark Ada always Level 1 and Level 2 Analysis is carried out.
  - There might however be a switch (pragma) to restrict analysis to Level 1 analysis only.


- **Data flow analysis.**
  - Checks input/output behaviour of parameters and variables.
  - Checks initialisation of variables.
  - Checks that changed and imported variables are used later (possibly as output variables).
- **Information flow analysis.**
  - Verifies interdependencies between variables.


- **Annotations** are added to the program expressing the correctness of the program.
  (Called **Pre-** and **Post-conditions**, more about this later.)
- SPARK Ada extracts **verification conditions**
  - They express that the program fulfils these correctness conditions.
- These verification are then fed into an **automated theorem prover**.
- The automated theorem prover is not always able to prove the correctness of the verification conditions, even if they are true.
- In that case one needs to feed verification conditions into an **interactive theorem prover**, e.g. the theorem prover **Coq**.

# Appendix 4 – SPARK's Design Methodology Diagram




(Yannick Moy, 2013)

# Appendix 5A – SPARK2C Solution Email (A)

Hi Jordan,

Thanks for speaking with me today. As we discussed, we do have a solution for your AVR development.

Our GNATPro Developer subscription comes with a SPARK2C toolchain which can convert your SPARK code to C code so that you can use any AVR toolchain to generate an executable to program the board. With the toolchain you also have access to:

- a restricted version of SPARK (called SPARK Discovery) which gives you access to the language subset and some basic proof capabilities (single prover).
- a native IDE (the GNAT Programming Studio or GPS)

You will have access to regular updates to the tools, including bugfixes and new features. New versions will be pushed several times per year.

This product is fully supported through an interface that is publicly visible – which means that all questions you ask will be visible to others, but you'll also see questions and answers asked by other customers subscribed to this product. This support is provided directly by the developers of the technology, with a guaranteed answer within 5 business days. One important element is that support is always provided on the latest version of the technology. This doesn't prevent you from archiving and using an older version though.

This product is sold as part of a yearly subscription, based on the number of unique (or named) users that will have access to it during the year. It starts at €6,000/year for one platform for up to 3 unique users.

Let me know if you have any questions regarding the GNATPro Developer offering or the SPARK2C tool. If you would like to move ahead with this option I can generate a formal quote to send to you. I will touch base with you again in a week or so to see how things are going. Feel free to reach out to me sooner if you have any questions.

Thanks,
--
Robert Tice
AdaCore Technical Account Manager
Email: tice@adacore.com
Cell: +1 929.426.5822
Office: +1 646.375.0726

## Appendix 5B – SPARK2C Solution Email (B)

Hi Jordan,

> Since this is a solution of SPARK2C, this means that program specifics not implemented in SPARK should be written in C, correct?

That is correct. You would typically have a BSP from somewhere that is in C that you can use to control peripherals and clocks and things. So you can use SPARK2C to generate most of your code, then modify the generated C code to do your system calls.

> However, is there any way to implement SPARK, Ada 2012 or above, and generate an executable to program the microprocessor/board?

Not with our technology. The AVR toolchain we had, that is discontinued, did not include support for Ada 2012. We do have support for ARM platforms with our currently supported technology. So if you can target something like an ARM Cortex M0, our technology will work for you. In that case you could use Ada2012 and SPARK Discovery to develop your application and generate ARM elf binaries. We have some examples of this on our GitHub: https://github.com/AdaCore/Ada_Drivers_Library/tree/master/examples

Let me know if you have any more questions.

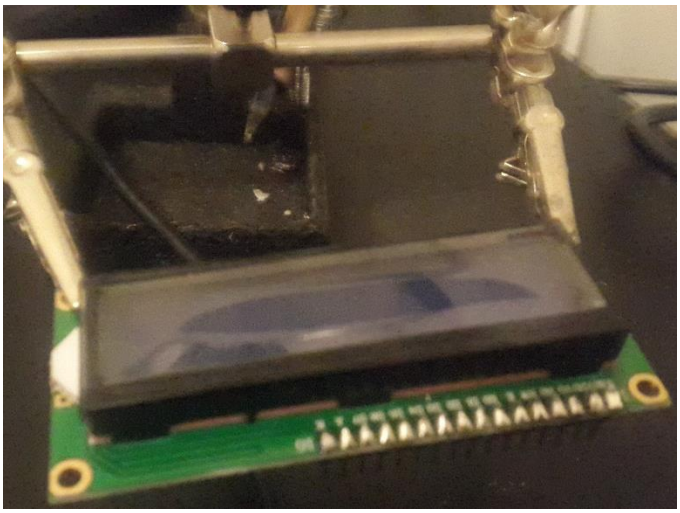
Thanks,

**PS.** Much of this solution was discussed over the phone, however, the available solution was discontinued in 2011, thus had no support for Ada 2012 or SPARK Tools.

