Cape Peninsula
University of Technology

Design of Asset tracking device with GPRS Interface

by

Robin Maharaj

Thesis submitted in fulfilment of the requirements for the degree

Master of Engineering: Electrical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Dr A.K. Raji
Co-Supervisor: Mr Q.J.Bart

Bellville
October 2018

DECLARATION

I, Robin Maharaj, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

IOT devices have the potential to improve asset lifecycle optimization because of their ability to provide relevant real time data to high level applications. This data with minimal latency can assist asset managers to control the behaviour of assets and asset users to optimize asset lifecycle cost. There are many environments that require asset tracking devices but this design focussed on motor vehicles with auxiliary functions and apparatus.

This research work documented the design of an asset tracking device built and tested on a 32.bit microcontroller platform with built-in CAN peripheral. This design resolved handling multiple serial interfaces collating data simultaneously concatenating this data and transmitting the data via GPRS interface as a single UDP sentence. Apart from interfacing various serial interfaces to the Stm24f4 this design also implemented a Wireless module as well as a multichannel ADC Module.

This design was accomplished by researching and implementing software techniques as well as researching the hardware/firmware in terms of DMA and Nested Vector Interrupt Controller of the STM32 devices.
The solution that this design will accomplish is to provide the industry an asset-tracking device with data capturing functionality capable of delivering the above needs at reasonable data cost. The device designed in this thesis is the client device of asset tracking network. This design was accomplished on a proof of concept basis delivering built hardware in the form of various application modules interfaced to a 32 –bit microcontroller via uart, SPI and CAN.

ACKNOWLEDGEMENTS

First, all praise be to GOD for giving me the opportunity to complete my research/design work successfully and for acquiring new knowledge in the process that I know He has plans for.

My sincere gratitude and appreciation to my supervisor, DR A.K Raji, for his invaluable guidance, enthusiastic help and consistent encouragement throughout the entire research project.

Also would like to extend my thanks to my co-supervisor Mr Quinton Bart.

I would like to dedicate this thesis to my late parents, my wife Gail, my three kids Aston, Aidon and Sashni for all the unconditional love and support and patience, for all the encouragement. For all the Sacrifices made in terms of time I would like to extend a special thanks to my kids I know they felt it most thanks, love you guys.

Robin Maharaj

TABLE OF CONTENTS

List of Tables

List of Figures

Appendices

GLOSSARY

Terms/Acronyms/Abbreviations	Definition/Explanation
ADC	Analogue to Digital Convertor
APN	Access Point Name
ATD	Asset Tracking Device
CAN	Controller Area Network
CPU	Central Processing Unit
DAC	Digital to Analog Convertor
DMA	Direct Memory Access
DNS	Direct Memory Access
ECU	Engine/Electronic Control Unit
EMI	Electromagnetic Induction
FDM	Frequency Division Multiplexing
FIFO	First In First Out
FSM	Finite State Machine
FTP	File transfer Protocol
GNSS	Global Navigation Satellite system
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GTP	GPRS Tunnelling Protocol
HTTP	Hyper Text Transfer Protocol
I/O	Input/output
IOT	Internet of Things
IP	Internet Protocol
ISR	Interrupt Service Routine
LAN	Local Area Network
MISO	Master In Slave Out
MOSI	Master Out Slave In

NMEA	National Marine Electronics Association
NTC	Negative Temperature Coefficient
OBD	On Board Diagnostics
PWM	Pulse width modulation
RF	Radio Frequency
SAE	Society of Automotive Engineers
SCI	Serial communications interface
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UDS	Unified Diagnostic Services
WAN	Wide area Network

Chapter 1
Introduction

1.1 Background

With the present trends in the modern digital era in terms of the Internet of things, there is an ever-growing need for devices to be connected to internet or to a remote control centre. The industry has undergone huge technologically advancements in the last twenty years both from a mechanical perspective as well as from an electronic and power electronic perspective. Low cost microcontrollers have advanced quiet tremendously giving us 32-bit microprocessor capability as well as numerous peripherals with numerous communications interfaces. In addition, the cost of this technology has decreased exponentially making asset tracking a viable option for technology solutions.

The public sector more so the government departments have thousands of specialised vehicles that are currently being abused. In the South African environment there is presently no device that is capable of bringing the different management systems together into one system for proper automated audit and asset management process. In this dissertation, we will embark on capturing data from numerous modules through different communication interfaces or techniques and relay this data to a remote control centre.

This project will focus on research methods in terms of acquiring data, processing this data converting to a data algorithm for transmission. The project will also look at efficient buffering and storage techniques as not all the collated data will be transmitted. This research project endeavours to solve these issues using a 32-bit micro-controller as an asset tracking and data acquisition device. It also looked at cost of transmitting data using to different protocols namely UDP and TCP in the form of HTTP. All of these translate into a perimeter-based asset-tracking device

This project will also employ GPS, GSM modules and Digital signal processing techniques using the 32-bit arm microprocessor. The design will tap into an OBD control unit using an OBD interpreter interfaced to an arm microprocessor via a UART peripheral.

The design has in addition interfaced a RF biometric interface with panic button via the microcontroller for the purpose monitoring security personnel of the public sector like Metro police or extended use by the public. The purpose of this interface is to stream data such as pulse rate, blood pressure as well distress signal to a control centre via the microcontroller. In the RF biometric module, the research has explored different radio frequency communication interfaces between the biometric device and the microcontroller like RF or wireless technologies and implemented the most efficient and robust interface.

The purpose of this project is to design an asset tracking control unit with a GPRS comms interface that would be able to connect remote control centre. The asset-tracking device will monitor input signals connected locally to the I/O module; it will communicate to the OBD unit of the vehicle through an OBD interpreter, the CAN interface will also transmit live vehicle data but this interface will also allow the control centre limited control functions; the microcontroller will relay buffered GPS data and it will also stream biometric data and a distress signal to a Remote Control Centre .It will process and transmit auxiliary data acquired by the asset-tracking Device.

The purpose of this specific asset-tracking modules is to capture engine running hours of the vehicles as the auxiliary equipment like the Hydraulic pumps for the PTO (Hydraulic cherry picker) is running while the vehicle engine is running but vehicle is stationery. This means vehicle mileage is not a good indication of vehicle use as well as not an accurate measure of fuels consumed by vehicle per mileage. With asset tracking device installed Vehicle maintenance can be carried out on running hours as opposed to mileage of the vehicle. This will ensure that an accurate cost of fuel consumed by the vehicle to operate auxiliary can be isolated from the cost to move the vehicle around. In the public sector the vehicles are not treated as mere vehicles but operational assets and thus the maintenance strategy surrounding these heavy duty vehicles.

Also the control centre will have vehicle OBD status at any given time as well as fault status of the engine and if required other control units at all times. If operator of vehicle operates vehicle or any equipment during the fault conditions the control centre will be informed in real time. The call centre can dispatch staff automatically during these exception conditions or the software can be programmed to automatically inform technical staff.

1.2 Statement of the research Problem

IOT devices have the potential to improve asset lifecycle optimization because of their ability to provide relevant real time data to high level applications. This data with minimal latency can assist asset managers to make and control the behaviour of assets and asset users to optimize asset lifecycle cost.

But the present perception of devices or applications in the IOT Space is that they are resource intensive as well as that they have short comings in terms of data efficiency. This

leads to asset managers neglecting to implement these solutions as part of asset optimization applications.

This research design embarked to investigate these shortcomings and implement solutions to reduce data cost using a 32 microcontroller. This design will look at these solutions using both hardware and software routines to improve on data cost and data latency issues using a microcontrollers

1.3 Objectives of the Research

The aim of this research is to achieve a barebones approach to the design of asset tracking device using different communication devices to interface into microcontroller to capture data, buffer data and transmit it with minimal data cost.

a. this thesis will focus on design to capturing data via different interfaces, which will handle data coming in at different rates from different communications interface

b. this thesis will also look at arbitration as well as priorities of the different interfaces as well as the microcontrollers handling direct io.

c. in term of data transmission, this research will experiment with different ways of sending data and examining the cost variables and reliability of sending data via gsm/gprs methods.

d. This research project will focus on research methods in terms of acquiring data, processing this data converting to a data algorithm for transmission. This thesis will also look at efficient buffering and storage techniques as not all the collated data will be transmitted. It will attempt to solve these issues using a 32-bit micro-controller as an asset tracking and data acquisition device.

e. This research will look at cost of transmitting data using different protocols namely UDP and HTTP.

f. To develop an RF biometric interface with panic button to the microcontroller for the purpose monitoring security personnel of the public sector like Metro police. The purpose of this interface is to stream data such as pulse rate, blood pressure as well distress signal to a control centre via the microcontroller. This will add value to the device from an operational perspective.

1.4 **Significance of the Research**

This research intends to significantly change the mind-set in terms of present development platforms in the South African context. It also intends to demystify the role of IOT devices with regard to asset life cycle management as to implement an asset tracking device with the purpose of collating data for the purpose of intelligently using it for remote call centre as well as for data analysis for asset life cycle management.

Very few research projects have been undertaken to implement an asset tracking device that will serve as an asset tracking device that will also serve as an automated data interface to assist with life cycle asset management optimisation in South Africa.

The purpose of this project is to build the foundation to generate and automate data from devices that have an impact on asset life cycle that can be optimised if not managed. In term of the Internet of things, there are a lot of assets that that have high maintenance cost and acquiring data from these devices as well as offering some controllability in many instances will improve maintenance cost and in turn optimise asset life cycle.

In this particular project, there are also numerous safety spinoffs like the RF biometric interface it assists the personnel using the vehicle in case they incapacitated they can press the panic button and if that is not possible and if their vitals are that of distress condition the device will relay a distress signal to the control centre and the Control centre can dispatch emergency personnel to that location.

1.5 Delineation of the thesis

The interfacing of the various modules poses numerous design challenges which includes hardware, software and electromagnetic interference.

This thesis will not cover the design or implementation of the control centre but will briefly discuss it with some pseudo code. The testing of the design will be done via a udp sockets application called sockets tester.

This thesis will be limited to the design and implementation of only the ATD, a client module that will communicate to a control centre and Central Remote Database

The design will be implemented on a proof of concept basis and as a development prototype.

This design includes various modules and architectures to bring about a device that serves multiple functions as well has several benefits from asset life cycle optimisation to applications that can be extended to safety and security of personnel or to the user of the asset being tracked.

1.6 Organisation of the Thesis

This thesis is organised into different chapters for ease of reading as well as separating the different technologies and modules. The breakdown of the chapters is as per table of contents. Chapter 2 covers the theoretical research that was acquired during this project, Chapter 3 is Hardware implementation of the design. Chapter 4 highlights the analysis of design and some findings related to the design. And the final chapter concludes this research and gives some insight to future work in the line of this research.

Chapter 2

Literature review

2.1 Introduction

The concept of asset tracking has been around for many years now and has been employed to different assets for different purposes. This research investigated asset tracking and monitoring. This research has reviewed these types and combined different aspects of asset tracking and monitoring techniques for the purpose of asset optimisation. In South Africa however there are very few developers or students that have embarked on this field and if they did they have adopted the rapid development platform to implement these solutions. In the South African context students and developers need to adopt a “bare metals” approach to implementing solutions so that they can develop their own software libraries and techniques and have a better understanding of the technologies present. It has been a trend in “third world economies “to implement a lot of solutions using rapid development platforms, this has resulted in not understanding the technologies and creating a lag in terms of development of new products because they do not have their own libraries and the foundation and acquired knowledge to develop new products. There might be an argument that there is various open source libraries available to implement these solutions to this I again say that these solutions without a deep understanding of what is being done also does not put us third world countries on the platform to be ground breakers in the embedded systems environment.

2.2 Perspective of this project

It is important to put the design approach employed in this design project into perspective. The growing trends at present in terms of development of new products is to employ the rapid development platforms like Arduino, mbed and the likes. These platforms certainly have their place industry but the underlying issues in developing countries like South Africa it takes away the opportunity to understand the building blocks of microcontrollers and embedded system engineering; in these countries we find developers merely adapting first world country’s design to suite their own environments, they simply use libraries written by their 1st world counterparts in their applications without in-depth understanding the architecture of the hardware is being used. In order for these counties to acquire this deep understanding of the architecture they need to code from the data sheet and write their own libraries using low level coding languages so that they can become pioneers/experts in the development of embedded products. This will also give them the in-depth understanding to solve problems that is presently persistent in these countries in terms of embedded systems

without the need to pull in foreign help to solve some of their design issues. During this design it was one of the key objectives was to write “our own” libraries to implement this design, which was accomplished quite successfully.

2.3 Research Methodology

The research methodology used in the project is Applied Research Methodology. This research looked at problems industry and parastatals face with regard to asset lifecycle optimisation, and also looked at how technology from an embedded systems perspective or Internet of things perspective can bridge the gap for asset life cycle optimisation. A great deal of the research methodology comes from the tacit experience an electronic engineer acquired through a long career in the asset maintenance environment and the wants or the gaps that were presented over the years. It is the asking the questions of how the new digital era and IOT can solve old asset lifecycle issues. A key factor in terms of this design was also to understand the architecture of the control centre as to understand how the data needs to be presented to the control centre/asset lifecycle management software.

Having researched the above the next phase of this research was to research all the technologies that was planned to be implemented in this design so as to implement a streamlined design process negating errors and shortcomings other developers may have experienced in this field.

This research can be broken down into three areas viz: Research software techniques. Researching the hardware as well as communication interface. In addition, the third phase, once a working prototype is produce to monitor data transmission cost using different data transmission techniques and protocols.

2.3.1 Phase 1: Literature review and protocols Investigation

Literature review was conducted to understand the technologies and techniques available with this research, apply, and improve on techniques and technologies researched.

In order to be implement the design of an asset tracking control unit the incumbent will be required to do the following:

Use and interpret NMEA commands and data to implement the GPS module

This is a fundamental software requirement in order to carry out this project. The project requires an expert knowledge of embedded design and to be able to program a microprocessor in the C-language. C is a very power language that allows a programmer access the inner workings of microcontrollers. The project employs expert knowledge of implementation of buffers as well numerous modules all competing for CPU time. The design will also employ advanced c programming techniques as well as use Finite state machines to implement the code.

2.3.2 Phase 2: Understanding the Hardware.

There was also a direct requirement to read and understand the datasheets for all the hardware that was implemented. The stm32f4 data sheet is on its own has more than 1800 pages as well as numerous application notes that complement the datasheet. This data sheet together with some research and forums really puts into context the capabilities as well as issues other developers and designers have had with the stm32f4 development board.

All the other modules implemented in this design required a detailed familiarisation with their data sheets to understand the interfacing of these modules to the mother module especially with regard to the voltages and electromagnetic interference, especially important because of the some modules implemented have radio frequency implication. Electromagnetic Interference especially of the close proximity of these modules to each other. This design has not covered any documentation in terms of the electromagnetic interference but expect that cognisance should be taken of this influential factor with regard to the design and layout of components and modules specifically knowing that many modules of this design operates in the RF domain.

In this research we implemented various technologies all of which required some form of investigation and research all of which has been documented in depth in some cases and succinct in other cases.

2.3.3 Phase 3: Investigate Serial Communication interface and AT Commands

Development of communication interfaces to the various devices connected to the microprocessor also required some investigation. Investigating SPI, Uart as well as CAN as a communications interface required in-depth understanding of the protocol implementation as well as a low level understating especially CAN. To this end the investigation assisted in developing efficient software routines necessary to implement the design.

Because this design is using different communication interfaces operating simultaneously it was imperative to investigate underlying techniques in terms of buffering this data efficiently

without loss or latency. This design investigated different buffer techniques. This thesis also researched principles of Finite state machines, Interrupt handling techniques as well other software techniques in order to accomplish this design. It was also the purpose of this design to reduce data costs which was essential if we ever wanted to take this design to market. This design also looked at finite state machines in terms of monitoring the asset-tracking device in order to control the mode that the device will be in, in order to optimise data costs.

Research AT Commands:

In order implement these interfaces in this design it was required to research and implement the AT commands to access the various workings and data of these modules. The AT commands are specific to the individual modules and would have to be implemented differently and separately.

This module is an integral part of this project in order to implement perimeter based asset tracking.

Research and implement AT commands set for both OBD, GPRS and GPS interfaces

2.4 Other Designs that influenced this Design

The first type of asset tracking that was researched was the popular vehicle tracking and performance monitoring systems. (Jenkins, 2006) in his research demonstrated the core underlying technologies, illustrated in, position tracking, i.e., GPS, performance monitoring, i.e., OBD-II, and wireless communications, i.e., GPRS

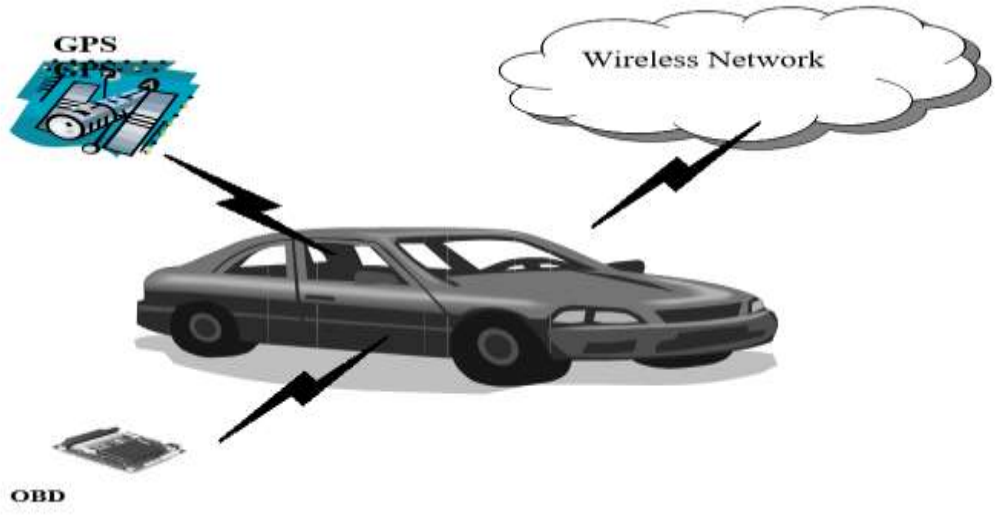


Figure 2.1 Vehicle tracking and Performance Monitoring

Jenkins in his design used GPS 35-PC receiver collected the recommended minimum data sentence (GPRMC) from the NMEA standard protocol. OBD-II data was gathered by a BR-3 interface. This interface incorporates a Microchip BR16F84-1.07 microcontroller, which operates on three SAE J1850 protocols i.e., VPW, PWM, and ISO-9141. (Jenkins, 2006)

Microsoft C# using Visual studio was used as the platform to write the data collection software. A tomcat web server, in conjunction with a MySQL database server, functioned as the gateway for users to view the location and performance data of each vehicle. The user interface was developed with JAVA SDK 1.4.2_05.

The data collection software combined GPS coordinates and OBD-II data into a single data stream that was sent to the server via the GSM/GPRS network. The data retrieved from the OBD-II system by continuous polling. Transmission of data to the server triggered by a received event from the GPS device connected to a serial port. This allows for a one-second minimum resolution. As it can be seen from the above approach of implementing the design or solution are all done on rapid development platforms or interfaces that are already implemented by some third party. This is all good and well if all that is to be achieved was a short-term solution to a business need but as for development and acquiring knowledge to implement new breakthrough designs this approach leaves one wanting.

Most of the implementation of the above design was implemented on High level application using high level programming languages; this research however have taken a step back and have explored development using the microcontrollers datasheet and writing routines from the datasheet using a low level programming language C. In addition, the other difference in this design is the microcontroller used. It is always the case that the microcontroller family that a designer was exposed to in their early years of development is the family of microcontrollers that they will go on to develop later in their careers purely based on familiarity with the said controller. Migration to other microcontrollers is only done if the features that developer is looking to implement is not present in their preferred choice but in the other.

The microcontrollers over the last decade has evolved quiet substantially and have capabilities of much more communication interfaces with advanced DMA capabilities this allows for interrupt or dma based communication interfaces. The other short coming of the design above is the method of polling used for the communication interface, very primitive as well as it ties up the processor putting restriction in the no of routines it can process.

(Bangali S.A & Shah S.K. Dr 2015) Implemented a vehicle tracking system for a school bus incorporating Biometrics, GPS, and GPRS also implemented on a Stm32f discovery board. The approach is fundamentally different as well as the objectives but a lot the work covered are very similar to what we want to implement. The biometric module implemented in this project is used for student identification via fingerprint identification and is verified by eeprom-

stored data. The GPS and GPRS Modules are used for vehicle location and internet connectivity.

The transmission protocol used for the transmission of data is HTTP. Their device is developed on a higher application layer than the device proposed in this design. There is also foreseeable high transmission cost with http as whether a device send 3bytes of data with http or whether it sends 80 bytes of data you are billed for the same with each http session. This design implements encapsulating data as well as using plain UDP for transmission thus reducing cost. The research also provides discussion of this in this dissertation.

The microcontrollers over the last decade has evolved quiet substantially and have capabilities of much more communication interfaces with advanced DMA capabilities this allows for interrupt or dma based communication interfaces. This thesis will explore and analyze the implementation of Serial communications interface using DMA and interrupt driven controller.

The design of Controller for multi-Layer parking controller based on the STM32 was researched based on the successful implementation of the CAN module as well as the use of the Stm32 arm microcontroller used. This design was based on the predecessor microchip the STM32f103 as opposed to the microcontroller used in this design but the baseline architecture is very similar and thus very useful information was derived from researching this design (Zhang et al. 2014).

The review of their design lead to understand the Power of using the DMA when implementing the ADC Module for this design as well as the architecture of the STM32 microcontroller in terms of using the can mailbox and the DMA without engaging the microprocessor thus being able to write macros to realize some of the control functionalities for the CAN Module.

Apart from researching other or similar thesis to this design it was important to research other theoretical and coding concepts in order to implement this design. Essentially that was the idea required to accomplish this design but after researching other thesis there was still unanswered questions and knowledge in terms of completing and implementing this design. Further researching of different technologies individually as well as software coding techniques had to be instituted to achieve a level of knowledge to implement the desired outcomes. Thus the research literature review was extended to the ensuing theories and technologies.

2.5 Direct Memory Access (DMA)

The design of the Asset tracking unit with GPRS relies on the proper use of the microcontrollers peripheral interfaces to get data from different modules and be able to store it in buffers for transmission to a control center and for selected local storage. To be able to achieve this it was necessary to understand and implement the proper use of DMA as well as Interrupts. It is with this in mind that the theory and implementation of DMA As well as Interrupts in this thesis was researched.

Direct memory Access negates the need to engage the processor for peripheral to memory, memory to memory and memory to peripheral transfer. DMA is a peripheral controller that controls the processor's bus directly. The DMA is an advanced high performance bus module. DMA only engages the processor for permission to use the Bus.

The STM32f4 DMA controller has numerous independently configurable channels that can perform autonomous transfers from peripheral to peripheral, peripheral to memory and memory to memory.

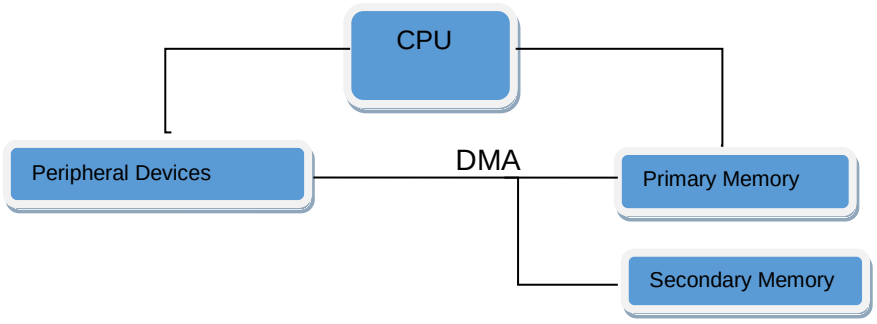


Figure 2.2 Showing DMA Transfer without engaging Central Processor

DMA enables the use of interrupts more efficiently as well increases data throughput and reduces the need for costly peripheral – specific FIFO Buffers it also reduces data latency.

2.5.1 Principle of Operation of DMA

In a routine DMA operation, some configurable event or flag or interrupt notifies the built-in DMA controller that data needs to be transferred from a peripheral to memory or vice versa.

The DMA controller then sends a DMA request signal/event to the CPU, requesting authority to use the bus. The CPU completes its current bus activity, stops driving the bus, replies with a DMA acknowledge signal to the DMA controller. The acknowledge signal is the signal that gives the DMA permission to use the bus.
The DMA controller then transfers one or more bytes, driving the preassigned address, data, and control signals without engaging the processor. When the process is completed, the DMA controller stops driving the bus dis-asserts the DMA request signal. The processor then resets its DMA acknowledge signal and then the bus control is handed back to the processor. In Embedded applications where the design requires minimal processor engagement DMA is the preferred method of transferring data from peripheral to memory and vice versa.

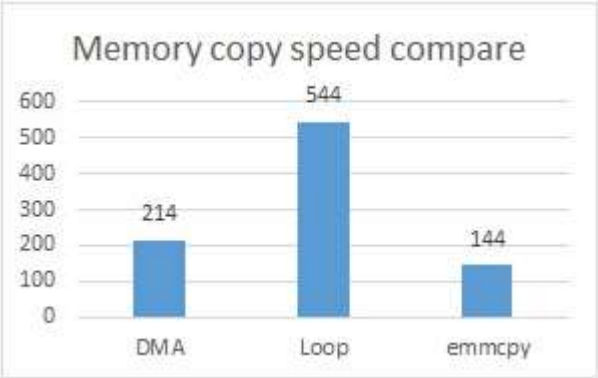
DMA controllers are typically initialized /configured by the software. Typical setup parameters include the length of the block, the base address of the destination area, the base address of the source area, which channel and stream to be used and whether the DMA controller should generate a processor interrupt once the block transfer is complete. See biometric module for detailed implantation of the DMA.

2.5.2 DMA: Burst or Single-cycle Mode

DMA operations can be performed in either burst or single-cycle mode. Some DMA controllers support both. In burst mode, the DMA controller keeps control of the bus until all the data buffered by the requesting device has been transferred to memory (or when the output device buffer is full, if writing to a peripheral). This is obviously not viable in Small active microcontroller designs that have many peripherals in use as this would tie up the bus. So in principle before one goes ahead and implement one or the other modes its best try to understand how the available bus will be affected. It is also advisable to kind a share bus usage when it come to the peripherals. In essence, what is being said that if when assigning a specific peripheral to a bus try and assign them according to bus usage if there is more than one busses like in the STM32f4 microcontroller.

In single-cycle mode, the DMA controller gives up the bus after each transfer. This minimizes the amount of time that the DMA controller keeps the processor off the memory bus, but it requires that the bus request/acknowledge sequence be performed for every transfer. This overhead can result in a drop in overall system throughput if a lot of data needs to be transferred. DMA isn’t great for very fast memory copies, the most defining benefit of DMA does occupy the main processor. DMA is certainly much faster than the conventional loop

method of transferring data but much slower memcpy function as can be seen in the graphic below:



In most designs, one would use single cycle mode if the system cannot tolerate more than a few cycles of added interrupt latency. Likewise, if the peripheral devices can buffer very large amounts of data, causing the DMA controller to tie up the bus for an excessive amount of time, single-cycle mode is preferable.

2.5.3 Setting up DMA Transfers on STM32f4

- need a “source”, a memory address when copying data from, a “destination”, i.e. where copying to.
- The size of the data chunks to be transferred. This needs to be done for both source and destination
- And some sort of signal telling the DMA controller when the next byte is ready to be transferred.

STM32f4 has 2 DMA controllers. Both Controllers are almost identical main difference being that DMA2 controller can only perform memory-to-memory data transfers. The other difference is that DMA2 is connected to APB2 while DMA1 is connected to APB1.Each DMA controller has separate streams. Each stream itself needs to be associated with one of eight channels.

These channels carry the signals to notify the DMA controller that data is ready to be transferred. Also known as DMA requests.

Which stream or channel to be used is determined by the following two tables?

Table 2.1 DMA1 Request Mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S2_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 2.2 DMA2 Request Mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2				DCMI
Channel 2	ADC3	ADC3				CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4			USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX				USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3			TIM8_CH4 TIM8_TRIG TIM8_COM

The above tables illustrate how the DMA streams are associated with which peripherals via which channel for example Using Tables above: To setup DMA for SPI_2Rx

Using DMA Controller 1; Stream 3 and DMA request signal Channel 0

For ADC1 we would be using DMA2, Stream 0 and Channel 0;

To complete the DMA setup one will need to configure the Microcontroller further. There are Hal Drivers that are available to set up the stm32f4 but in this design chose to use the standard peripheral libraries keeping in line with the bare bones approach.

SPI portion of the Biometric module Interface will be used to explain with a bit of code how to set the data transfer using dma;

For this one will need to Include two file i.e. the DMA.h and DMA.c files.

For the Biometric module, the STM32F4 controller is set up to receive data via the SPI2.

- The DMA request for the received buffer is associated with channel0 of stream3 of DMA 1
- The peripheral base address is a pointer to data register of the SPI2 module the biometric module is interface to the asset-tracking device via SPI2.
- The memory base address is the pointer to an array Bio_Buffer. This where the data for the biometric module that was acquired is stored.
- For DMA mode there are two options: normal and circular. In normal mode, the DMA stops transferring bytes after the specified number of data units, whereas in circular mode it simply returns to the initial pointer and keeps going.
- the peripheral address doesn't change, the pointer location into memory does get incremented thereby stepping through my "Bio_Buffer" array.
- the size of this array is exactly 6*84 bytes
- it is also imperative to specify a priority level for this DMA stream as in this design numerous peripherals are being used and relying on DMA to handle transfers on other peripherals as well.

Each DMA stream has its own small FIFO (first-in-first-out) buffer, which can temporarily store data from the source before transferring it to the destination. In addition, the transfer can happen in burst instead of single transfers.

So for a quick recap of DMA with the STM32F4. It can be an incredibly powerful feature that allows one to make the most of available computing power of the microcontroller. So as long as there is data that is being passed around the microcontroller from peripheral to memory or vice versa or from memory to memory, DMA is very useful but it if none of those requirements needs to be implemented then DMA will seem like a waste of time and resource. How useful it is will depend on the precise nature of the task to be accomplished. Key driving factor is that if we have Data that need to be passed or moved around use DMA it's the quickest most efficient way to do that without processor intervention. The DMA controller has definitely allowed embedded designers a channel to improve data latency issues as well as reduce processor overhead quiet incredibly making the 32-microcontroller very viable options in the IOT space.

2.6 INTERRUPTS

In this research project, it was necessary to understand and conceptualise the inner workings of interrupts. This understanding enabled the design to implement most of the modules in this thesis as well as put them together to function as one unit.

Interrupt is a powerful concept in embedded systems for control of how we want to execute the time-critical events and handle them in controlled and prioritized manner. In a typical embedded system, the processor (microcontroller) can only perform one task at a time but is responsible for doing more than one tasks often in time-constrained manner.

The arm microprocessor has Nested Vector Interrupt Controller(NVIC) that makes the task of handling/processing prioritising the different interrupts quiet an easy task that would have been quiet complicated if it was not for this feature of the microcontroller.

Interrupts are essentially kind of a hardware/software trigger that gets remembered as a flag that can be called to get the micro to perform a routine and return to its main function or its last operation before it was called.

Interrupts are an essential feature of a microcontroller that enables the software to respond in a controlled manner to internal/external hardware and software events. Interrupts were used extensively in this thesis for example in the GPS module for the transmission and reception of serial data via the uart.

Using interrupts requires that one understands how the central processor processes an interrupt so that we write code to handle them effectively.

2.6.1 Exceptions

Exceptions are events that occur that intentionally or intentionally cause changes to the program routine. When an exception occurs the central processor suspends executing the current task and executes part of the program called the exception handler. When exception handler completed, the processor returns to the normal operation. An interrupt is a type of exception.

2.6.2 Using Interrupts in Arm Microprocessor

In order to enable a specific interrupt separate bit or flag needs to be, e.g. RIE (The UART Receive Interrupt Enable bit). The hardware sets the flag bit when it wants to receive an interrupt. The software must clear the flag bit when it has handled the interrupt and allow the device to trigger an interrupt.

There are number of special registers in the microcontroller processor that contain the processor status and defines the operation states and interrupt/exception masking. These special registers are not memory mapped. There are specially defined registers that enables or disable all armed interrupts according to the PRIMASK bit (visualgdb.com n.d).

Interrupt Polling

Some interrupts share the same interrupt vector like the receive and transmit routines of the UART. In this case, the software routine i.e. The ISR is responsible for polling the status flags to see which event has triggered the interrupt. Handling interrupt routines can very intricate, very close attention should be payed to the data sheet and app notes that are available.

2.6.3 Nested Vector Interrupt Controller

All Arm Cortex processor provide a Nested Vectored Interrupt Controller (NVIC) for handling interrupts. The NVIC receives interrupts and exception requests from various sources. In order to set up interrupts effectively proper understanding of the datasheet and application notes is necessary. But having said that it quiet self- explanatory and easy to implement because the nested vector controller takes care of most arbitration and conflicts of the different interrupts. What needs to be properly implemented is the priorities of the IRQs.

The NVIC forms integral part of all Cortex-M processors and provides the processors' outstanding interrupt handling abilities. NVIC is integral in this this design because of the number of peripherals are interfaced to the microcontroller. NVIC supports numerous interrupts up to 32 IRQ in some ARM devices (www.st.com n.d). Depicted below is a Visual Architecture of the NVIC in relation to other core parts of the Stm32f microcontroller.

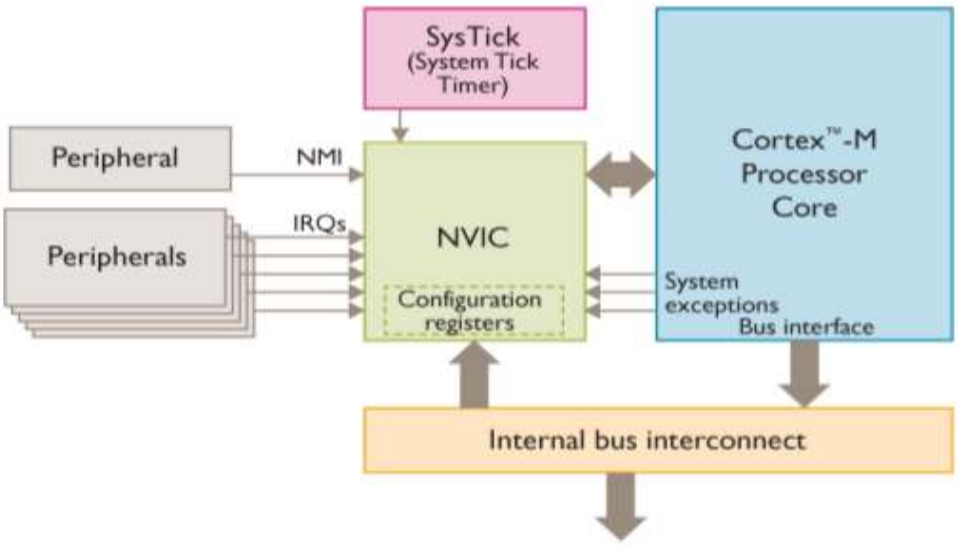


Figure 2.3 Architecture of NVIC

Most of the NVIC settings are programmable. The configuration registers are part of the memory map and can be accessed as C pointers. There are also numerous libraries including the standard peripheral libraries from ST Microelectronics that are available to be used when setting up the NVIC. Using the standard peripheral libraries of the Stm 32f4 device makes configuring the Asset tracking device quiet simple using *Structs* types in the c language. So all we do is use an instance of the structure that is defined in the standard peripheral library obviously after including the necessary file in the start-up code. Inside the NVIC, each interrupt source is assigned an interrupt priority. A few of the system exceptions like such as NMI haves a fixed priority level, and others have programmable priority levels. By assigning different priorities to each interrupt, the NVIC can support Nested Interrupts automatically without any software intervention.

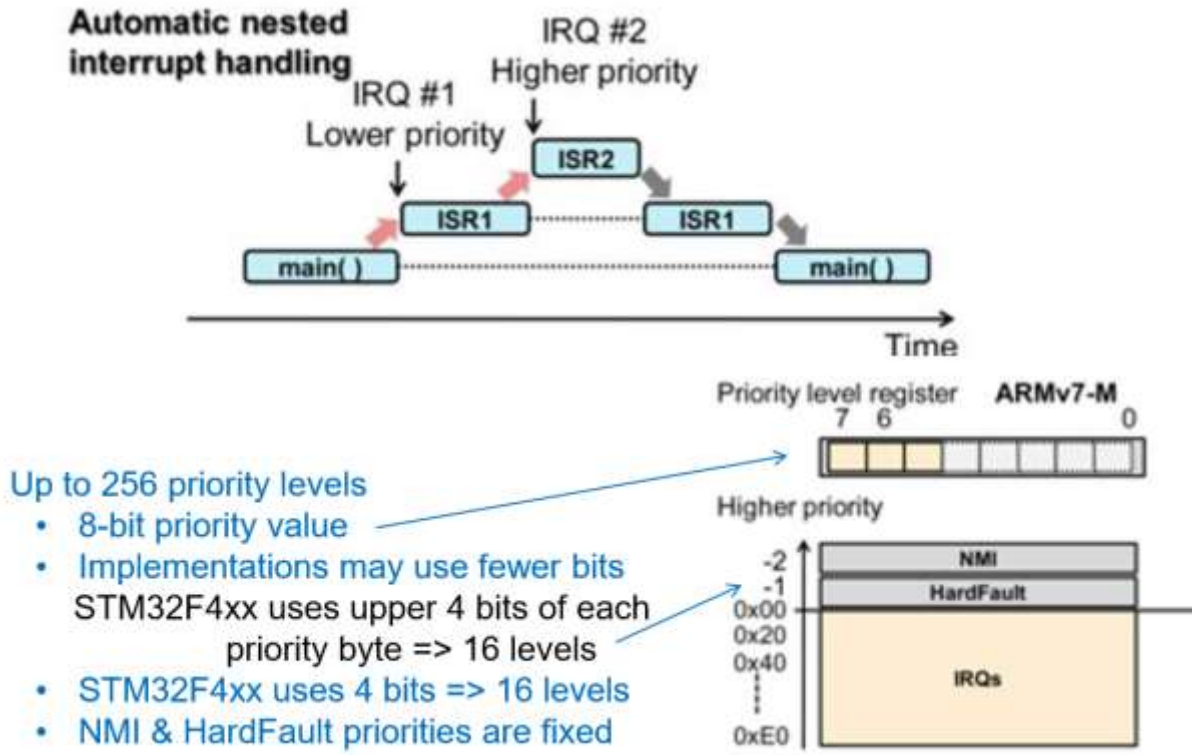


Figure 2.4 Graphical explanation of handling of Priority Levels in STM32 Controller

The architecture provides 8-bits of priority level settings for each programmable interrupt or exception.

The Stm32f4 uses a stack based exception model which essentially means that when an event and or exception occurs that a number of registers are pushed onto the Stack. These

Register are restored to their original values after the exception handler finishes. This feature allows exception to be written as normal c functions, and reduce processor overhead in terms of interrupt processing.

In addition, the Cortex-M processors use a vector table that contains the address of the function to be executed for each particular interrupt handler. On accepting an interrupt, the processor fetches the address from the vector table. Again, this avoids software overhead and reduces interrupt latency.

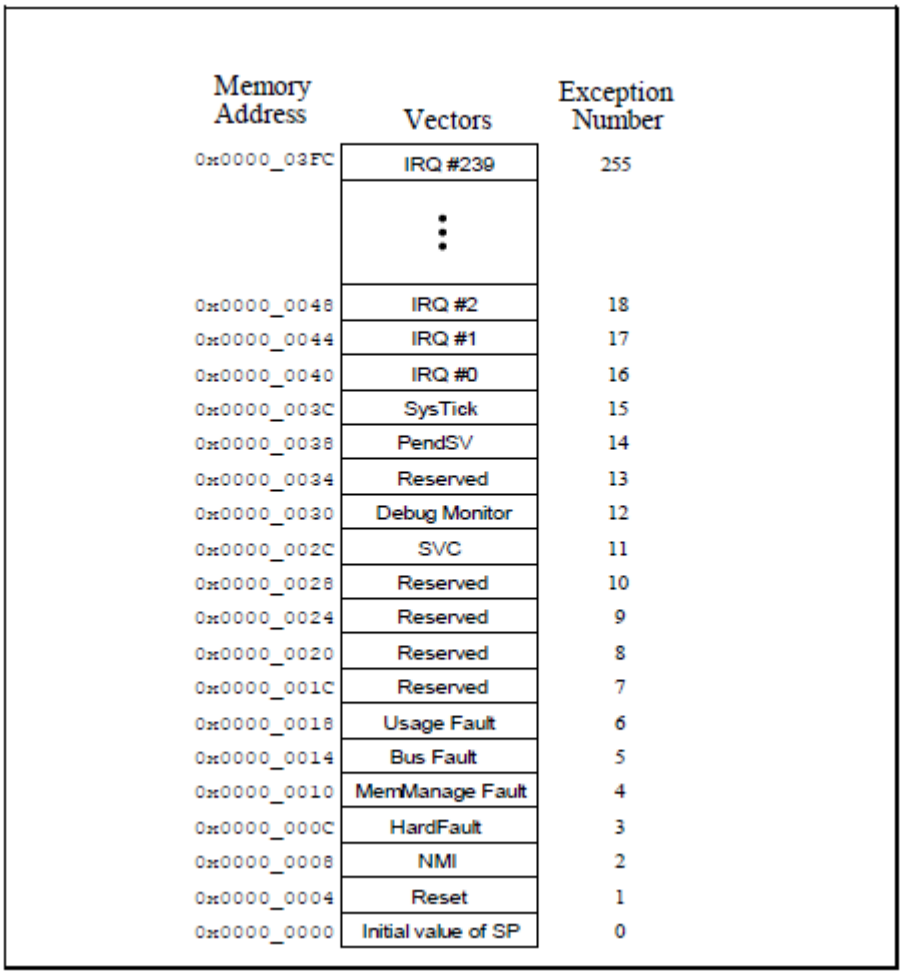


Figure 2.5 Example of Vector Table

Various optimization techniques are also used in the Cortex-M processor implementations to make interrupt processing more efficient and make the system more responsive:

Tail chaining – can be explained simply if there is another exception pending when an ISR exits, the processor does not restore all saved registers from the stack and instead moves on to the next ISR. This reduces the latency when switching from one exception handler to another. In a high speed data acquisition environment this can be a very useful feature of the NVIC it allows processes to seem real time.