# Appendix 6 – Soldering

**Before:**

**After:**



# Appendix 7A – LCD (Implementation) Package

```
--with Ada.Unchecked_Conversion;
with Interfaces;                    use Interfaces;

with AVR;                           use AVR;
with AVR.Wait;

with LCD.Wiring;

package body LCD is


   Processor_Speed : constant := LCD.Wiring.Processor_Speed;


   procedure Wait_10ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 10_000);

   procedure Wait_5ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 5_000);

   procedure Wait_1ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 1_000);

   procedure Wait_64us is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 64);

   -- set Enable high for a very short time to initiate write.
   procedure Toggle_Enable is
      use AVR.Wait;
      use LCD.Wiring;
   begin
      Enable := True;
      Wait_4_Cycles (1);
      Enable := False;
   end Toggle_Enable;

   procedure Output (Cmd : Unsigned_8; Is_Data : Boolean := False) is
      use Wiring;
   begin
      --  control pins
      ReadWrite := False;
      RegisterSelect := Is_Data;

      --  write data
      if Wiring.Bus_Width = Mode_4bit then
         --  high nibble first
         Data0 := (Cmd and 16#10#) /= 0;
         Data1 := (Cmd and 16#20#) /= 0;
         Data2 := (Cmd and 16#40#) /= 0;
         Data3 := (Cmd and 16#80#) /= 0;

         Toggle_Enable;

         --  then low nibble
         Data0 := (Cmd and 16#01#) /= 0;
         Data1 := (Cmd and 16#02#) /= 0;
         Data2 := (Cmd and 16#04#) /= 0;
         Data3 := (Cmd and 16#08#) /= 0;

         Toggle_Enable;
      else --  8 bit mode
         null; -- !!! FIXME
      end if;

      if Is_Data then
         Wait_1ms;
      else
         Wait_10ms;
      end if;
   end Output;

   procedure Init is
      use LCD.Wiring;
   begin
      --  set data direction registers for output of control and data pins
      Enable_DD         := DD_Output;
      ReadWrite_DD      := DD_Output;
      RegisterSelect_DD := DD_Output;
      Data0_DD := DD_Output;
      Data1_DD := DD_Output;
      Data2_DD := DD_Output;
      Data3_DD := DD_Output;
      --  wait at least 16ms after power on
      Wait_10ms;
      Wait_10ms;
      --  write 1 into pins 0 and 1
      Data0 := True;
      Data1 := True;
      Toggle_Enable;
      Wait_5ms;
      --  send last command again (is still in register, just toggle E)
      Toggle_Enable;
      Wait_64us;
      --  send last command a third time
      Toggle_Enable;
      Wait_64us;
      --  set 4 bit mode, clear data bit 0
      Data0 := False;
      -- now we can use the standard Command routine for set up
      if Wiring.Bus_Width = Mode_4bit then
         case Display.Height is
            when 1 => Command (Commands.Mode_4bit_1line);
            when 2 => Command (Commands.Mode_4bit_2line);
            when others => null;
         end case;
```

```
      else -- mode_8bit
         null;  --  !!! FIXME
      end if;
      Command (Commands.Display_On);
      Clear_Screen;
      Command (Commands.Entry_Inc);
   end Init;


   --  output at the current cursor location
   procedure Put (C : Character) is
   begin
      Output (Character'Pos (C), Is_Data => True);
   end Put;


   --  output at the current cursor location
   procedure Put (S : AVR_String) is
   begin
      for C in S'Range loop
         Put (S(C));
      end loop;
   end Put;


   --  output the command code Cmd to the display
   procedure Command (Cmd : Command_Type) is
   begin
      Output (Unsigned_8 (Cmd), Is_Data => False);
   end Command;

   --  clear display and move cursor to home position
   procedure Clear_Screen is
   begin
      Command (Commands.Clear);
   end Clear_screen;
   --  move cursor to home position
   procedure Home is
   begin
      Command (16#02#);
   end Home;


   --  move cursor into line Y and before character position X.  Lines
   --  are numbered 1 to 2 (or 1 to 4 on big displays).  The left most
   --  character position is X = 1.  The right most position is
   --  defined by Lcd.Display.Width;
   procedure GotoXY (X : Char_Position; Y : Line_Position)
   is
   begin
      case Y is
         when 1 => Command (16#80# + Command_Type (X) - 1);
         when 2 => Command (16#C0# + Command_Type (X) - 1);
      end case;
   end GotoXY;

end LCD;
```