• NVIC can halt stacking (and remember its place) if a new IRQ is received.

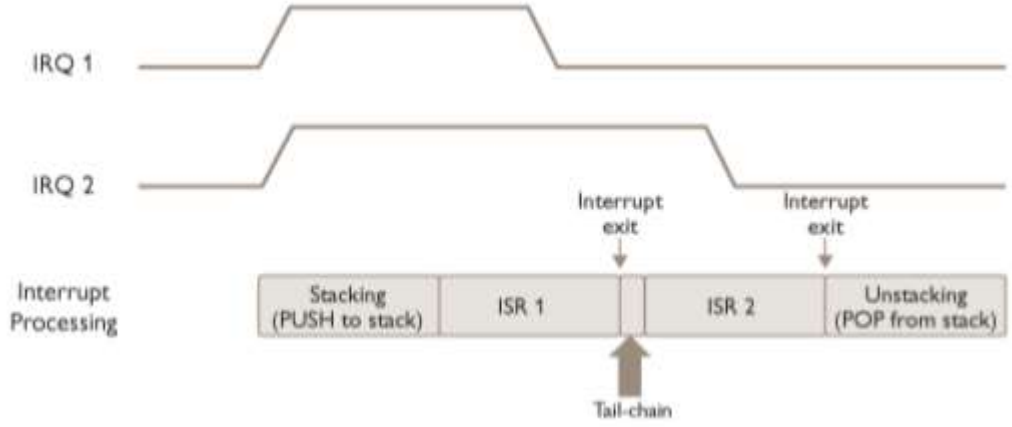


Figure 2.6 Timing Diagram Illustrating of the principle of Tail-chaining

Stack pop pre-emption –Another optimisation technique used in interrupt handing of the NVIC is Stack pop pre-emption.If during the andling of an event another exception occurs during the unstacking process of an exception, the processor abandons the stack Pop and services the new interrupt immediately. By pre-empting and switching to the second interrupt without completing the state restore and save, the NVIC achieves lower latency in a deterministic manner which simply means the NVIC takes controll of the time taken to handle and optimise both interrupts. (Github.com, n.d)

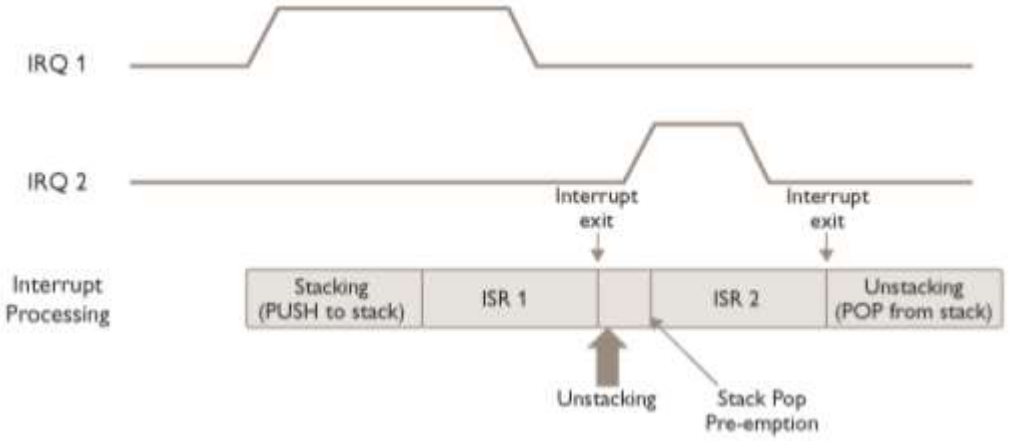


Figure 2.7 Timing Diagram Illustrating of the principle of Stack-Pop Pre-Emption

In the Arm cortex microprocessors there are 2 different kinds of priorities: this feature handles arbitration of interrupts and can be very useful when tasks have to be executed with very low latency; the two of priorities is preemption priorities and sub priorities

Events that have the same preemption priority will look to the sub priorities for arbitration decision which means then that one with the high sub priority will be executed first. If they then have the same sub priority then the first come first served rule will be used as the arbitration decision. The default rule is that the event with highest preemption priority will always be executed first. This arbitration capability of the arm processor is fundamental in this design because of the number of different peripherals and events there is for processing data. It is important that there is no ambiguity in terms of priority because that could lead to increased latency or complete failure with regard to handling the interrupt.

Previous generation 8- bit microcontrollers did not have Nested Vector Interrupt controller which are now introduced in the Arm micro-processer. This innovated feature has gained the respect and admiration of main embedded designers and makes handling multiple peripherals a breeze. In the past years it was always a challenge handling a large number of peripherals and processes asynchronously, there were arbitration techniques that had to be built into the code to handle conflict of two or more events. NVIC allows the code to much more efficient as there is a dedicated controller handling the interrupts. The processor can now be programmed to handle numerous and cumbersome tasks as the NVIC if programmed properly. It is however a steep learning curve that is well worth the effort of learning because of the numerous benefits it bring to an embedded designer.

2.7 Serial Peripheral Interface Protocol (SPI)

The Serial peripheral interface is an internal interface to the microcontroller that allows it to communicate with external devices.

A detailed a thorough understanding of the SPI protocol was necessary for this thesis; it was essential for the implementation of the biometric module. The grasp of the SPI concept is essential for any embedded systems designer. The SPI communication stands for serial peripheral interface communication protocol, which was developed by the Motorola in 1972. SPI is a hierarchical synchronous communication full duplex protocol amongst electronic devices which means that carries data signals in both directions. It also means that it uses separate lines for clock and separate lines for data so as keep both sides in synchrony.

There are four key pins that make up the SPI interface namely

– Clock, Miso, Mosi and chip selector – it manages to transfer data between two or more devices. As part of an SPI setup there is a master (usually a micro controller) and one or more slaves.

The Clock pin serves as a coordination mechanism; it tells the devices when to read/write data.

Chip Selector is used to inform the slave device that we want to exchange data, this is done by merely pulling the CS pin down. In a multi slave set up the devices share the same clock, MISO and MOSI lines but have their own independent chip selector.

Mosi- As the Name suggest is Master in Slave Out (output pin)

Miso- Is Master In Slave out (Input pin)

SPI protocol uses four wires named as MISO, MOSI, CLK, SS used for master/slave communication. The master is predominantly a microcontroller, and normally the slaves are other peripherals like sensors, GSM modem and GPS modem or like in this design a module like the nRf2401l module, etc. Multiple slaves are interfaced to the master through a SPI serial bus. The SPI protocol does not support the multi-master communication, and it is used for a short distance within a circuit board (Marwedel, 2010). There are four i/o signals associated with the SPI peripheral

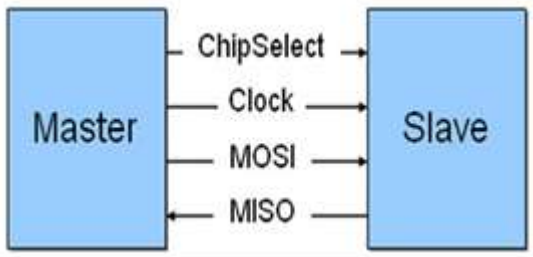


Figure 2.8 Serial Peripheral Interface

MISO (Master in Slave out): The MISO line is configured as the input in a master device and as an output in a slave device. Used to receive data from a slave device.

MOSI (Master out Slave in): The MOSI is a line configured as the output in a master device and as an input in a slave device wherein it is used to synchronize the data movement. This terminal is associated with receiving data from the master device.

SCK (Serial Clock): This signal is always driven by the master for synchronous data transfer between the master and the slave. It is used to synchronize the data movement both in and out through the MOSI and MISO lines.

SS (Slave Select) and CS (Chip Select): This signal is driven by the master to select individual slaves/Peripheral devices. It is an input line used to select the slave devices. Master Slave Communication with SPI Serial Bus. When this line is active then the SPI communication interface between the two devices is enabled and data exchange can begin.

The SPI protocol is implemented quiet extensively in the thesis and deserved a mention as it is the backbone of successful implementation of the Biometric Module. The STM32f4 serial peripheral interface offers various operating modes which makes it a high configurable interface which in turns requires a detail knowledge of the data sheet to implement correctly. It is a very useful interface and is used in many hardware interfaces like display panels, data acquisition chips to name a few.

2.8 Analogue to Digital Convertor (ADC)

An ADC is a peripheral that allows measuring the voltage (between 0 and V_{ref}) on a certain configured input of the microcontroller. The basic principle of operation is that peripheral sample the digital input or value of the signal received at specific peripheral stores an array in the order it was received order for further processing with regard to voltage reference signal. The resolution of the analogue signals is the number of samples taken per cycle. ADC resolution is one of the key factors to determine how precise the conversion can achieve. If a chip has a resolution of 8-bit (0-255), meaning that it can detect 256 different levels of input analog signal. The stm32f4 microcontroller is being used has a 12-bit ADC, which increases the resolution of analog conversion to 4096 steps (from 0 to 4095). This resolution is configurable to 12-bit, 10-bit, 8-bit or 6-bit where faster conversion times can be obtained by lowering the resolution.

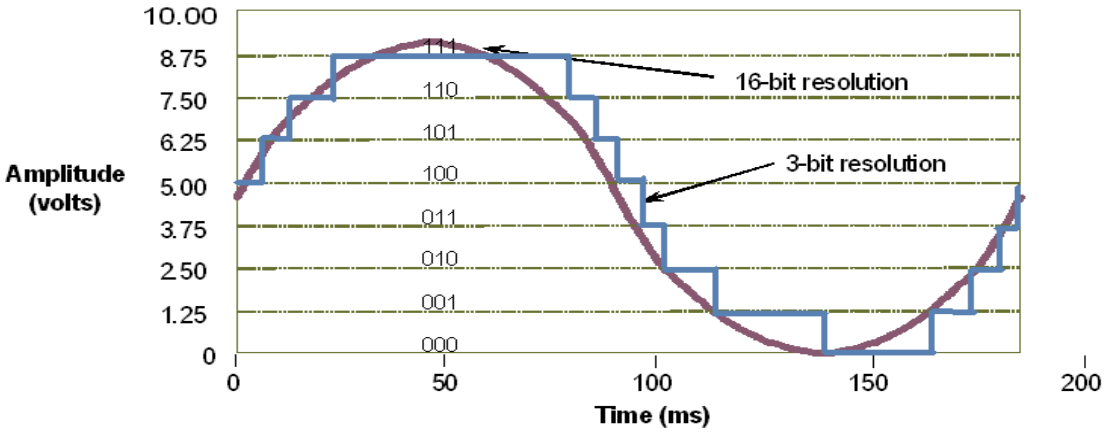


Figure 2.9 ADC showing 3-Bit resolution and 16-bit resolution Signal [30]

The STM32F4 has a built-in Analogue to Digital converter.

2.8.1 The Implementation of the Multi-channel ADC Module

The basic principle of operation of the multichannel ADC Module is;

- Firstly: required 5 unused GPIO ports to be configured as analogue inputs. These ports will be reading the sensors that attached to these ports referenced to 3.3 Volts. Most sensors on motor vehicles are normally 5v sensors which means the device will need to do some level shifting.

- The ADC module on the microcontroller has to be configured for multichannel processing. See appendix A.
- Seeing that device will be using DMA to transfer the data from the ADC peripheral to the ADC Buffer. This will also be set up.. In this design the peripheral_base_address of the ADC is the Data register of the ADC the device is using. The memory address will be the array that is created to store the respective converted ADC values ConvertedAdcValue[].This array will be volatile as the values can change any given time and outside the code.
- So each index of the array will store one sensor value, the value of the sensor that the GPIO is reading. Each index of the ConvertedAdcValue will be copied to a static array and each index of that array will be stored as global variables for use in the code.
- E.g. int Vspd=ConvertedAdcValue[0];The static buffer created will be used in the serialization routine for transmission to the remote control centre. The variables created from this array will be used by the GSM state machine to control the various modes of transmission.
- The ADC Module will also responsible for determining the engine running hours that is necessary for asset management. The Ignition of the vehicle and Engine speed are the two parameters used to start the Running_Hours function which is a counter:

  ```
  if(IGN>0&&Rpm>0)
  {
  Running_Hours();
  }
  ```

2.9 OBD Module

2.9.1 Overview

The fundamental purpose of this design is to acquire data from a specific asset and present it in a desired format to remote collection service for storage as well as analytics. The asset in question is a modern motor vehicle.

Motor vehicles are no longer just a mechanical apparatus they have numerous electronic control units networked together.

Since 1996, most vehicles have been required by law to monitor their own emissions performance and to report on it through an On-Board Diagnostics port. This has lent us the

opportunity to able to retrieve this data and possibly analyse it but also to make decisions based on this data in terms of asset lifecycle management as well as asset optimisation. Before 2008, several different protocols were used as base protocol for transferring OBD data. All vehicles after 2008 have to be CAN compliant. Today's vehicles have many built- in computer systems that control parts of the car such as fuel injections, airbags or brakes. All of these systems are controlled by one of several Electronic Control Units (ECU), which communicate with each other over the internal high speed Controller Area Network (CAN) of the car.

In this chapter, we focus on the implementation of the OBD module on its own and it the next Chapter the CAN module on its own. The reason being their functional implementation in this thesis is different for the purpose of asset tracking and also they are fundamentally different technologies. As many of the vehicles that we are interested in monitoring will be vehicles prior to 2008 the OBD module is and essential module for the asset tracking device.

5.2 The OBD II Standard

The On-Board Diagnostics (OBD) can discover and diagnose problems with the data reported by the ECUs. If a problem occurs, the OBD system generates a trouble code that which makes it possible for a service engineer to identify and fix the problem. Trouble codes and other diagnostic information can be accessed by plugging an OBD scan tool into the OBD interface in the car. The main advantage with having an OBD system is that it makes it easier to diagnose faults that occur in the vehicle. An advantage from a sustainability point of view is that vehicle emissions can be reduced by discovering and fixing problems, that makes the vehicles emission levels rise, that otherwise might not have shown any noticeable symptoms to the user or service personnel. This design however will not be using the full functionality of the OBD system just the monitoring capability as the main purpose is to retrieve data that can impact asset lifecycle management. There is clearly numerous application for using the fault codes and engine management diagnostics capability of the OBD module but will not implemented in this design as this design is focussed on the tracking of specific asset data.

There are numerous end-user products for extracting OBD trouble codes from cars but these products are stand-alone with little to no user-based content around the faults and or monitoring. For the asset-tracking device, this will be monitoring pre-2008 vehicles and in some case specialised vehicles pre 1996.

On Board Diagnostics OBD is the computer system built into cars that monitors the performance of the engine components as well as other control units. It consists of several ECUs that uses various sensors to collect data and evaluate the performance of the car. The OBD module in this design will monitor various predefined data or PID's of the vehicle and

relay it to the control centre. There is how ever very few or no aftermarket implementation of remote monitoring real time obd data to a control centre

There is no available implementation of an OBD software library that can freely be used on an embedded system to communicate with the OBD system in a car and extract information.

The objective of this module is to develop a system that can monitor data present by the OBD system and make the information accessible to the asset manager.

OBD2 permits five different communication protocols, as listed in table 3-1, which can be used to communicate with the OBD2 interface. Most vehicle manufacturers only implement one of these protocols so it is often possible to identify the used communication protocol by looking at which pins are present on the connector.

Standard Description

SAE J1850 Pulse-Width Modulation (PWM)

SAE J1850 Variable Pulse Width (VPW)

ISO 9141-2 Similar to RS232

ISO 14230 Keyword Protocol 2000 (KWP2000)

ISO 15765 CAN (250kbps or 500kbps)

2.9.2 Research the OBD protocol

The main purpose of this design and thesis is to design an Asset Tracking device that will relay data to a control center. The asset in question is a modern motor vehicle. The modern motor vehicle is equipped with an obd network.

This requirement will be acquired through researching the above topic with the understanding of how the OBD protocol operates so that the developer would be able to translate this operation into C-code routines in order for the microcontroller to parse relevant or requested OBD data to the control centre. This design also researched different obd interpreter chips and hardware. The two chips that has chosen to research is ELM327 chip and the st1110. The most convenient chip to use based on cost and availability when we build the OBD interpreter with serial-interface. For any end-user software/interface to communicate with the OBD port of the vehicle it is necessary for an obd interpreter IC between the vehicle and the serial interface. There are numerous options of this type of circuitry available but the most popular and reliable interfaces are the Elm327/329 and the ST1110 obd interpreter IC. There are numerous variations of these IC's. During the research, these two chips were investigated in order to get a feel of their capabilities to understand their capabilities and their cost and availability to the South African market. When this design goes into production these will be key product driving factors from both cost and delivery perspective.

The table below demonstrates the differences between the two OBD interpreter chips. The st1110 is the by far the product of choice but for the purpose of this thesis the Elm327 chip was used purely because of availability to build and test this module for completion of this thesis.

Table 2.3 Detail Comparison of the OBD Interpreters

	ELM327 v1.4	STN1110
Base microcontroller	PIC18F2580	PIC24HJ128GP502
Architecture	8-bit	16-bit
Processing speed	4 MIPS	40 MIPS
Flash (ROM)	32 KB	128 KB
RAM	1.5 KB	8 KB
Pin count	28	28
Available packages	PDIP, SOIC	PDIP, SOIC, QFN
Supply voltage range	4.5 to 5.5V	3.0 to 3.6V[1]
Supports all OBD-II protocols	yes	yes
ELM327 command set	yes	yes
Enhanced "ST" command set	no	yes
Firmware upgradeable	no	yes
Large OBD message memory buffer	no	yes
Low power mode	yes	yes
Supported UART baud rates	9600 bps to 500 kbps	38 bps to 10 Mbps
OBD message filtering	basic	advanced
Price each, for 1000 units	$24	$10
Price each, high volume	$19	$4.95

.

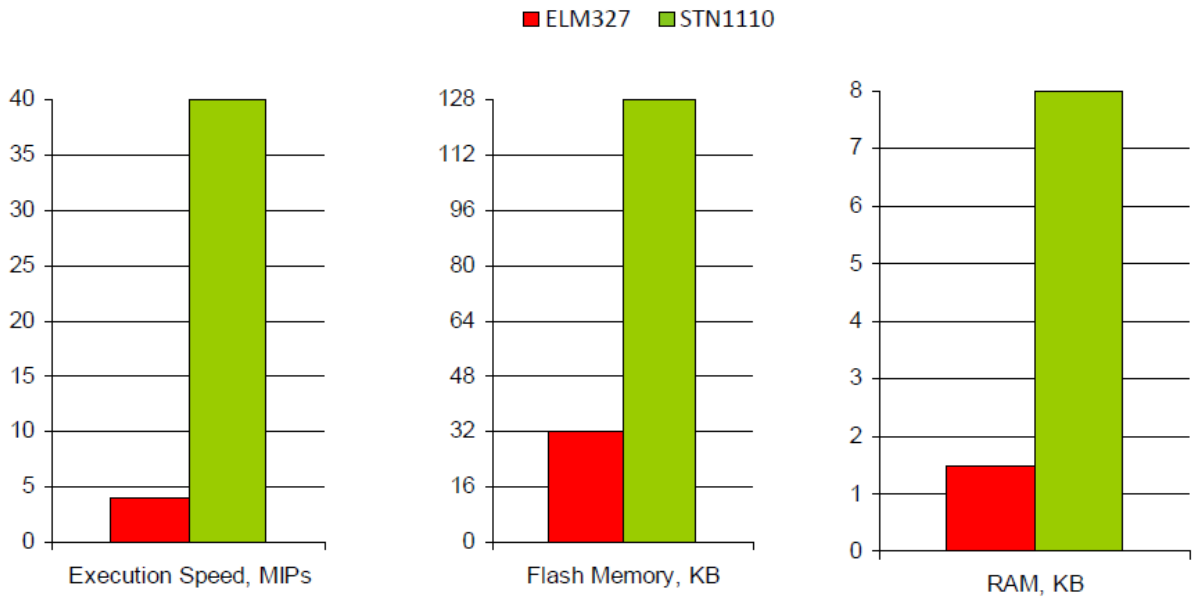


Figure 2.10 Graphical Comparison of Performance factors

As can see from the above graphs the ST1110 can execute software routines 10 times faster than the ELM 327,has four times more flash memory, has five times more RAM and yet it is far cheaper than its counterpart. It also has a significantly large OBD buffer that is very important for this design especially of the fact that this design will be operating the obd interpretor in monitoring mode.

The ELM 327 is a device that can data translate data that can be used by computers, smart phones and from the ISO 15765-4 interface into serial data form that can be directly translated into data that can be used by computers, smart phones and other devices.

The elm327 expects to communicate with any device be a computer a smart phone or microcontroller through a rs232 serial connection. In this research project we have set up a Uart connection to the microcontroller for the purpose of communicating with the vehicle.

Most diagnostic products are based on the ELM327/ELM329 by ELM electronics or the ST1110 microcontroller which supports all the different OBD communication protocols. The microcontroller presents a simple serial interface to the OBD with which an application can communicate using the standardized ELM327 command protocol.

2.10 Controller area Network (CAN) Module

With rapid evolution of the motor vehicle as well as the coherent evolution of information technology and microprocessors we find that these enhancements finding their way into the motor vehicle.

The modern motor vehicle has placed demands on type of microcontrollers and type of network used. Vehicles in this era may have more than fifty electronic control units communicating over different networks to manage vehicle functions ranging from essential to cosmetic functions. The essential systems run on high speed networks whereas the cosmetic systems run on lower speed networks. Safety systems such as airbags employ dedicated high speed network communication, as does the powertrain control for communication between the engine and transmission controllers. Figure 10.1 below illustrates how a typical network arrangement would look like on a modern motor vehicle.

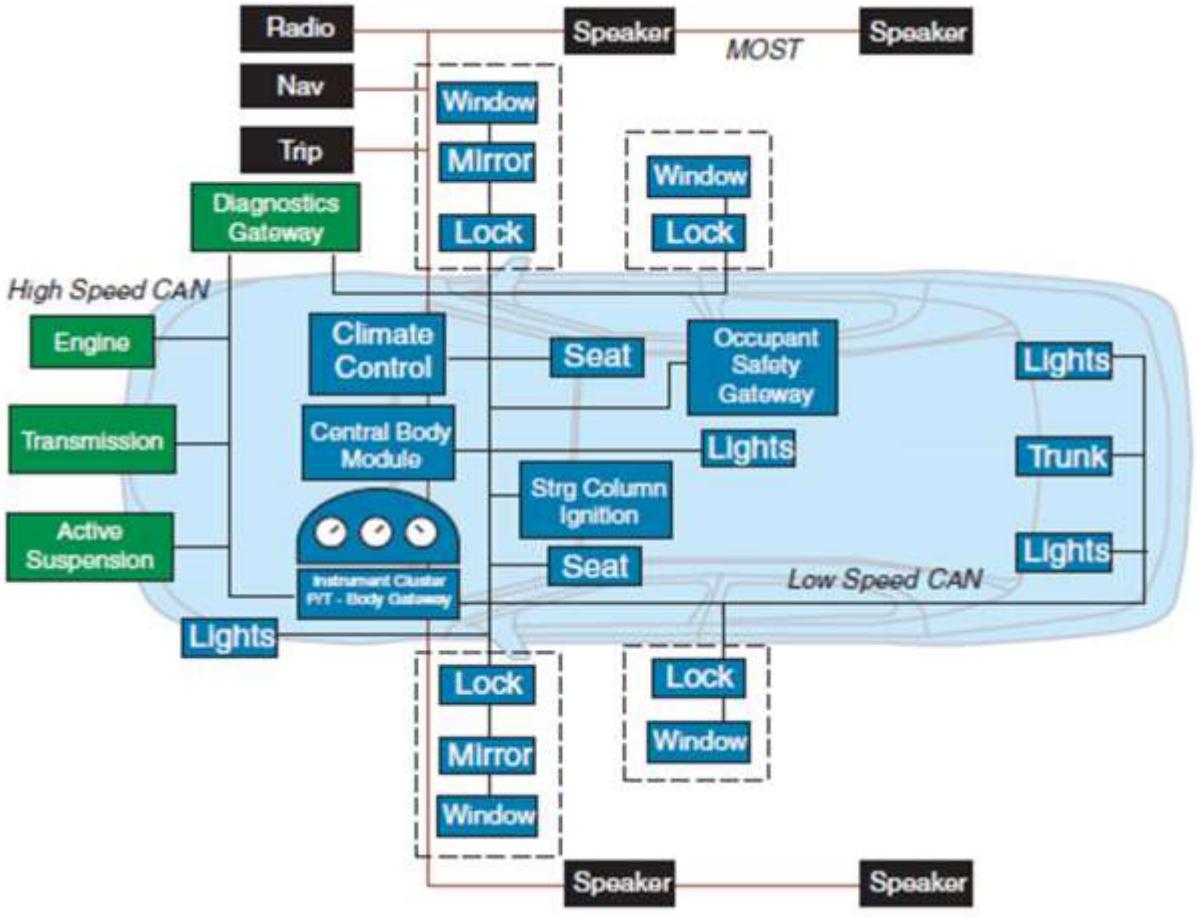


Figure 2.11 Illustration of a Typical CAN Network in a Motor Vehicle

In recent years the automotive industry has introduced numerous safety and cosmetic features that created the need for high speed networks for the transmission of real time data

between processes. Initially various multiplexing systems were introduced which decreased the number of interconnections there was between the electronic control units but as the number of control units increased, the industry sat with the same problem that was originally there. Multiplexing became very cumbersome and there were various data latency issues so technically the data was not real time as require by some of the systems.. Robert BOSCH Gmbh corporation designed a multi master serial communication protocol called Controller Area Network (CAN) protocol for robust and real time for in-vehicle networking.
With the implementation of CAN in the modern automobile; there were great optimisations made in terms of the number of interconnections made as well as the quantity of copper used per vehicle.

Typically ecu's are networked together on one or more buses based on a Controller Area Network (CAN).These different ecu's communicate with each other by sending CAN packets, these packets are transmitted to all ecu's on the network. These ecus are differentiated from each other by their own unique address or identifier. Because of this, it easy for a foreign ecu to masquerade on the network if it assumes the identity of one of the ecu's

ECU's are embedded devices with sensors and actuators attached to them; this gives us the ability to introduce controllability from the control centre to the ecu. So from data perspective all that is needed to control any ecu on the network is to know the address/identifier of that ecu as well as the instruction data that is encapsulated in the can packet.

Sketching the basic principle of "hacking" the CAN Network at application layer, CAN packets contain an identifier and data. Use of the word "hacking" is subjective as this is owners device and owners assets but many manufacturers do everything in their power to restrict outside or after- market piggy backing. The embedded designer however needs to translate the data
Received from the propriety standard of the specific manufacture if he needs to use it to accomplish and interpret data for what it was originally intend to do.
CAN can be implemented on different layers of the OSI model. Most user applications implement CAN at higher level whereas automobile manufactures implement CAN at a lower level. Most "hacking" takes place at lower level. In order to introduce controllability to the module it is required to implement CAN at a low level where we intercept CAN messages/data to implement macros to carry out the necessary output tasks via the GPIO interfaced power electronics interface module or via the engine control unit of the motor vehicle or any of the other control units on the CAN network.

2.10.1 Characteristics of CAN protocol

The main characteristics of CAN protocol are

- Multi master hierarchy
- Priority based bus access
- Baud rate up to 1MBits/sec
- Error detection and fault confinement

A CAN bus is a half-duplex, two wire differential bus.

When referring to the CAN bus lines one refers to either the CAN High or CAN Low Lines. The two lines, CAN_L and CAN_H, form the communication bus for the nodes to transmit data or information. (Murphy, 1997)

The logic levels of a CAN Bus is determined by the difference of these two bus lines. Differences of greater than 2 Volts are treated as dominant level and if it O volts then considered recessive

In the CAN protocol, nodes communicate data or information through messages termed as frames. A frame is transmitted on to the bus only when the bus is in idle state. There are four different types of frames which are used for communication over CAN bus.

- Data Frame – Used to send and frame containing node data for transmission
- Remote Frame – Used to request data of a specific identifier
- Error Frame – Used to report an error condition. Transmitted by any node detecting an error
- Overload Frame – Used to inject a delay between two data or remote frames.

CAN is a broadcast type protocol so when a node transmits a frame all nodes on the network will receive it.(Barrenscheen J, Dr.1998)

The CAN controller hardware provides a message filtering that controls whether the received frame is relevant to that node or not.

A maximum of 8 bytes can be attached to each identifier. A standard CAN identifier is 11 bits long. Eleven bits gives a possible 2048 different identifiers although in practice few vehicle manufacturers use more than about 20. Because a CAN frame can have up to 8 data bytes, it is common that a number of signals are attached to each identifier. An example CAN frame is shown below.

Identifier	Data Bytes							
	1	2	3	4	5	6	7	8

Figure 2.12 Frame structure of CAN ID field with data.

CAN protocol is a standard that is applied by all manufacturers when it comes to the physical aspect of CAN but the way it is applied in terms of data and identifiers differ drastically from manufacturer to manufacturer. There is no prescribed standard into how it is implemented.

This essentially means that a specific identifier on let's say Ford motor vehicle contains the rpm signal in a data frame it doesn't mean the Toyota vehicle with the same identifier will carry the rpm data in that data frame. A typical CAN frame on a vehicle might look like this:-

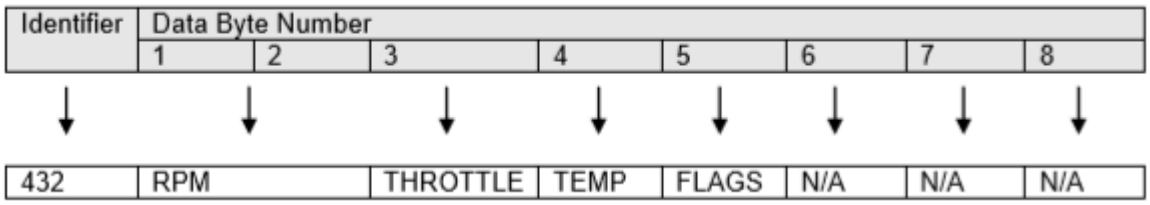


Figure 2.13 Example of data in data fame

The above is a data frame with an identifier and eight bytes of data; such a frame may be transmitted by the engine control periodically. In this data frame the first 2 data bytes are used to represent RPM as a 16bit number. The 3rd byte contains an 8-bit value of the throttle position sensor. The fourth byte is an 8-bit value, engine coolant temperature. The 5th byte contains 8-bit representation of a status of switches that the ecu could be monitoring and bytes 6,7 and 8 are not used. If a control unit for example the instrument panel needs to know engine coolant temperature then it only needs to listen out for a message with an identifier of 432 and extract the fourth byte. If a control unit need to know engine rpm like the GSM state machine it only need to look out for a data frame with identifier 432 and extract the first two bytes.

2.10.2 Structure of a node in CAN network

The CAN controller implements only three layer of the ISO/OSI Reference model

in a node. The physical layer and data link layer are integrated on the Can controller chips and the libraries for the connection between the data link layer and the application layer are provided by the CAN chip manufacturers.

CAN provides four different types of message frames for communication, the breakdown of each frame is follows.

2.10.2.1 Data and Remote Frame

So the important thing to know about the architecture of the data and the remote frame are that they are exactly the same. The RTR Field of the frame determines if the frame is remote frame or a data frame. If RTR is set to dominant level then that specific frame will act as Data

frame. Data frame is used transmit data to another node or nodes on the network. Each frame can only transmit a maximum of eight bytes.

If however if the RTR field is recessive then that specific frame will act as a remote frame. The duty of remote frame is to request data from any node on the network.

Each data frame will be assigned a unique message id.(Bosch 1991)

When a remote frame is accepted by the relevant node, a data frame will be generated by that node with the requested data and put on to the bus.

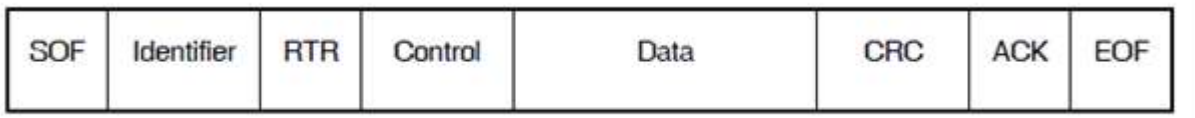


FIGURE 2.14 : Architecture of Data and Remote frame.

The following are the fields in data and remote frame:

• SOF field (1 bit) – means START Of Frame which is pretty self-explanatory. A single dominant bit indicates start of frame. also used for bus synchronisation

.• Arbitration Field – made up of two daughter fields, Message Identifier and RTR

field.

• Message Identifier (11/29 bits) – Also known as the address field. This field contains a message ID for each frame which is either 11 (standard ID) or 29 bits. This field is used for arbitration as the lower the address or value of this field the higher the priority on bus probably why it is known as the arbitration field.

• Remote Transmission Request (RTR) (1 bit) – The RTR field

Determines whether it is a data frame or a remote frame.

• Control Field: made up of 6 bits – Also made up of two daughter fields, IDE and DLC field

• Identifier Extension (IDE) Bit (1 bit) – This bit determines the format of the

message ID in the frame, either a standard 11-bit format or extended 29-

bit format.

• Data Length Code field (4 bits) – In a data frame This field is used to controls or sets the amount of data being transferred . In a remote

frame, the value in this field represent the amount of data it is requesting. The

following are the values of the DLC bits for the corresponding amount of

data.

• Data Field – This field contains the actual data and it is not applicable for remote

frame.

• CRC field (16 bits) – The Cyclic Redundancy Check field consists of the CRC Sequence and a CRC

Delimiter bit.

A node uses data frame to transmit data to any other node on the network. The

RTR field determines whether the message frame should act as data frame or a remote frame. When the RTR bit is set to dominant level, then the message frame will act as a data frame. A maximum of 8 bytes of data can be transferred using a single data frame. Each data frame will be assigned a unique message ID using which the node decides whether the data is relevant or not.

A remote frame is used to request a data frame from any node on the network.

When the RTR bit is set to recessive level, then the message frame will act as a remote frame. While requesting data from a node, the length of the data field in control field (DLC bits) of the remote frame should be same as the requesting data frame otherwise a bus collision occurs. As soon as the remote frame is accepted by a node, a data frame will be transmitted on to the bus with the requested data. When two or more nodes on the network request the same message at the same time a bus collision occurs.

2.10.3 Bus Arbitration

In a single bus communication protocol like CAN any node may access the bus at any given time, if the bus is idle. (Tindell et al, 1997) If more than one nodes wants to gain access to the bus at the same time then the node /message with the highest priority i.e. the lowest

address(CAN identifier),the CAN Identifier of zero will have the highest priority on the CAN bus.

When a remote frame and a data frame that has the same CAN identifier competes for the bus the Data frame wins due to the additional RTR bit which belongs to the arbitration field of the frame. The bus arbitration technique of CAN also influences in the reduction of data collisions. CAN protocol provides a non-destructive bus arbitration mechanism.

2.10.4 Message Broadcasting

CAN protocol is based on message broadcasting mechanism, in which the frames

Transmitted from one node is received by every other node on the network. The receiving nodes will only respond to the data if it is addressed to them. (Zuberi & Shin, 1997)The CAN protocol has a message checking service and will only respond to messages intend for itself. Messages are basically filtered by a filter mechanism. No Messages in CAN are acknowledged so this reduces unnecessary traffic.

But the receiving node checks for the frame consistency and acknowledges the consistency. If the acknowledge is not received from any or all the nodes of the network, the transmitting node posts an error message to the bus. If any of the nodes are unable to decode the transmitted message due to internal malfunction or any other problem, the entire bus will be notified of the error and the node re-transmits the frame (Tindell et al 1995).

2.10.5 Addressing Modes in CAN

A CAN network can be configured to function as either the standard 11-bit addressing mode or the extended frame format the 29- bit addressing mode. The only difference between these two formats is the length of the identifier field of the frame, The IDE bit indicates the type of addressing is used.

2.10.5.1 11-bit addressing

The 11-bit addressing mode is used if the IDE control bit is set to zero. For 11-bit addressing, the functional ID of 0x7DF can be used in the arbitration field by external diagnostic tools and works as a broadcast address where every ECU will receive messages with that ID. An ECU will respond to messages with its assigned ID plus eight in the arbitration field. For example an ECU with assigned ID 0x7E0 will receive messages sent with that ID (and broadcasts) but

when replying it will set the arbitration field to 0x7E8 to indicate that it is a reply. (CAN in Automation, n.d)

2.10.5.2 29-bit addressing

If the IDE control bit is set to one, 29-bit addressing mode is used. When using 29-bit addressing the arbitration field contains both the source and target address of the transmission as specified in table 3-6, and is used to provide compatibility to other serial communication protocols that might be used in the car. The 29-bit addressing mode is also called extended addressing mode.

Table 2.2: The structure of the 29-bit arbitration ID

CAN id type	bit 28-24	bit 23-16	bit 15-8	bit 7-0
Functional	0x18	0xDB	Target Address	Source Address
Physical	0x18	0xDA	Target Address	Source Address

In 29-bit mode the functional address of 0x33 is used as a broadcast address and 0xF1 as the external diagnostic tools address (Zuberi & Shin 1997). This means that the functional ID of 0x18DB33F1 is used as the broadcast ID. To send messages to a specific ECU, physical IDs are used. Using physical IDs, ECUs will receive messages with physical ID 0x18DAXXF1 and reply with the physical ID 0x18DAF1XX, where F1 is the address of the external diagnostic equipment and XX represents the assigned physical address of the ECU as defined in SAE J2178-1.

2.10.6 CAN hardware filter

CAN modules supports hardware filters that can filter the incoming messages. The filter compares the arbitration ID of the message to either a range of accepted IDs, or uses a mask and acceptance register to identify accepted IDs. Any message that has an arbitration ID that is not identified as accepted is discarded

2.11 GPS Module

GPS Modules have finally become affordable enough to implement on a wide scale especially for IOT devices for the purpose of adding tracking capabilities. The GPS System

was actually invented by the U.S. Army for military purposes (e.g. missile guidance, tracking planes and soldiers, or navigation), later they made the system available to civilians.

The process of getting the location of any device is matter of triangulation of signals. The process of triangulation is basically the gps receiver sending time signal to the four satellites and because the position of the satellite is known and it is also known how fast the signal travels through space the distance between the receiver and each satellite can be calculated. Using the distance calculated from each satellite the GPS can calculate its position after triangulating these four signals.

There are different GPS formats that GPS modules transmit the two most common formats are NMEA and RTCM. RTCM is the Radio Technical Commission for Maritime Services and NMEA is the National Marine Electronics Association. The U-blox module that we have implemented in this design transmits serial data in the default NMEA 0183 sentence format.NMEA 0183 is a combined electrical and data specification for communication [35]. There are different GPS Formats available but one most commonly used for tracking is NMEA 0183, there are talks of the tracking devices moving to the new NMEA 2000 format.

The u-blox module relays computed GNSS variables such as velocity position, course UTC etc to the UART peripheral of the asset-tracking device. GNSS modules have a standard serial interface either TTL or Rs-232. Data is broadcast via this interface to the peripheral device in the special data format in this specific module in the NMEA 0183 format. This format is a standardised format by the National Marine Electronics Association to ensure that data gets relayed without any ambiguity or errors.

The U-blox can be configured to transmit different NMEA 0183 sentences. The different types of NMEA sentences are list below

GGA, GLL, GSA, GSV, RMC, VTG, TXT

$GPGGA	Time, position, and fix related data of the receiver.
$GPGLL	Position, time and fix status.
$GPGSA	Used to represent the ID's of satellites which are used for position fix.
$GPGSV	Satellite information about elevation, azimuth and CNR
$GPRMC	Time, date, position, course and speed data.
$GPVTG	Course and speed relative to the ground.
$GPZDA	UTC, day, month and year and time zone.

Table 2.3 Different types of NMEA Sentences types

Most end-user GIS applications require GPS data to drive their applications like the control center application in the call center. There are applications compatible with the NMEA format but most GIS applications require the GPS in KML format. KML(Key markup Language) is an XML syntax and file format for modelling and storing geographic elements such as lines, polygons points etc for GIS applications like google earth and googles maps. Most GIS use

this format to overlay. This GIS application processes KML data or files the very same way that do to HTML and XML files. KML format has inherent tag-based structure with names and attributes for special graphic display. The conversion from NMEA to KML is mere conversion of specific indices to specific data types like latitude and longitude data from $GPGGA needs to be converted to decimal format. There are a number of open source applications that do this conversion for us.