# Appendix 7B – LCD (Specification) Package

```
package LCD is
   pragma Preelaborate;


   package Display is
      Width : constant := 16;  -- max number of characters in a line
      Height : constant := 2;  -- number of lines
   end Display;

   type Line_Position is new Unsigned_8 range 1 .. Display.Height;
   type Char_Position is new Unsigned_8 range 1 .. Display.Width;

   type Command_Type is new Unsigned_8;


   procedure Init;
   --  initialize display

   procedure Put (C : Character);
   procedure Put (S : AVR_String);
   --  output at the current cursor location

   procedure Command (Cmd : Command_Type);
   --  output the command code Cmd to the display

   procedure Clear_Screen;
   --  clear display and move cursor to home position

   procedure Home;
   --  move cursor to home position

   procedure GotoXY (X : Char_Position; Y : Line_Position);
   --  move cursor into line Y and before character position X.  Lines
   --  are numbered 1 to 2 (or 1 to 4 on big displays).  The left most
   --  character position is X = 1.  The right most position is
   --  defined by Display.Width;
   package Commands is
      Clear                  : constant Command_Type := 16#01#;
      Home                   : constant Command_Type := 16#02#;

      --  interface data width and number of lines
      Mode_4bit_1line        : constant Command_Type := 16#20#;
      Mode_4bit_2line        : constant Command_Type := 16#28#;
      Mode_8bit_1line        : constant Command_Type := 16#30#;
      Mode_8bit_2line        : constant Command_Type := 16#38#;

      --  display on/off, cursor on/off, blinking char at cursor position
      Display_Off            : constant Command_Type := 16#08#;
      Display_On             : constant Command_Type := 16#0C#;
      Display_On_Blink       : constant Command_Type := 16#0D#;
      Display_On_Cursor      : constant Command_Type := 16#0E#;
      Display_On_Cursor_Blink : constant Command_Type := 16#0F#;

      --  entry mode
      Entry_Inc              : constant Command_Type := 16#06#;
      Entry_Dec              : constant Command_Type := 16#04#;
      Entry_Shift_Inc        : constant Command_Type := 16#07#;
      Entry_Shift_Dec        : constant Command_Type := 16#05#;

      --  cursor/shift display
      Move_Cursor_Left       : constant Command_Type := 16#10#;
      Move_Cursor_Right      : constant Command_Type := 16#14#;
      Move_Display_Left      : constant Command_Type := 16#18#;
      Move_Display_Right     : constant Command_Type := 16#1C#;
   end Commands;

private
   pragma Inline (Command);

end LCD;
```

# Appendix 7C – LCD (Wiring) Package

```
with AVR;                           use AVR;
with AVR.MCU;

private package LCD.Wiring is
   pragma Preelaborate;

   type Bus_Mode is (Mode_4bit, Mode_8bit);

   Bus_Width         : constant Bus_Mode := Mode_4bit;

   Data_Port         : Nat8 renames MCU.PORTA;
   Data_DD           : Nat8 renames MCU.DDRA;

   Data0             : Boolean renames Data_Port (0);
   Data1             : Boolean renames Data_Port (1);
   Data2             : Boolean renames Data_Port (2);
   Data3             : Boolean renames Data_Port (3);
   Data0_DD          : Boolean renames Data_DD (0);
   Data1_DD          : Boolean renames Data_DD (1);
   Data2_DD          : Boolean renames Data_DD (2);
   Data3_DD          : Boolean renames Data_DD (3);

   RegisterSelect    : Boolean renames MCU.PORTA (5);
   RegisterSelect_DD : Boolean renames MCU.DDRA (5);
   ReadWrite         : Boolean renames MCU.PORTA (6);
   ReadWrite_DD      : Boolean renames MCU.DDRA (6);
   Enable            : Boolean renames MCU.PORTA (7);
   Enable_DD         : Boolean renames MCU.DDRA (7);

   --

   Processor_Speed   : constant := 4_000_000;

end LCD.Wiring;
```