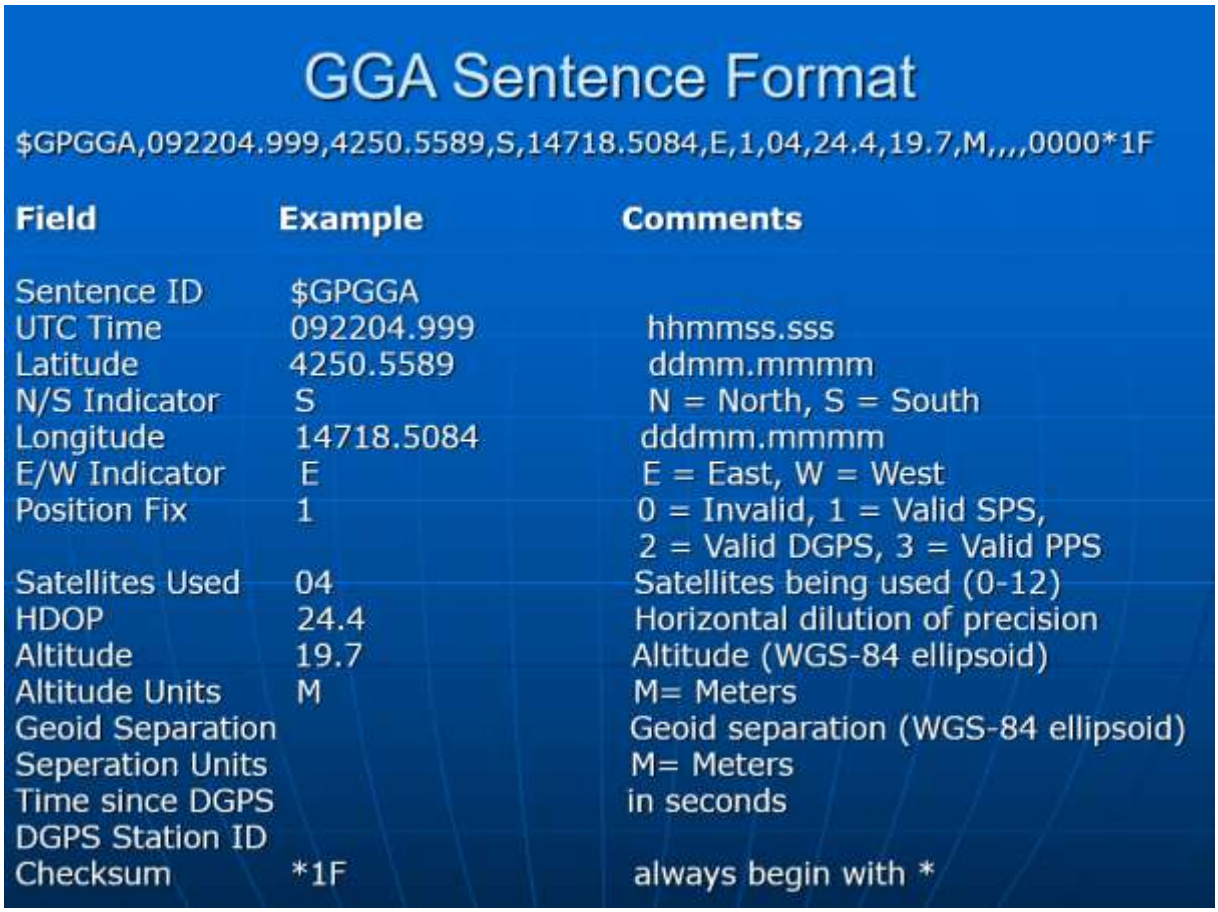


Figure 2.15 Explanation of NMEA Sentence of Type $GPGGA

2.12 Biometric Module

The Biometric module is a wireless module capable of transmitting biometric data to the microcontroller with a range not less than 50 meters. Before implementing this module, it was

necessary to research the different options in terms of relaying data wirelessly for a specific range. This thesis researched the implementation and application of the different wireless technologies. This area has evolved tremendously over the last decade especially with IOT gaining ground in the industry. There were various options available to meet the requirements of this design as mentioned briefly in chapter2: literature review but after researching the wireless environment and review different implementations of the wireless technologies the most robust and popular for ATD specification requirement was the nrf2401l module because of the following reasons:

- Cost
- Ease of use
- Operating frequency
- And Range

Cost

The nrf2401 wireless module is one of the cheapest on the market going for as little as R50 a piece. There are variations of the theme available and if a designer is looking for a bit extra range there are versions of this module that come with an external antenna for under R90. For a successful wireless link this design will need a minimum of two modules and a microcontroller each with SPI. The ideal module for this design would have been to use the nrf24E RF module as this module has a built-in 8-bit microcontroller but unfortunately during the building of the prototype there weren't any of these devices available, the design implemented two nRF2401L modules with two microcontrollers instead.

Operating frequency

The nRF2401l operates at 2,4Ghz band. When looking at distance of transmission or range, these modules using the right setup can transmit over several kilometers. While 2.4GHz is technically not a microwave, its wavelength is close. Frequencies over 3GHz are classified as a microwaves. Being such a small wavelength allows very high gain antennas to be used such as parabolic dish antennas. However, for this project it is only necessary for line of sight range of about 50 meters. So a basic lower power module was more than adequate for this design. The biometric device must be small enough to fit a wristband and consume very little power.

2.12.1 Functional description of Biometric module

The purpose of this module is to have the biometric data and a panic button wirelessly connected to the stm32f mother module so that this data can be collated, buffered and transmitted to the remote control centre. To achieve this interface, the device will be using an nrf2401L module and a nrf24e1 module. However, as mentioned early for this implementation two nRF2401l modules will be used, one stm32f4 board and one Arduino UNO board; as during implementation of this design there were no nRF24E modules available. If an nRf24e1 module available, the design would have done away with the Arduino microcontroller as the nRf24e1 has an 8-bit microcontroller and a 10-bit ADC built into itself. We will however discuss both and sometimes discuss the modules interchangeably.

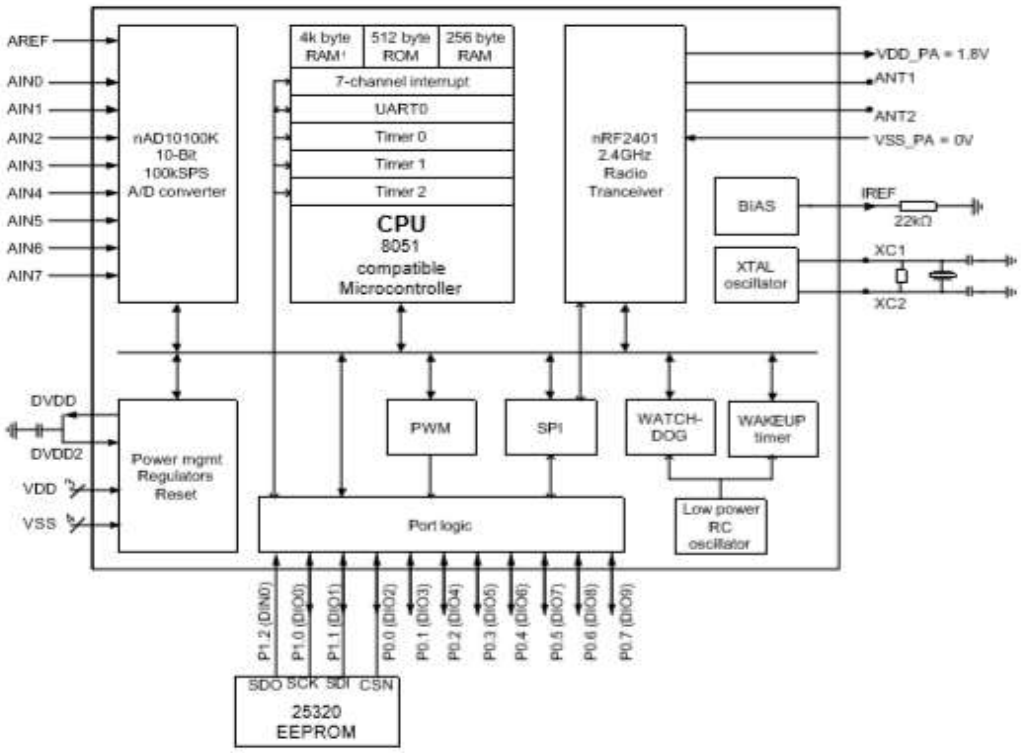


Figure 2.16 Block diagram of nRF24E1 with built-in microcontroller.

The nRf24E is the desired module, as we would not require a separate microcontroller to complete the wireless link. The other requirement specifically when the device is in production, the transmitting module needs to very small and be wearable by a person.

For the purpose of this design and for testing the biometric module interface is made up of two RF module interfaces and two microcontrollers. Each nrf2401l module is connected to each microcontroller via the SPI interface as seen in Figure 7.1 above.

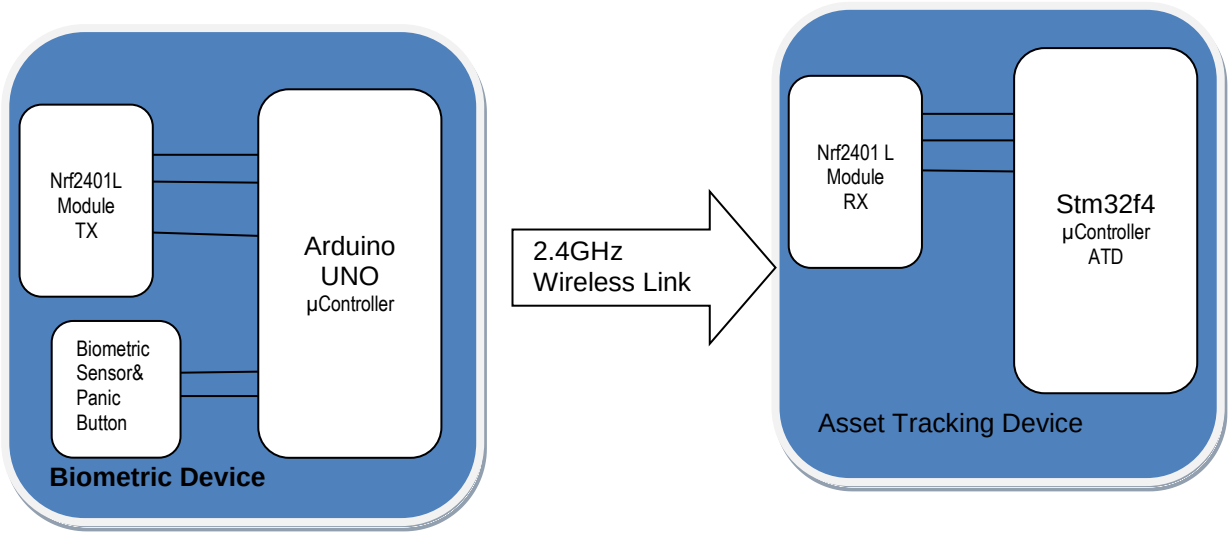


Figure 2.17 Diagram illustrating Biometric Module

2.12.2 Principle of operation of the Biometric module.

- The ATD with the master nrf2401l module attached will be monitoring the airwaves for biometric data and Panic button data.

- The Master device which is the Arduino Uno 2401l interface via SPI will send biometric Data with regular time intervals. If the panic button is pressed this is an event where an interrupt will be generated and Panic data will be sent to the ATD.

2.13 GSM/GPRS Module

GSM Originally stood for Groupe Speciale Mobile name after a group that devised it. The official protocol was launched in Finland. GSM is considered 2G (second-generation) protocol is now called Global Systems Mobile communications. Each base station is interconnected to another base station like cells which in ideal conditions have a radius of up to 35 kilometers, therefore we talk about the cellular network. The original 2G network was based on FDM frequency division modulation and later TDMA relied on circuit switched technologies and was designed originally for speech traffic.

GSM system is originally designed for speech traffic. In the previous generation GSM architecture data delivery was very complicated, hardware intrusive and clumsy which lead to high device and architecture costs. Data connections relied on cumbersome multiplexing techniques created with a dial-in connection protocol like that of fixed telephone networks. Some enhancements were made to the original GSM specification to enable more efficient data connections. These include HSCSD for faster switched data transfer and GPRS for packet data delivery.

The generic Packet Radio Service (GPRS) operates in packet mode to efficiently transfer mobile data and signalling information. GPRS enables dynamical radio resource allocation between users in order to optimise the network and radio resource usage.

GPRS is the packet-oriented extension of GSM. This extension relies on the re-use of the radio infrastructure of GSM while introducing new network nodes in the core network providing the required packet switching functionality. GPRS is mainly intended to provide better service for Internet applications compared to existing more expensive circuit switched services of GSM. With the convergence of data GPRS technology is evolving every day and making data transmission and optimisations on the network seamless as well as less expensive.

2.13.1 Using GPRS

The main packet protocol to be used in GPRS is IP which has two subsets TCP and UDP, defined. Radio and network subsystems are strictly separated in order to allow GPRS network subsystems to be used with various radio access technologies.

Cellular data transmission has been an increasingly attractive mechanism for communication with remote, non-permanent or mobile devices and of recent times have extended their applications to IOT devices. The ability to collect and distribute data virtually anywhere without requiring the limitation of working within specific Wi-Fi “hot spots” is a powerful medium for efficiency and reliability. The cellular infrastructure of today in South Africa covers even the most remote of locations. This also makes IOT devices on the fly portable requiring no configuration if the device is moved.

However, the fact that cellular data service providers meter or measure means that the frequency of transmission and amount of data sent in each exchange can have significant cost implications to the end user or asset owner especially in South Africa where cellular data cost is significantly high than the rest of the world. In order for asset tracking devices to have an impact in terms of adding value to asset optimisation these costs have to be minimal therefore it’s imperative that the design of an asset tracking device reduces the data cost and employs efficient data transmission techniques and algorithms. During this research we embarked on comparing different data format transmissions to explore the cost using these different formats as well as their suitability for different applications. We will also have gained deeper insight and perspective and baseline knowledge for further and future development.

In cellular data networks, data is transmitted in TCP or UDP packets comprised of a data payload of one or more bytes and a set of additional bytes called a header that is attached to the data payload together with some form of IP encapsulation (Perkins 1996).

As mentioned earlier the asset tracking device implements a GPRS interface for the purposed of transmitting packed data to a remote control centre. This research, investigated

the two subset of the IP protocol namely TCP and UDP individually with the purpose of understanding their differences and cost implications in the context of asset tracking.

In the cellular data network the Basic TCP/IP protocol family cannot handle IP host mobility. The way that TCP/IP is handled in cellular network environment is fundamentally different to the normal LAN and WAN network. TCP/IP and UDP has to contend with "mobility issues". Mobile IP extension is developed to allow an IP node to move and still maintain its IP address and ongoing connections. (Cai & Goodman, 1997)The wireless network architecture has a dynamic IP address functionality where it uses a permanent home address and a temporary visitor address given by a foreign agent.
It is known that the predecessor cellular networks are designed for speech traffic. Mobile IP allows TCP/IP connections over these networks, but data connections in these predecessor networks were not effectively supported however with the convergences in terms of the need for data, cellular networks have evolved with enhancements that are very "data effective" which are now present in GPRS and third generation cellular. One can only imagine how the 4^{th} generation cellular will influence data transmissions.

The cellular network environment poses many challenges in terms of high bit error rate and the attenuation effects present in a radio network. Further to these issues the mobile cellular environment host location may change during a connection which creates a problem to the basic routing protocols. Therefore the introduction of GTP.

The ability of a node to change its attachment point from one link to another while maintaining its IP address and all old communications is not possible with normal IP routing, where IP addresses and routing are based on subnetwork prefixes. Different protocol is thus needed to solve the two challenges associated with the mobility: when a node switches to another link without changing its IP address, there is loss of route so the it cannot receive packets in its new address, and if the node changes its IP address when it moves also loss of original route therefore it must terminate and restart any communications.

The Mobile IP introduces three new functional entities:

• Mobile Node

A mobile node is a host that changes its attachment point from one network to another. It will do this while maintaining its original IP address while maintaining its ongoing connections.

• Home Agent

Home agent is an intelligent switching device or router on a mobile node. Its primary function is to maintain location information for the mobile node and tunnels datagrams to the mobile node when the node is away from the home network.

• Foreign Agent

Foreign agent is a router on a network that a mobile node visits. It provides routing services to the mobile node, and de-tunnels and delivers the datagrams that were sent by the home agent to the mobile agent.

2.13.2 Overview of Billing of GPRS data

Cellular networks have complex process in terms of measuring cellular data.

Cellular networks generally will measure data flow at a router or some other network device. For example, on a GSM network, this router is referred to as a GGSN. The GGSN sits at the far edge of the cellular part of the carrier's network from the mobile equipment – the GGSN is the device that ultimately gives the mobile equipment its IP address.

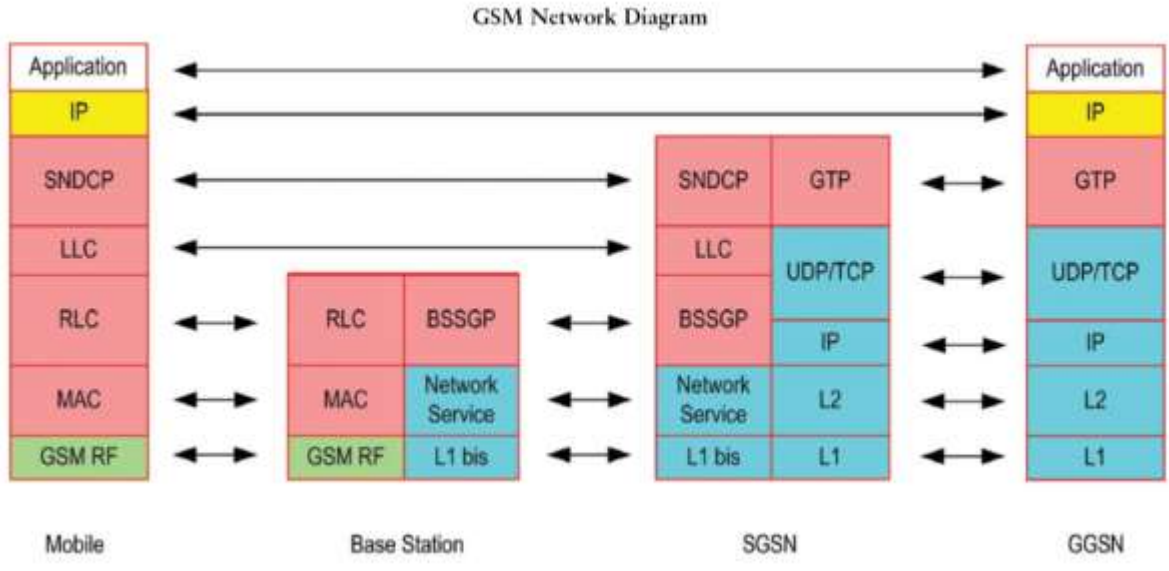


Figure 2.18 GSM Network Diagram

Simplifying, the device opens a tunnel to a GSN (Gateway Serving Node) which is a server installed at the carrier. Which GSN to use is based on the APN (Access Point Name) supplied when the tunnel (PDP context) is requested this is very similar to way routing takes place on a Wide area network. The tunnel is assigned an IP address at the GSN and that is the address used for IP communication. Packets will be filtered at the GSN and routed to the specific User data and GTP signalling and control information has to be exchanged between GSNs. The path is determined by the IP address and the UDP port number. The combination of these two parameters is commonly referred to UDO/IP path.

(Rhodes & Schwarz 2005)Traffic is tunnelled between the GSN and the device using telecom specific protocols. Packets are not broadcast out to all devices and then filtered there. So essentially the GSN acts as a router if we were talking about a typical LAN. The above topology applies to GPRS using 2G but other protocols like 2.5/3/4G use the same structure with some enhancements.

GTP, GPRS Tunnelling Protocol: This protocol tunnels user data and signalling between GPRS Support Nodes in the GPRS backbone network. Tunnelling means that the data is not influenced by lower layer protocols. All PDP, PDUs are encapsulated by the GPRS Tunnelling Protocol. GTP is used for data transport and for signalling information between two GSN's.

In the case of data transport, GTP makes use of the tunnelling mechanism, which transports data between two endpoints. Which way the data packets are conveyed is dependent on the application. This determines for example reliable data transfer or not.

Many applications, especially in the device monitoring space, have built-in mechanisms in place to validate data and request retransmission of missing information. Reliable data communications are quite commonly implemented over UDP using these applications. Letting the application handle the data integrity/retransmission often results in considerable cost savings and can sometimes even lower latency end-to-end. There is an increase in algorithms that specialise in handling UDP data of late targeting the IOT space.

There are applications that oversample deliberately. So if there is a loss of communication the information or data is not lost as there is a high probability that the information was sent because of oversampling.

2.14 TRANSMISSION CONTROL PROTOCOL (TCP)

TRANSMISSION CONTROL PROTOCOL (TCP) The TCP format was first defined in an IETF RFC specification, RFC 761.The term "TCP/IP" stands for Transmission Control Protocol / Internet Protocol and refers to a number of protocols. The "IP" part of the term, which stands for Internet_Protocol, is used by TCP and UDP. TCP is connection orientated (which means that some signalling happens back and forth to establish a virtual 'connection' first). It also includes mechanisms to provide error detection, error correction and correct packet delivery order. TCP also includes flow control, to avoid the sender overloading the receiver, and congestion control to avoid network overload (Kurose & Ross, 2013).

In general, a TCP/IP header has 40 bytes, and each TCP/IP packet sent will generate a reply acknowledgement (ACK) packet of 40 bytes with no data inside of it which essential means that the overhead of each data transmission is a minimum of 80 bytes. If a packet is not acknowledged, the TCP/IP packet containing the data will be resent. All of this is handled by the protocol itself and cannot be controlled by any of the higher network layers. The rate of retransmission of packets depends on the reliability of the underlying network and the configuration of the TCP stack. Additionally, the application itself may attempt to resend the data, so even if the TCP/IP stack discards the packet due to a network timeout condition, the application itself may send the data again, causing a new packet to attempt to propagate across the network.But for an embedded system, TCP can be overkill.

In GPRS environment data cost can accumulate quiet quickly under bad network conditions:

In a network where TCP/IP is implemented a dropped packet will generate multiple TCP retransmissions by default before the data is discarded. This is great concern in term of data monitoring where the cost can reach undesirable proportions if the polling rate is less than the TCP timeout, the application might send multiple requests for the same information across the network, and occasionally receive double responses from the device. In general, the configuration of TCP keep-alive should not be set for a value greater than the amount of time the application will continue to listen for a valid response (Kurose & Ross 2013).
Each time a TCP connection is re-established or torn down, additional traffic is generated. Keep-alive packets to ensure connections remain active also generate traffic. In general four keep-alive packet transmissions take as much data as one close and re-open connection. There is also a great impact in terms of data latency as well as data costs. In terms of this design original intention to reduce data cost this does not contribute to this design requirement thus more attention was payed to the UDP protocol.

2.14 User Datagram Protocol-UDP

UDP is a much simpler protocol to understand and use than its transport layer peer TCP. UDP provides only for the "connectionless" exchange of information between two devices on the network. Since there is no "connection" to track and maintain and there is no guarantee of successful communication, the UDP protocol is stateless.

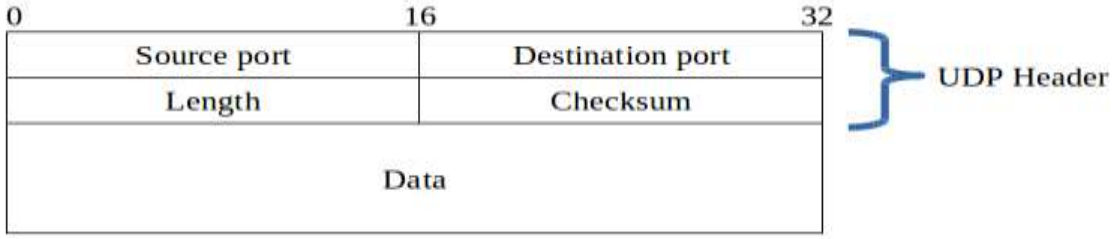


Figure 2.19 UDP structure

UDP is a "best-effort" protocol. When a packet is sent the protocol doesn't concern itself with delivery of the packet. The protocol itself does not have a mechanism to check delivery of message. If we want to ensure delivery or acknowledgement the software itself need to implement a mechanism to handle that. TCP handles the connection, the successful delivery and the throughput as well. UDP could be useful for sensor networks and packet-oriented applications where data streaming and real time applications are not required like in growing Internet of things applications. One of the prime design requirements of the asset tracking device is to achieve a reduction in data cost. Looking at the type of data as well as the integrity of data required for this design it can be seen that UDP protocol will be more than adequate for the this design as opposed to TCP where there unnecessary cost overhead as well as the number of retries could render this design useless from a data cost perspective. The standard UDP/IP packet will have 28 bytes of header data, however UDP packets are only sent one time, and there is no ACK. UDP therefore has at least a 50% advantage in overhead on a highly reliable network. If the application or data device will resend or re-query for data, UDP can offer significant savings in terms of network efficiency (Kurose & Ross 2013). In the evolving GSM/GPRS network reliability of service is reaching incredible proportions and therefore the ever increasing throughput of UDP packets on a gprs network in comparison to years before (CAIDA n.d, 2014).

UDP and TCP are used in different situations for different applications. TCP has its place in high reliability applications like file transferring using FTP where a loss of packets or even one packet could render the whole transfer useless and file could end up as a corrupt file. UDP is unreliable and connection-less which suits real time applications such as VOIP and streaming. Packet loss up to certain limit in these applications are not as critical as in a file transfer. If the application can withstand minimal packet loss then UDP is more than adequate for the application. Udp also has no mechanism in place to manage the order of the packets received it is expected of the application to handle this. Secondly TCP has many variants depending upon how they react to packet loss by controlling its window size.

Many applications, especially in the device monitoring space, already have mechanisms in place to validate data integrity and request retransmission of missing information. Reliable data communications are quite commonly implemented over UDP using these applications. In the South African context, the Cellular networks are extremely reliable and more often than not UDP will be more than adequate for monitoring systems.
Most asset managers especially in third world countries at this stage in terms of Internet of Things are very concerned of data costs and require solutions that can meet this requirement. Most IOT base solutions need to show a real return on investment to be implemented as well as to show direct benefit to the asset owner. Fortunately, for the asset-tracking device it represents real projected savings in terms of asset management.
In this application, loss of packets does not pose a problem in terms of data integrity well as the ordering of the packets or data. The ordering of the packets if needs be could easily be overcome by time stamping the data packets and have the application take care of the ordering of packets.

2.16 Buffering techniques

In this design, it was necessary to have good grasp of implementing proper buffering techniques. It was this part of the research that long hours of researching different techniques to gain a great and complete understanding of low level buffering and the use of the different resources the microcontroller has to effect efficient buffering.

In this design there are numerous peripherals collecting data asynchronously and at different rates. In terms of data handling efficiency if data comes in slowly in individual bytes as in letters like in this design its more convenient and efficient to handle them as chunks of data. This creates a need for buffers, also seeing that collecting data from different peripherals at different baud rates and that this data represents different type of data not specifically type of

data but more so kind or source of data with would also want to store them at different locations that can be easily accessible and handle it the way we want to.

For this design and in layman's terms what is needed to do is store data that is acquired via the different peripherals store it at a specific location (buffer) and transmit it from this location when we want to in a controlled manner. Also for the purpose of transmitting udp packets via the GSM module it is necessary to serialize this data for transmission with the intent to reduce data cost.

Buffers in short sits between two asynchronous routines and makes sure that either of them can act when they want without forcing the other routine to wait for it. The buffer that will be implemented is one that the data comes in and leaves in the same order first-in, first-out (FIFO) buffers. For a simple FIFO incoming data can be stored in the last free slot in an array, and read the data back out from the beginning of that array until it runs out of data. A great feature about even the simplest buffer; one process can be reading data off of the front of the array while the other is adding data on the back. They don't overwrite each other and can both access the array in turns or in bursts.

There are basically two types of FIFO buffers a linear and a circular. Linear buffers are suitable if is required to read everything at once it is also a less cumbersome buffer in that it is easy to determine if the buffer is empty or full, if the last available slot is the last byte in the array then its full and if it is the first slot in the array then its empty.

The trick to writing multiple messages to a single file or stream, is to keep track of where one message ends and the next begins. Most serializing buffers are not self-delimiting, so buffer-parsers cannot determine where a message ends on their own. The easiest way to solve this problem is to write the size of each message before you write the message itself. When reading the messages back in, read the size, and then read the bytes into a separate buffer, then parse from that buffer. In terms of serializing the data on the ATD low level "serialization" routines written in C was implemented whereas in the remote control center there are numerous options available dependent on the SDK platform used to implement the software of the control center and asset lifecycle management software.

The principle of serialization has many forms in different programming languages but they are all a mere variation of the theme and the principle is the same. The theory and implementation of serialization and de-serialization is a thesis in its own right but the principle is very similar to the buffer algorithm in a sense. A very import ant part of the serialization process is to have some unique delimiter to separate the data between each buffer; on the other side or the receiving end the software routine will be use the same delimiter to segmantise the data back to its original form. It is also important to realize that the gprs

tunneling protocol will also implement its own segmentation of the datagrams and on the receiver side the concatenation of the datagrams; this is part of the transport layer protocol implemented by the GPRS tunneling.

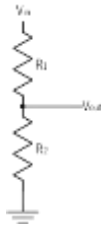
The best way to explain circular buffer implemented in this design is to explain the functions in the code that implement circular buffering for this we will explain the implementation of the circular buffers in the GPS module as seen in **Appendix A**

2.17 Level Shifting

With the evolution of microcontrollers and other digital devices the voltage range of these devices vary from device to device and microcontroller to microcontroller. The norm for microcontrollers has been 5V for many years. Recently, however, new controllers have been put into circulation that operate only at lower voltages. 3.3V has become common, and some controllers even operate as low as 1.7V. In line with microcontrollers changing, the peripheral devices that attach to them like sensors and actuators that attach to them are changing as well. For example, many accelerometers on the market now operate at 3.3V instead of 5V. Because of this, it has become essential to convert the output voltage of one device to match the input voltage of another. This is called **level shifting**. There are a number of ways to do it.

There are three basic ways or principles of level shifting viz;

2.16.1 Level Shifting With a Voltage Divider



Voltage divider level shifting.

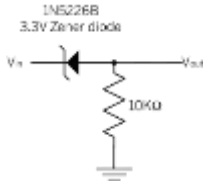
In the image above, $V_{out} = V_{in} * R_2 / (R_1 + R_2)$.

If the resistors are equal, $V_{out} = V_{in}/2$.

If $R_2 = 2*R_1$, then $V_{out} = V_{in}*2/3$.

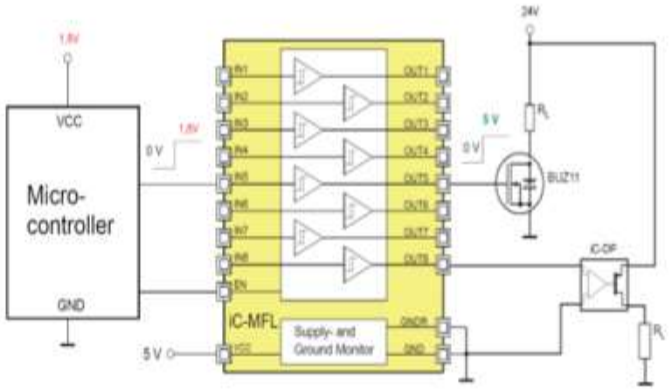
2.16.2 Zener diode Clamping

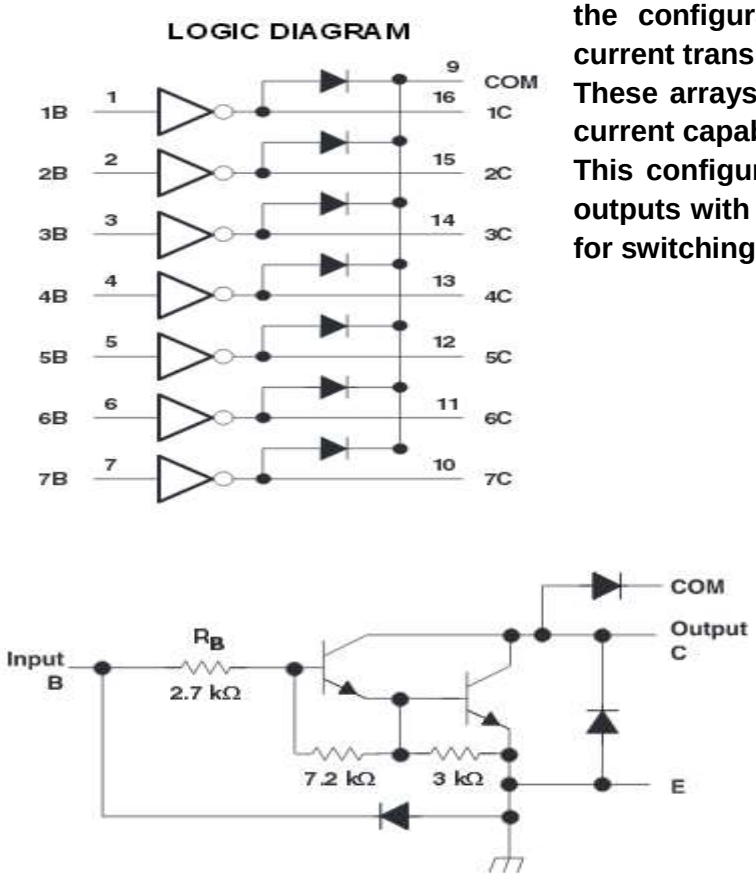
The zener diode can also be used to clamp or to reduce the voltage. Zener diodes are diodes that allow current to flow from anode to cathode just like a regular diode, but also allows current to flow from cathode to anode up to a particular voltage, called the **breakdown voltage**. Zener diodes come in various breakdown voltages, application. For example, the 1N5226 is a common 3.3V Zener diode, and the 1N4733 is a common 5.1V Zener diode. When one is using a Zener diode as a voltage shifter, one should connect the cathode to the source, and the anode to the output, usually with a pulldown resistor as shown below. The above method is used when there very few circuits that requires level shifting if there is more than one circuits or outputs that require level shifting we employ a dedicated level shifting IC.



This is a typical circuit for 5V-to-3.3V level shifting. If Vin is 5V, then Vout will be 3.3V with this circuit

2.16.3 Level shifting with a level shifting IC





the configuration alongside includes high current transistor arrays.
These arrays can be paralleled out if higher current capability is required.
This configuration also feature high voltage outputs with common cathode clamp diodes for switching inductive loads

Figure 2.20 Circuit Representation and Logic Diagram ULN2003

Level shifting is an important component of the overall design of the Asset tracking device as the different modules operate on different source voltages. Proper isolation of these modules from each order is imperative.

CHAPTER 3

Hardware Design and Implementation

This design uses a stm32f4, a 32-bit microcontroller. The stm32f4 development board can be seen as the mother module to five other daughter modules interfaced to it via various communication interfaces. These modules are connected to the stm32f4 module via the GPIO, UART, SPI or CAN

The device is intended to be used as one of many field devices (see figure 4.2) that is linked to a call centre via GPRS.

3.1 The STM32F Discovery Board ATD

This unit is the central core of this project. It is the microcontroller that will process most info and carry out most of the tasks. The programming blocks of this module is as follows:

- Processing of analog signals handled by the GPIO and ADC of the microcontroller.
- CAN Module
- ADC controller handles analogue to digital conversion .of signals that a necessary for asset monitoring that is not available on the OBD or CAN network or any other mandatory signals required for the asset lifecycle management.
- Biometric Module wireless nrf2401l module that acquires biometric and panic button data
- GPS Module
- OBD Module
- All the functions and modes of operation are programmed on this microcontroller.

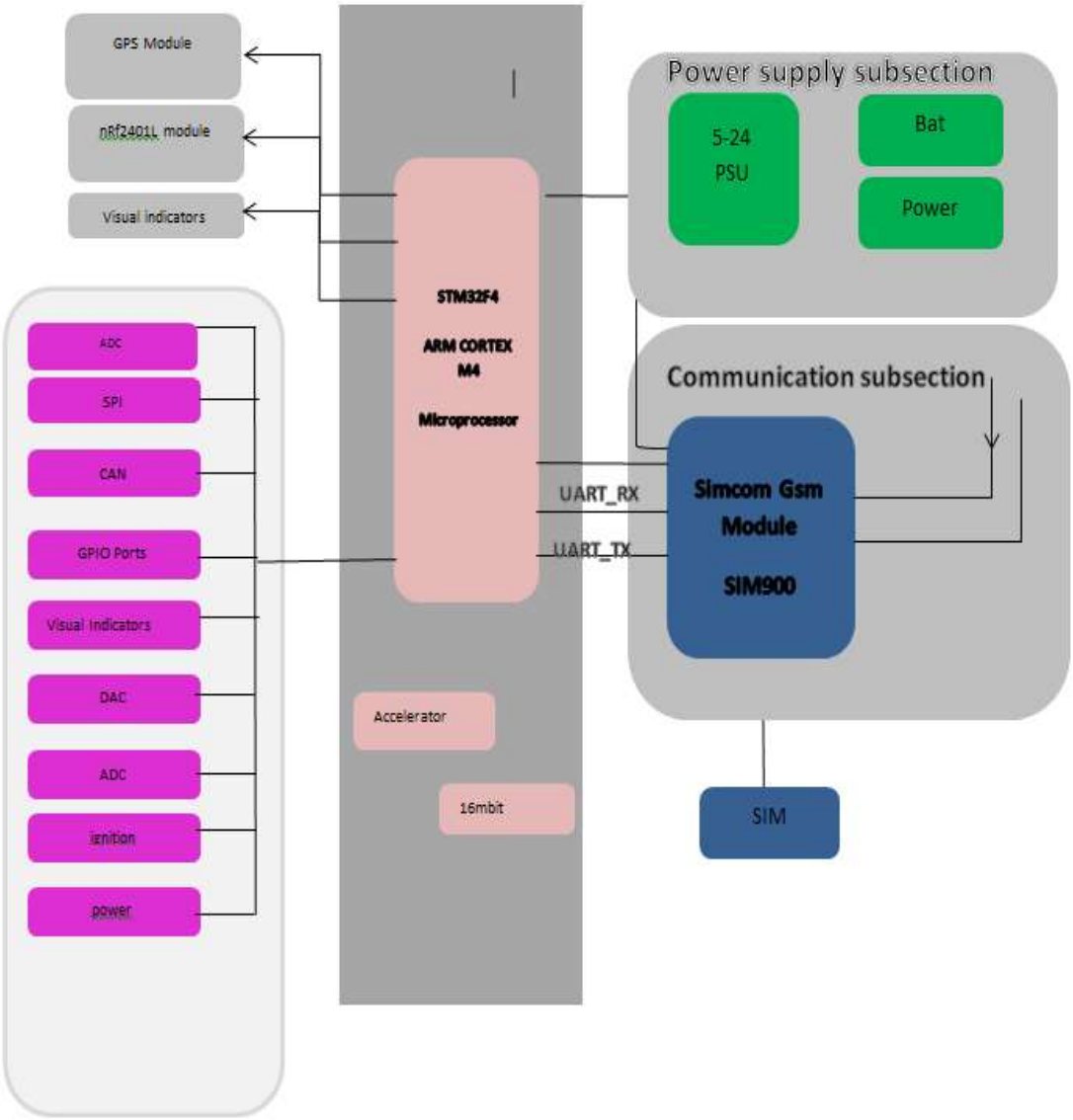


Figure 3.1 Functional diagram of asset Tracking Device

The above diagram is the functional diagram of how design is implemented on the stm32f4 discovery board.

3.2 Asset-tracking Device Different Scenarios

In order to get a perspective of what the asset tracking device is and how it fits into the asset life cycle management paradigm as well as the applicability of the device, this thesis will sketch out a few different scenarios to demonstrate how this device can be applied to different applications and how having a device like this will also assist or drive a maintenance management environment/system.

3.2.1 The asset-tracking device Application Scenario 1:

Asset-tracking device fitted to a vehicle of a safety and security officer e.g. metro police. The metro police officer will wear a biometric device, which will be paired to the ATD via the biometric wireless module namely the nRf2401 module. The ATD fitted to the vehicle with its own unique id (Cell phone number) will transmit GPS, OBD, Biometric as well as other relevant I/O information to a call centre. The call centre will process this data against to ring fenced data.

Scenario: metro police goes on a call out to domestic violence situation, is attacked and wounded. The biometric device that has a panic button that can be activated/pressed transmits this distress signal to the call-centre. If he/she cannot activate panic button because he passes out or incapacitated this will result in a pulse rate drop, this data will transmitted to call centre. Pulse rate threshold will be reach as per call centre software this will alert call centre agent who can alert/dispatch nearest available officer of the situation and aid can be dispatched efficiently. This can also be an automated process by the call centre software.

Also vehicle obd data like engine coolant temperature, oil pressure etc being transmitted to call centre and if the vehicle is operating under exception conditions this data is captured as well call centre agent will be alerted to inform the vehicle operator of this violation and ask the operator to stop if condition is breached, like vehicle moving while vehicle is over-heating or oil pressure to low. This data also gets logged and stored for data analysis for asset management system. The cost saving benefits for the asset owner ito maintenance management will greatly increase as this will in effect reduce engine failure because of driver negligence. Also driver patterns and excessive revving and abuse can also be monitored via this data.

3.2.2 Asset-tracking Device Application Scenario 2:

ATD device fitted to Heavy duty truck with Cherry picker or any other auxiliary equipment requiring PTO (Power take-off) system. Here again the device will be connected to the obd network of the vehicle. The device will monitor obd data, gps data any i/o data as well as running hours of the vehicle. Most vehicles are maintained against the mileage of the vehicles but vehicles that have auxiliary equipment that that have PTO systems have the engine running while vehicle is standing still. These engines are running for hours while vehicles are standing still while the PTO system is in operation.