# Appendix 8A – Wrapper Class (Implementation) Package

```
--# pragma SPARK_Mode (Off);
with AVR.MCU;
use AVR;

with AVR.Real_Time.Delays;
use AVR.Real_Time.Delays;

with AVR.Strings;
use AVR.Strings;

with AVR.Strings.Edit.Generic_Integers;
use AVR.Strings.Edit.Generic_Integers;

with Commands;
use Commands;

with Interfaces;
use Interfaces;

with AVR.Wait;


package body Wrapper_class
is

--start of ADC
   ADC_Start_Conversion : Boolean renames MCU.ADCSRA_Bits (MCU.ADSC_Bit);
   ADC_Enable           : Boolean renames MCU.ADCSRA_Bits (MCU.ADEN_Bit);
   ADC_Interrupt_Flag   : Boolean renames MCU.ADCSRA_Bits (MCU.ADIF_Bit);
   ADC_Left_Just_Read   : Boolean renames MCU.ADMUX_Bits (MCU.ADLAR_Bit);

   procedure Init_Standard_Input   is
   begin
   end Init_Standard_Input;

   procedure Init_Standard_Output   is
   begin
   end Init_Standard_Output;

-->Initializes ADC
   procedure Set_Analog_As_Input (Item : in Unsigned_8) is
   begin
      -- code from aVR ada to get pin 56 into item
      MCU.DDRC_Bits := (others => Item);
   end Set_Analog_As_Input;

   procedure Set_ADMUX_Reg (Item : in Unsigned_8) is
   begin
      -- code from aVR ada to get pin 56 into item
      MCU.ADMUX :=  Item;
   end Set_ADMUX_Reg;

   procedure Set_ADCSRA_Reg (Item : in Unsigned_8) is
   begin
      -- code from aVR ada to get pin 56 into item
      MCU.ADCSRA :=  Item;
   end Set_ADCSRA_Reg;

   procedure Set_DIDR0_Reg (Item : in Unsigned_8) is
   begin
      -- code from aVR ada to get pin 56 into item
      MCU.DIDR0 :=  Item;
   end Set_DIDR0_Reg;
-->End of ADC Initialization


-->Starts ADC Conversion
   procedure Set_Start_Conversion (Item : in Boolean) is
   begin
      -- code from aVR ada to get pin 56 into item
      ADC_Start_Conversion :=  Item;
   end Set_Start_Conversion;
```

```
   procedure Reset_ADC_Flag (Item : in Boolean) is
   begin
      -- code from aVR ada to get pin 56 into item
      ADC_Interrupt_Flag :=  Item;
   end Reset_ADC_Flag;
-->end of ADC Conversion Reset flag to start new conv

-->starts checking if adc conversion is complete
   procedure get_Conv_Complete (Item : out Boolean) is
   begin
          --returns interrupt flag
                Item := ADC_Interrupt_Flag;
   end get_Conv_Complete;
-->end of checking if adc conversion is complete

-->start getting ADC result
   procedure get_Low_Byte (Item : out Boolean) is
   begin
      -- item becomes low byte
      Item := MCU.ADCL;
   end get_Low_Byte;

   procedure get_High_Byte (Item : out Boolean) is
   begin
      -- item becomes high byte
      Item := MCU.ADCH;
   end get_High_Byte;
-->end of get ADC Result

-->end of ADC
--Start of LCD

     --processor
   Processor_Speed : constant := 16_000_000;
      --LCD data buses
   Data0             : Boolean renames MCU.PORTD_Bits (PORTD0_Bit);
   Data1             : Boolean renames MCU.PORTD_Bits (PORTD1_Bit);
   Data2             : Boolean renames MCU.PORTD_Bits (PORTD2_Bit);
   Data3             : Boolean renames MCU.PORTD_Bits (PORTD3_Bit);
   Data0_DD          : Boolean renames MCU.DDRD_Bits (DDD0_Bit);
   Data1_DD          : Boolean renames MCU.DDRD_Bits (DDD1_Bit);
   Data2_DD          : Boolean renames MCU.DDRD_Bits (DDD2_Bit);
   Data3_DD          : Boolean renames MCU.DDRD_Bits (DDD3_Bit);

   --LCD control bits

   RegisterSelect    : Boolean renames MCU.PORTD_Bits (PORTD4_Bit);
   RegisterSelect_DD : Boolean renames MCU.DDRD_Bits (DDD4_Bit);
   ReadWrite         : Boolean renames MCU.PORTD_Bits (PORTD5_Bit);
   ReadWrite_DD      : Boolean renames MCU.DDRD_Bits (DDD5_Bit);
   Enable            : Boolean renames MCU.PORTD_Bits (PORTD6_Bit);
   Enable_DD         : Boolean renames MCU.DDRD_Bits (DDD6_Bit);

   procedure Wait_10ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 10_000);

   procedure Wait_1ms is new AVR.Wait.Generic_Wait_Usecs
     (Crystal_Hertz => Processor_Speed,
      Micro_Seconds => 1_000);

   procedure Wait_1sec is
   begin
          delay 1.0;
   end Wait_1sec;
```

```
   procedure Toggle_Enable is
      use Wiring;
          use AVR.Wait;
   begin
      Enable := True;
      Wait_4_Cycles (1);
      Enable := False;
   end Toggle_Enable;

--Start of LCD Init
   procedure Init_LCD_DataRegisters (Item : in Usigned_8) is
   begin
      Enable_DD                := Item;
      ReadWrite_DD             := Item;
      RegisterSelect_DD := Item;

      Data0_DD := Item;
      Data1_DD := Item;
      Data2_DD := Item;
      Data3_DD := Item;
   end Set_LCD_DataRegisters;

   procedure Set_LCD_Data0 (Item : in Boolean) is
   begin
          Data0 := Item;
   end Set_LCD_Data0;

   procedure Set_LCD_Data1 (Item : in Boolean) is
   begin
          Data1 := Item;
   end Set_LCD_Data1;

   procedure Set_LCD_Data2 (Item : in Boolean) is
   begin
          Data2 := Item;
   end Set_LCD_Data2;
```

```
   procedure Set_LCD_Data3 (Item : in Boolean) is
   begin
          Data3 := Item;
   end Set_LCD_Data3;

   procedure Set_Command_4Bit (Item : in Boolean) is
   begin
               if (Item = True) then
                       Commands.Mode_4bit_2line;
                       Commands.Display_On;
                       LCD.Clear_Screen;
                       Commands.Entry_Inc;
               end if;
   end Set_Command_4Bit;

--End of LCD Init
--Start of LCD Output Procedure
   procedure Set_RorW (Item : in Boolean) is
   begin
      ReadWrite := Item;
   end Set_RorW;

   procedure Set_RegSel (Item : in Boolean) is
   begin
      RegisterSelect := Item;
   end Set_RegSel;

   procedure Set_Higher_Nibble_Read_First_D0 ( Item : in Unsigned_8) is
   begin
               Data0 := Item;
   end Set_Higher_Nibble_Read_First_D0;

   procedure Set_Higher_Nibble_Read_First_D1 ( Item : in Unsigned_8) is
   begin
               Data1 := Item;
   end Set_Higher_Nibble_Read_First_D1;
```

```
   procedure Set_Higher_Nibble_Read_First_D2 ( Item : in Unsigned_8) is
   begin
               Data2 := Item;
   end Set_Higher_Nibble_Read_First_D2;

   procedure Set_Higher_Nibble_Read_First_D3 ( Item : in Unsigned_8) is
   begin
               Data3 := Item;
   end Set_Higher_Nibble_Read_First_D3;

   procedure Set_Lower_Nibble_D0 ( Item : in Unsigned_8) is
   begin
               Data0 := Item;
   end Set_Lower_Nibble_D0;

   procedure Set_Lower_Nibble_D1 ( Item : in Unsigned_8) is
   begin
               Data1 := Item;
   end Set_Lower_Nibble_D1;

   procedure Set_Lower_Nibble_D2 ( Item : in Unsigned_8) is
   begin
               Data2 := Item;
   end Set_Lower_Nibble_D2;

   procedure Set_Lower_Nibble_D3 ( Item : in Unsigned_8) is
   begin
               Data3 := Item;
   end Set_Lower_Nibble_D3;
--End of LCD Output Procedure
--Start lcd display Integer as a string
        procedure set_Display_Int(Item : in Integer) is
                L : Unsigned_8;
                T : AVR_String(1 .. L) := (others => ' ');
        begin
                        Put_U32 (Value  => Unsigned_32(Item),
                                 Target => T,
                                 Last   => L);
                        Put(T(2..L));
        end set_Display_Int;
--End LCD display Integer as string
--> End LCD

--> Start Timer

        CS12 : Boolean renames MCU.TCCR1B_Bits (MCU.CS12_Bit);
        WGM12 : Boolean renames MCU.TCCR1B_Bits (MCU.WGM12_Bit);
        Interrupt_Flag : Boolean renames MCU.TIFR1_Bits (MCU.OCF1A_Bit);
        TCNT1 : Unsigned_16 renames MCU.TCNT1;

--Start Timer Init
   procedure Set_Timer_Props (Item : in Boolean) is
   begin
      CS12 := Item;
         WGM12 := Item;
   end Set_Timer_Props;


   procedure Set_TimerCounter_InitVal (Item : in Unsigned_16) is
   begin
               TCNT1 := Item;
   end Set_TimerCounter_InitVal;

   procedure Set_CompareVal (Item : in Unsigned_16) is
   begin
               MCU.OCR1A  := Item;
   end Set_CompareVal;
--End Timer Init
--Start Timer Procedures
   procedure Reset_Interrupt (Item : out Boolean) is
   begin
               Interrupt_Flag := Item;
   end Reset_Interrupt;

   procedure Check_Interrupt (Item : out Boolean) is
   begin
               Item := Interrupt_Flag;
   end Check_Interrupt;
--End Timer Procedures

--> End Timer
end Wrapper_class;
```