There is two management issues one maintenance of engines done on mileage would not be a good indication from a maintenance management perspective and would need to be done on running hours of the engine. The fuel consumption and trends can also not be measured accurately and cannot tie up with financial systems and creates room for fuel theft but with atd device installed running hours of vehicle as well as fuel consumption can be easily monitored. Also the other data from the obd system can also be monitored especially low oil pressure and high engine temp/ over-heating can be monitored. For older models without oil pressure can be monitored via the ADC module of the ATD.

3.2.3 Asset Tracking Device Application Scenario 3.

In the commercial environment like Short term insurance call centre. The ATD devices can be fitted to vehicles of customer of insurance companies for an additional monthly fee. The insurance company can monitor the vehicle and customer. The target customers could person with chronic illness. Apart from all the vehicle tracking services offered the insurance company will monitor Vehicle obd data as well as biometric data on the customer's behalf. Lots of time the vehicle engine management light turns on in the vehicle and driver has no idea what the fault is. Because obd data going to call sent customer can receive sms with exact details of the fault is, he/she can also be further advised on the details of the fault code and if he wishes be directed to the nearest workshop. If the fault is critical, a technician can be dispatched to the aid of the driver.

Also aged person or person with chronic illness can also subscribe to this facility with biometric module if vitals monitored and outside defined parameters the call can call the driver or dispatch medical help immediately thus saving a life possibly.

Figure 3.2: Asset Tracking Device Concept

3.3 Asset Tracking Device Concept Design

Apart from other functions, the asset-tracking device is primarily responsible for data acquisition of the primary asset the motor vehicle. The data acquired will be transmitted to a control centre for the purpose of asset optimisation and operator/application response. There are five data acquisition modules used in this design but only three directly relate to asset optimisation. The module for acquiring live vehicle data for the asset-tracking device is the OBD Module. There are cases where the vehicle itself does not allow monitoring of data via the obd protocol in this case we resort to acquiring data via the CAN interface module. And if CAN not possible we will resort to acquiring essential vehicle data via the Multichannel ADC Module. Figure 4.3 below is the flow chart representation of the Asset Tracking Device.

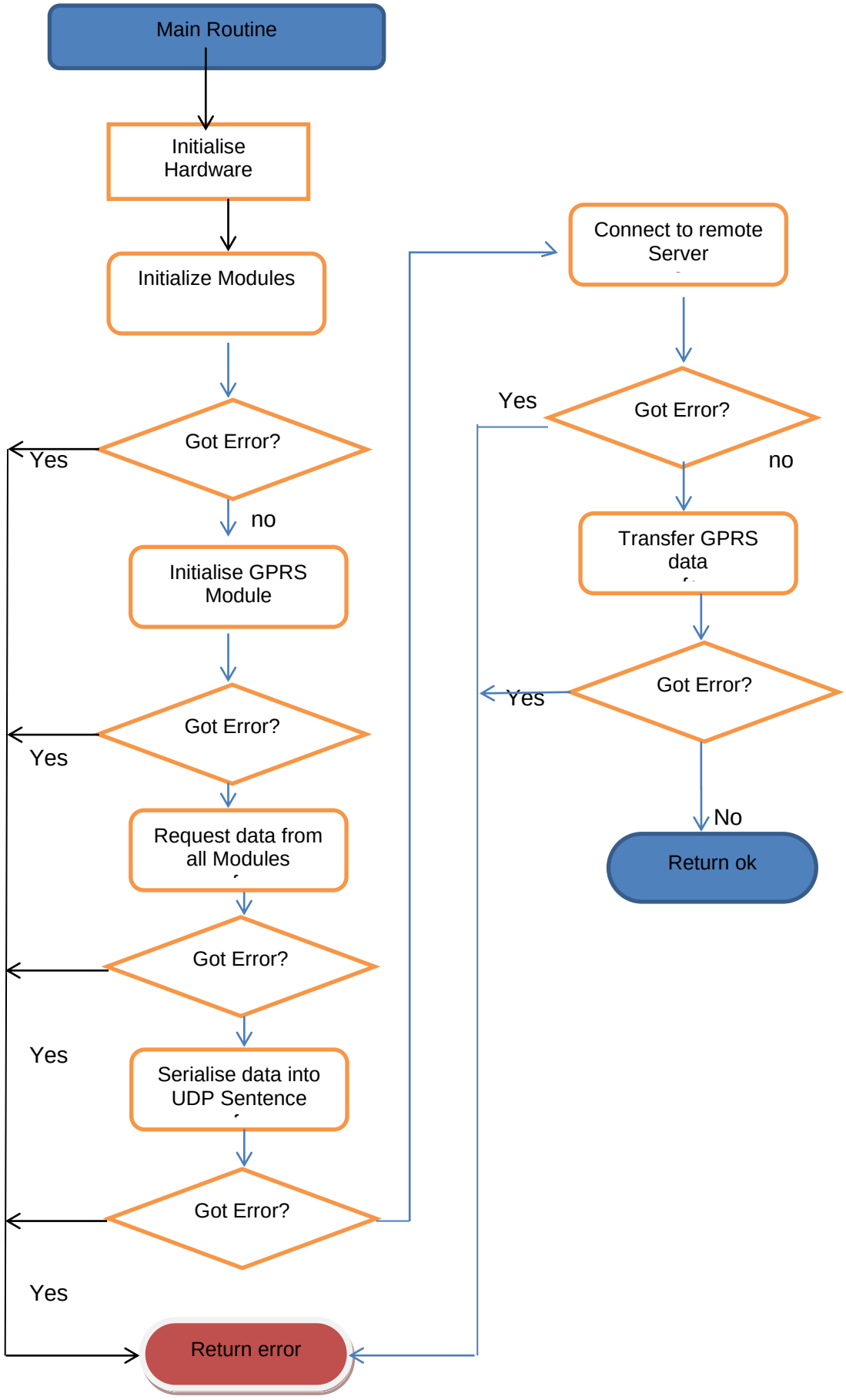


Figure Flow 3.3 Chart of Asset Tracking Device

3.4 Hardware architecture of Discovery Board

This chapter will focus on the detail the hardware design for Asset tracking device with GPRS, including a brief synopsis of the software modules used as well as a discussion of the on-board peripherals. It is in the microcontroller and the coding of the microcontroller is whereby the electronic engineering and power systems engineering of this project lies. It is the clever use of the microcontroller capabilities that make this design.

Figure 3.4 The Hardware architecture of Discovery Board (ST Microelectonics,n.d)

Before going about the coding the microcontroller it is imperative that we understand what we want to achieve in terms of functionality. It is also important that we understand the limitations of microcontroller with regard to the no of tasks that can be accomplished at a time. With a project of this complexity, it is important we research the microcontroller datasheet and use its resources available.

The ST32f4 Discovery is quiet an incredible piece of hardware and offers many peripherals functions that are not dependant on the CPU and operate independently; they do however require the use of the appropriate busses.

One of the biggest challenges in this project was the challenge of the different voltage requirements as well power elecrtronics interface required to achieve totall functionality of the assets tracking device.I breiefly discuss someof these challenges and the solutions.

3.5 Power Electronics Interface

The stm32f microcontroller operates in the 5V and 3V domain. The ASSET tracking device will used in a modern motor vehicle is part of 12V system. The ATD would also operate in the 24 Volt environment if it was used to control diesel systems in the trucking and marine environments. The power electronic interface will serve two functions:

3.3.4.1 Protection and isolation of the microcontroller from transients in the 12/24 volt environment.

3.3.4.2 Switching actuators that the microcontroller would normally not be able to switch/control.

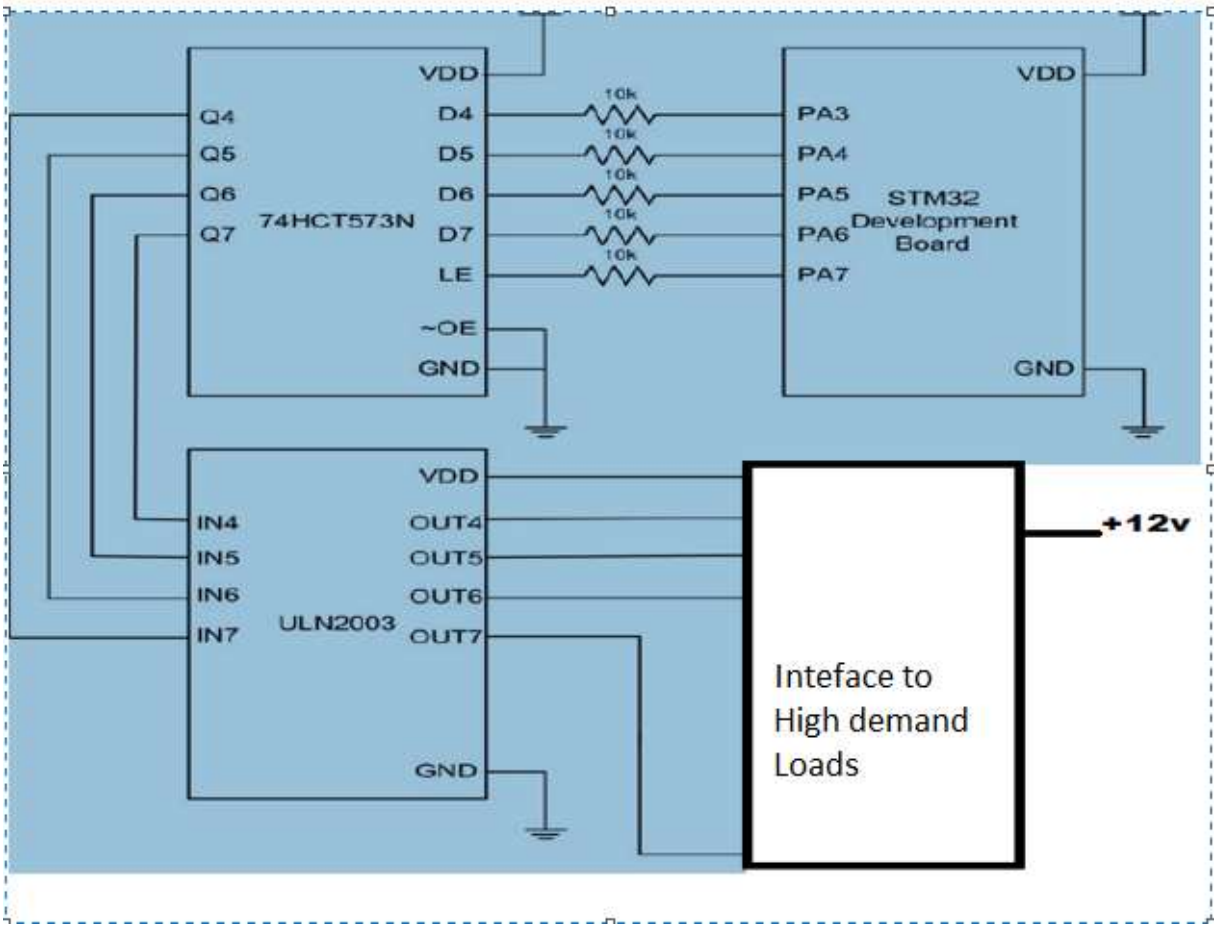


Figure 3.5 Power Electronics Interface used in ATD

The power electronics interface can be seen as a separate module. It allows the asset-tracking device to drive loads with high current demand. This interface is important for remote controllability of the asset tracking device. Necessary care was taken to electronically isolate this module from the rest of the asset tracking device because of the high switching currents that is present when switching high demand loads.

This unit will simulate data required by the ATD that would normally be received by the ATD via the CAN Network. Data sent by this unit will be valid data signals that will be used by the ATD to transmit to the Remote Control centre. Programming this unit will have the following components:

The CAN Module

Data simulation module this data will represent various signals like throttle position signal (TPS), vehicle speed signal (VSS), rpm that will be signals that the vehicle ecu will transmit under normal conditions.

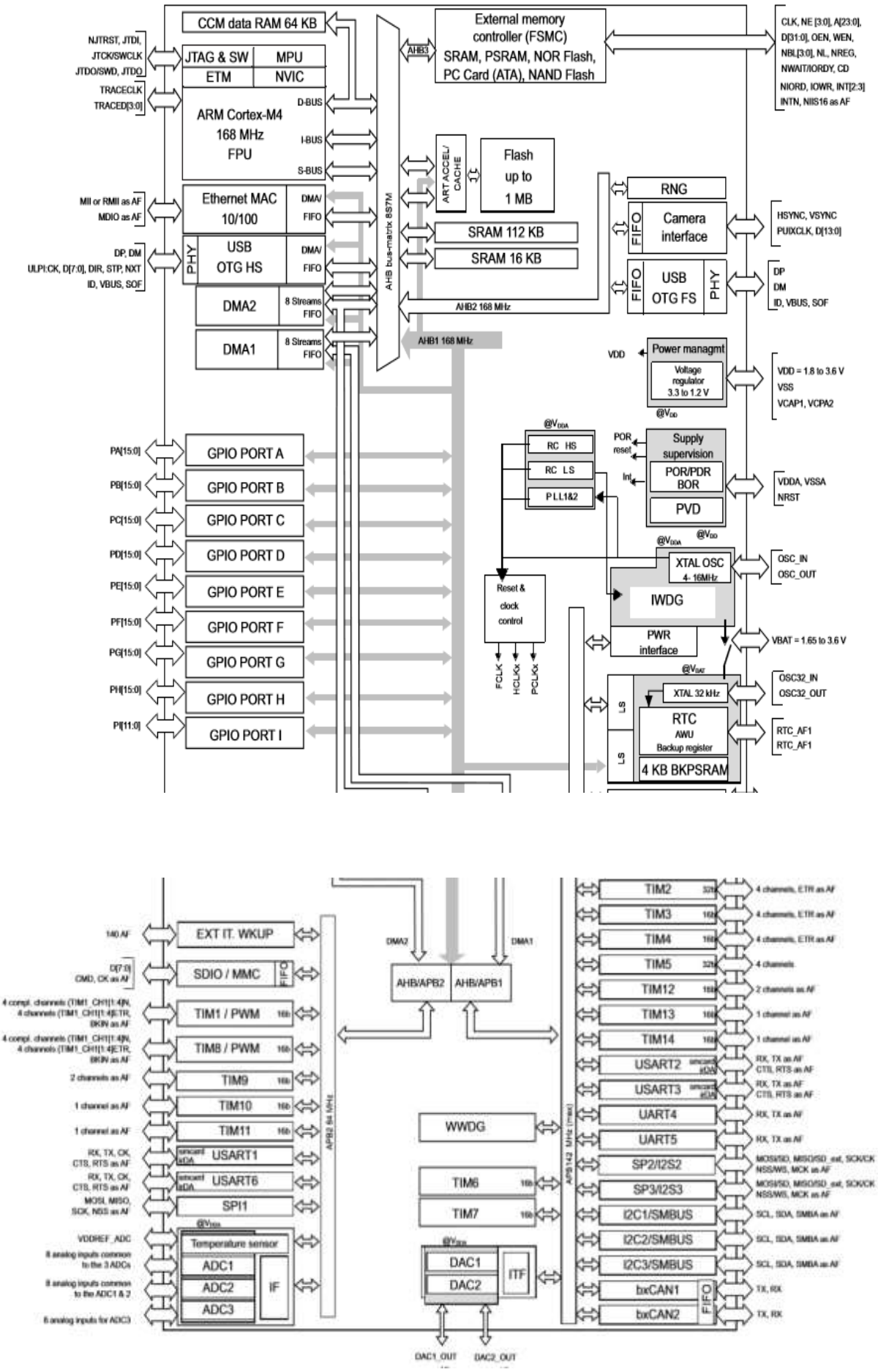


Figure 3.6 Block Diagram of STM32f4 Discovery Board.

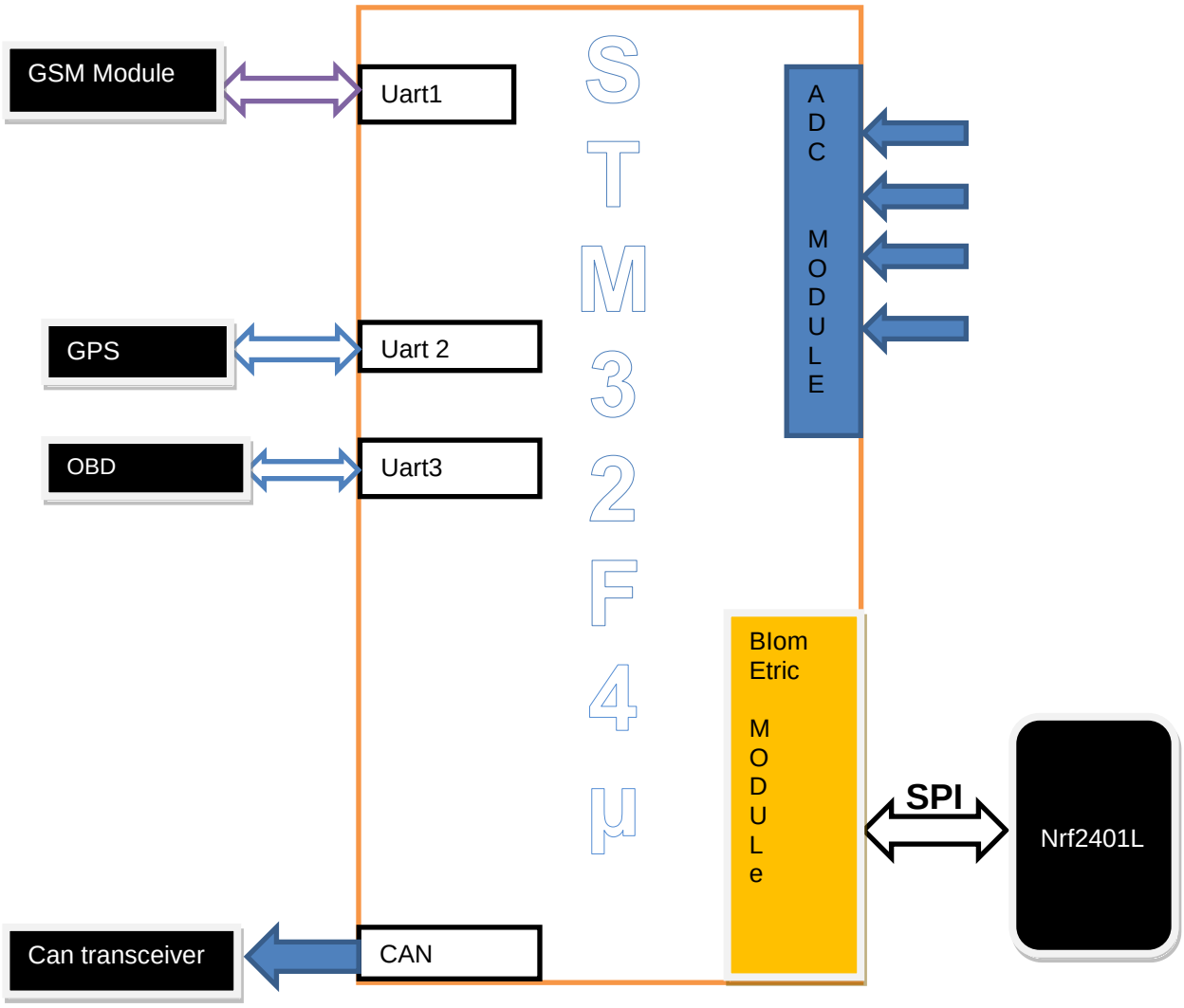


Figure 3.7: Hardware structure of the Asset Tracking Device with GPRS Interface.

3.6 The Software Modules

CAN Module

The multiple ADC channels using DMA module.

Stm32f with nRF2401 to simulate biometric interface

GPRS/GSM module with finite state machine.

The OBD Module.

The Stm32F4 ECU Module to simulate different Signals via CAN Messages Module

The GPS module

3.6.1 The CAN Module

This Module was implemented on the ATD Stm32f4 board. This module handles the CAN protocol in terms of the ISO standard. It involves setting up the different registers and mailboxes to handle the different frames with respect to the protocol. It also associates the relevant GPIO ports to the CAN. Sets up the Baud rate for the Network; sets up the relevant bus and clocks for the Can Controller.

This module is used to set up the CAN Node with regard to the address identification and filtering of CAN Messages. It also handles the programmer defined instruction sets.

The CAN Module on the ATD filters the messages on the Can Network and it if it is intended for it accepts the message and carries out Hex code instruction that it is programmed to do. This module will also be configured as CAN filter with the identifier being that of the instruments cluster and data received by the CAN module will be further screen for PID's that is required by the Asset tracking device, this screened data will be transferred by the DMA to OBD_Buffer. This low-level implementation of the CAN module makes it possible for the Remote Call Center to have limited control of the vehicle.

3.6.1.1 Implementation of the CAN module of Stm32f4

The purpose of Can Module interface is two- fold It will supply OBD data for new vehicles after 2008 year models that we cannot collate data via the Obd module. The secondary or other purpose is to give the control centre remote controllability of limited functions of the vehicle.