# Appendix 8B – Wrapper Class (Specification) Package

```
-- This file was created by Jordan Lee Mauro Buhagiar

--#pragma SPARK_Mode (On);

with Interfaces; use Interfaces;

package Wrapper_class
is
-->Start ADC |
        -- this procedure initialises standard input
        procedure Init_Standard_Input
     with Global => (Output => Standard_Input),
          Depends => (Standard_Input  => null);

   -- this procedure initialises standard output
        procedure Init_Standard_Output
     with Global => (Output => Standard_Output),
          Depends => (Standard_Output  => null);
-->Init ADC Registers
   -- as_get gets a character from console IO
        procedure Set_Analog_As_Input (Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   -- as_get gets a character from console IO
        procedure Set_ADMUX_Reg (Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

        procedure Set_ADCSRA_Reg (Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

        procedure Set_DIDR0_Reg (Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));
-->End of Init
```

```
-->Starting ADC Conversion
        procedure Set_Start_Conversion (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

        procedure Reset_ADC_Flag (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));
-->End of Conversion

-->Start ADC Conv Complete Check
        procedure get_Conv_Complete (Item : out Boolean) ;--@
   --# with Global => (In_Out => (Standard_Input)),
   --# Depends=> (Standard_Input => Standard_Input, Item =>     Standard_Input));
-->End of Check

-->Start getting ADC Result
        procedure get_Low_Byte (Item : out Boolean) ;--@
   --# with Global => (In_Out => (Standard_Input)),
   --# Depends=> (Standard_Input => Standard_Input, Item =>     Standard_Input));

        procedure get_High_Byte (Item : out Boolean)  ;--@
   --# with Global => (In_Out => (Standard_Input)),
   --# Depends=> (Standard_Input => Standard_Input, Item =>     Standard_Input));

-->End ADC
```

```
-->Start LCD

   procedure Init_LCD_DataRegisters (Item : in Usigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_LCD_Data0 (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_LCD_Data1 (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_LCD_Data2 (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_LCD_Data3 (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Command_4Bit (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_RorW (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_RegSel (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Higher_Nibble_Read_First_D0 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Higher_Nibble_Read_First_D1 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Higher_Nibble_Read_First_D2 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Higher_Nibble_Read_First_D3 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Lower_Nibble_D0 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Lower_Nibble_D1 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Lower_Nibble_D2 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_Lower_Nibble_D3 ( Item : in Unsigned_8) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure set_Display_Int(Item : in Integer) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

-->End LCD

-->Start Timer
   procedure Set_Timer_Props (Item : in Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_TimerCounter_InitVal (Item : in Unsigned_16) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Set_CompareVal (Item : in Unsigned_16) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Reset_Interrupt (Item : out Boolean) ;--@
   --# with Global => (In_Out => Standard_Output),
   --# Depends=> (Standard_Output => (Item, Standard_Output));

   procedure Check_Interrupt (Item : out Boolean) ;--@
   --# with Global => (In_Out => (Standard_Input)),
   --# Depends=> (Standard_Input => Standard_Input, Item =>     Standard_Input));
-->End Timer

end Wrapper_Class;
```