STM32F4 have introduced CAN peripheral interface in their series 32-bit CPU Apart from their high performance and reliability The STM32f4 group is equipped all the peripherals and features of their competitors. CAN interface or peripherals are now becoming a standard peripheral on all 32-bit MCU's.

The CAN module on STM32f4 group implements two channel of CAN bus protocol according to the specification of ISO 11898-1. This allows communication of messages in both standard identifier (11 bits) and extended identifier (29 bits) and allows data rates up to 1 Mbps. It was for this reason that we chose this particular microprocessor.

3.6.2 Baud Rate of CAN protocol

The baud rate of the CAN protocol can be set to a maximum of 1Mbps. It can be varied using the external clock source, settings of the internal clock source and prescaler values in the set up registers.

Where Segment Length is used to synchronize data between two nodes as each node may or may not have the same clock frequency. The baud rate on the CAN bus used in this project is 500Kbps.

3.6.3 Interrupts of CAN protocol on STM32f

The CAN module on STM322f provides with the following interrupts.

- Transmission complete interrupt
- Reception complete interrupt
- Error interrupt
- Transmit FIFO interrupt
- Receive FIFO interrupt

The first two interrupts are used in normal mailbox mode of the CAN protocol whereas the last two interrupts are used in FIFO mailbox mode. The error interrupt is enabled for all the mailboxes in either normal mailbox mode or FIFO mailbox mode.

3.6.4 CAN Control Network for the ATD

CAN has earned a reputation of being reliable, robust and real time high speed serial communication protocol which has been around the automotive industry since 1986. It was implemented successfully in one of the toughest environments and have come out tops, it is for this reason it is the only standing automotive protocol that will be around for many years to come (Navet et al, 2005). Since 2008 it is the mandatory protocol on European standard vehicles.

The way CAN Module is implemented in the Asset tracking device is to assume the address/identifier of the Instruments cluster and listen for all messages that would normally sent to the instruments cluster. This process of assuming the address of the ECU on the network is called CAN Filtering. We will not be sending any remote data frames as this will be conducted by the vehicles original Instruments cluster; we will just be intercepting actually not intercepting but receiving all CAN messages meant for the Instruments cluster while the Vehicles Original Instruments cluster also receives those frames.

The data from the Engine control unit is sent on the CAN bus which is received by all the nodes on the bus and the message filtering technique decides whether to act upon the data or not. The kind of data that is relayed to the instruments cluster is data like vehicle speed, engine rpm, engine coolant temperature signal, Fuel level etc. All this information is information that we require remote control centre. Once we receive this message we will once again buffer this message in the CAN_Buffer where we will access this buffer with the serialisation routine for transmission by the GPRS module.

The protocol creates a master-to-master communication bus and every message transmitted on the bus is received by all the nodes connected to the bus. The message filtering technique decides whether the received data is relevant to the node or not. The error detection and fault confinement techniques provided by the protocol are the added features which keep the bus working without any errors and virtually detach the faulty nodes which transmit corrupt messages on the bus (Barrenscheen J, Dr.1998).

For the purpose of acquiring vehicle data for the call centre it is only necessary to operate on the application layer of the CAN protocol. The above specification of the lower level of the CAN protocol only comes into effect when it is necessary to control the ecu's via the Can bus via the control centre. In the application layer of the CAN protocol there are two types of CAN packets normal CAN packets and diagnostic packets. Normal CAN packets are sent by different ecu's and can be seen at any given time on a CAN Network. Normal CAN packets can either be broadcast messages or it could be command for a specific ecu to act upon.

In this design the asset tracking device is part of the CAN network of the vehicle and assumes the identity of the Instruments cluster as mention earlier. This implementation of Can on the ATD device is the low level implementation of CAN and its sole purpose is to offer controllability to the Remote call centre. The higher-level implementation of the CAN protocol is implemented via the OBD module. There is however some redundancy in the design as in the low-level implementation of CAN, the data that is intended for the instruments cluster has all the data parameters that is necessary for the application programs in the control centre.

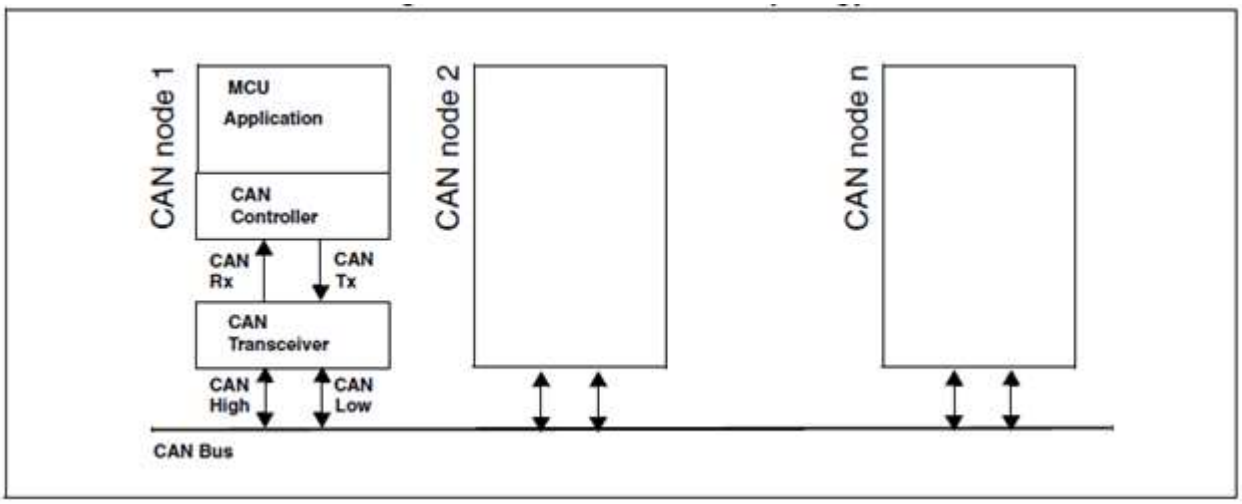


Figure 3.9 CAN Network with Transceivers (Barrenscheen J, Dr.1998)

3.7 GPRS /GSM Module

The core purpose of this module is to transmit data to the remote control center. This module is used by the ATD to serialise the data from the different buffer locations namely the GPS_buffer, the ADC_buffer, the OBD_ Buffer and the Biometric_buffer and transmit it as UDP packets to the remote center. The module also incorporates a finite state machine that controls the time intervals of the data transmission to the control centre.

This module has three states of operation that is determined by the vehicles states Dormant; Idling and Vehicle_ moving. These modes of operation is implemented by state machine as seen in **Section 5.3..**The software component of this module will also set up uart interface between the ATD and GSM module.

To reduce data cost the device needs to manage the amount and frequency of data that is sent in different vehicle states. Knowing for a fact that when the vehicle is park and vehicle ignition state is “off” there is no need to Transmit GPS data frequently and other vehicle data to the control centre or to asset tracking database.

If the vehicle is stationery and but idling the device needs to transmit vehicle data frequently but GPS needs to less frequent. In addition when the vehicle is moving the device needs to transmit GPS data most frequently almost in real time. This scenario spells out that we have different event driven states which mean we can use a Finite state machine to implement the code.

The trick to implementing this solution is to have function in the code to monitor the vehicle states. There are already variables to assist with this from the ADC Module. Ignition and vehicle speed. (These variables must be declared as 'global' so that it can be used in this part of the code). **See Appendix C**

3.7.1 Implementing the GPRS Module

In order to implement the GPRS interface to the STM32f4 ATD device we have chosen the SIM900 Module. This design is interfaced to the stm32f4 device via Serial communication interface using UART peripheral. The GPRS module implemented in this design is the core module in terms or transferring data to the control center, without this module transferring data efficiently with a high degree of availability and reliability renders this design useless with regard to the objectives. In terms of reducing data cost this module has implemented a state machine that monitors the vehicle state and puts the module in the appropriate data transmission state.

3.7.2 Modes of Operation

The GPRS interface sim900 that was chosen to use has different modes of operation viz transparent and non-transparent modes but will be implemented using the transparent mode to transfer UDP datagrams to the remote control centre. The Sim 900 module itself has two modes of connection for the implementation of the TCP/UDP. The modes of operation for TCP/UDP: Multi connection and single connection node. When in single connection mode, the GPRS can work both in transparent and non- transparent mode. See SIMCOM datasheet for detailed explanation of the different modes and applications. This datasheet and application note gives a comprehensive specification including fault-finding procedures regarding the module. When in multi-connection mode sim900 module can only operate at non-transparent mode which means it can only operate as an absolute TCP/UDP Client or only as an absolute TCP/UDP Server. For the purpose of this design, it will operate in single mode.

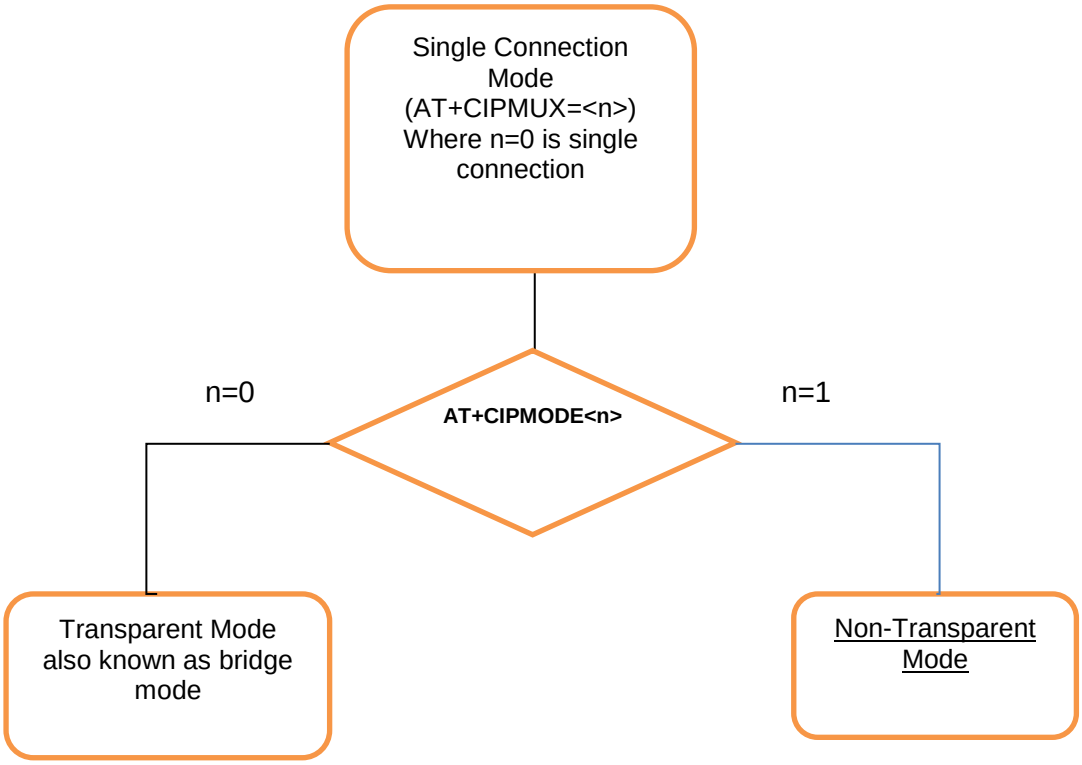


Figure 3.10 Flowchart of achieving single connection Mode in SIM9000

Transparent Mode

Transparent Mode in the sim900 module is an essential mode for handling high-speed data exchange or channel between a client and server node. Transparent mode once established is like a “fixed line” data channel as it provides a special data mode for high data receiving and sending by TCP or UDP application task. Once the sim900 is in transparent mode it allows only limited AT commands. When operating in this mode it is better or preferred to have a large buffer from where the sim900 can “serially” transfer the data.

In this design routine or function was implemented in the code to and from transparent mode in order to get back full AT command functionality. This mode was essential for configurations purposes of the GSM module. Switching out of transparent mode puts sim900 in command mode.

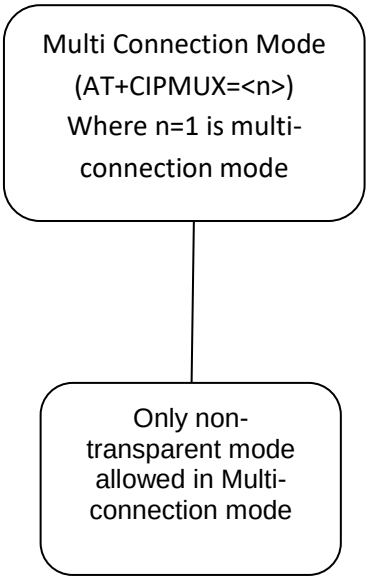


Figure 3.11: Flowchart of achieving Multi-connection Mode in sim900

To enable the features of the GPRS modules device manufacturers implement propriety AT command interfaces for TCP/UDP connections. These AT commands are used to configure the GPRS device to do following for the purpose of this design:

- bringing up TCP/UDP connection,
- setting up TCP configuration, send/receive data to remote server,

Apart from the above configuration the AT commands can be used for various other interactions with the simcom module.

Sim900 modules have embedded TCP/UDP stack. This lets us create IP connections, which are needed for common applications like connecting to a remote server, email, FTP, HTTP etc.

Once the IP is initialized, a successful IP address is available. Once the IP is initialized, a connection to a remote server to a specific port can be made.

The structure used for delivering data in UDP mode is SIM900 TCP/IP application. There are two modes: Single connection and multi-connection.

3.7.3 **Sending UDP over GSM**

Sending UDP data over GPRS in the 2G GSM environment has challenges in terms of mobility as the host location may change during connection. The 2G mobile environment was designed primarily for speech services. With 3G on the other hand these problems don't exists as their main purpose was Internet services and data transmission. Tunnelling is the method employed to resolve these mobility issues. There are three types of Tunnelling methods namely IP in IP encapsulation, Minimal Encapsulation method and Generic Routing Encapsulation Method. The only of these methods that should always be supported in Mobile IP is IP in IP encapsulation.

3.7.4 Configuring the ATD for GPRS Specific to sim900

In the asset tracking device we will not use the AT commands via a terminal program but as an automated routine in the code. This routine /function is there to primarily initialise the GPRs module for the purpose of transmitting data from the final serialised buffer to the control centre.

The GPRS module is interfaced to the stm32f4 mother board via a serial communication interface, using a uart peripheral.

It was essential to include delays for response back from the GPRS module during the initialising of this module.

One of the routines was to set up the various different UART's for the ATD one of these UART's was used for the GPRS module using interrupts.

To establish a UDP connection with remote server

Start a UDP connection (AT+CIPSTART="UDP", "IP address of the server", "port number of the server"), if the connection established successfully, response "CONNECT OK" will comes up from the module.

Once the "Connect ok" response is received we can now send UDP packet to remote server with the AT command (AT+CIPSEND). Module will receive the data from server automatically, and send out the data from serial port. The process of send data with UDP is similar to TCP

3.7.5 To send data

There are three primary ways to send data using the sim900 module after registering a TCP connection or registering a UDP connection.

To send variable length data we will use the (AT+CIPSEND) command, then type the data when response “>”, ctrl+z (0x1a) is used to start the sending.

To send the fixed length data with (AT+CIPSEND=LENGTH), then type the data when response “>”, the data will be send automatically when the length of the data equal to LENGTH.

You do not need to type the terminal symbol in this case. There is another way to send data automatically. First, set the time that send data automatically with the command of AT+CIPATS=<MODE>, <TIME>, then type AT+CIPSEND command, enter the data when response “>”. The data will be sent automatically when it comes to the setting time. When the remote site received the data, response “SEND OK” will comes out from the module. Module still on the command state, you can repeat the steps if you want send other data. Note: The maximum of data block that can be sent at each time is 1024bytes [22].

3.7.6 Data sending

SIM900 provides three ways to send data: changeable data length sending, fixed data length sending and timed sending (SIM900_AN_TCPIP 2013). SIM900 also provides a method to let you know how much data is sent out from the module and received by remote server on an active TCP connection.

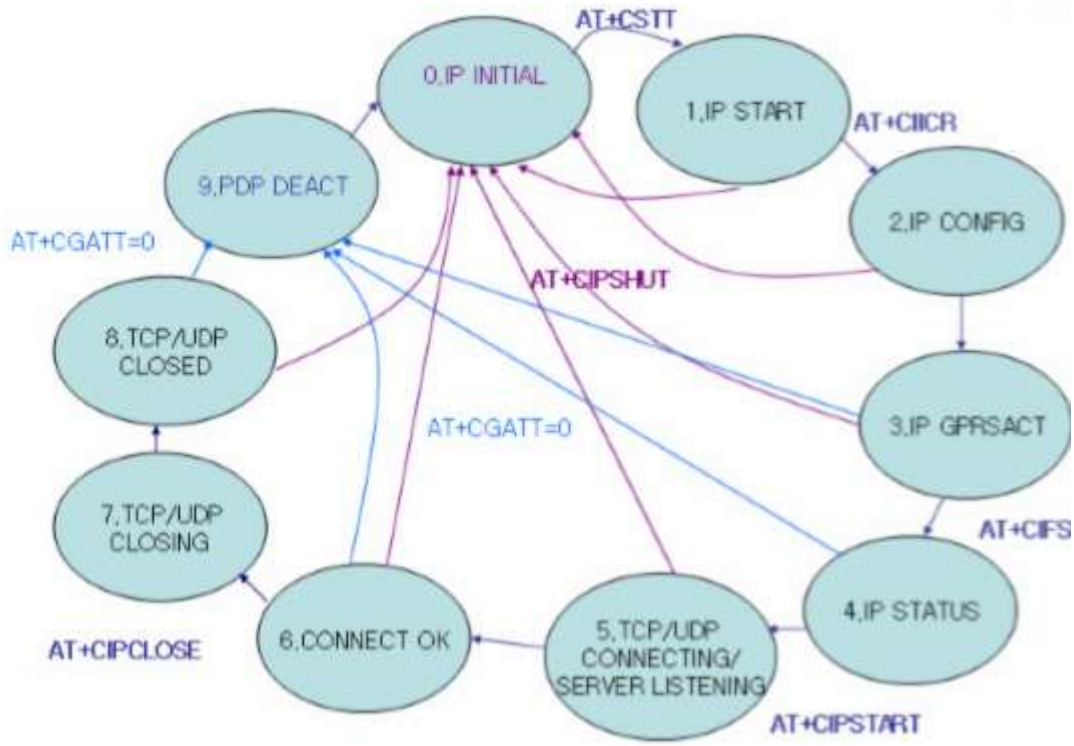


Figure 3.12:Finite state Machine :GPRS States Diagram for single connection.

3.8 Multiple channel ADC Module using DMA

This module is also used by the ATD to process analog input signals that are not available via the CAN interface or OBD module. The implementation of this module has a 12-bit resolution, which translates to 4096 segments of translation. This module is implemented using direct memory Access to free up the CPU to carry out core tasks. Because the input signals are changing all the time and these signals are required by the CPU to carry out instruction loops; it was imperative that we found a way to implementing reading of these analogue signals without using too much of the CPU's time. The principle of operation is as captured in the code in Appendix D. Also this module will be always acquiring data for data transmission as well as for the finite state machine

This module is implemented to enhance the data processing value of the asset-tracking device especially in the earlier model vehicles that do not have OBD ports or the vehicles

where OBD system does not allow for us to enter monitoring mode using an OBD interpreter. There are many vehicles that do not have sensors already install that can transfer important asset management data like oil pressure engine coolant temperature and vehicle speed etc. The ADC module is also implemented for the purpose of monitoring engine running hours on vehicles that auxiliary equipment.

The rationale of getting engine running hours' data to extend life cycle of the engine by implementing maintenance cycles based on running hours on the heavy duty trucks with PTO's and other auxiliary equipment. The problem is primarily the fact that these specific type vehicles have their engines running while vehicle is stationery to provide power to the auxiliary equipment that is attached to the vehicle. This creates to problems for the asset owner in terms of asset lifecycle management as the vehicle mileage is not a true reflection of the engine wear and that the fuel usage cannot be accurately managed and does not tie up to other management systems like e-fuel system that the government uses. Because of the unpredictability it leads to abuse as employees can draw more fuel than is required for their vehicles, the state loses millions of rands annually due these types of discrepancies.

The ADC Module is very essential part of the asset-tracking device but can be very resource intensive which this design cannot afford ito the number of resources that have already utilised in this design. The STM32f4 microcontroller allows for a multichannel implementation of the analogue to digital controller which essentially means that this design use one ADC controller to monitor/convert multiple analogue inputs (**see appendix B** for implementation of code of multichannel ADC convertor using DMA). This module also provides information for the state machine in the GSM module the global variables stored in this module i.e. int ign and int vspd which use to acquire the vehicle state. The data that was acquired via the ADC Module is data does not change quickly which means the timing cycles between reads can be longer which allows us to read multiple signals in one read cycle allowing us to free up valuable resources to carry out other important microcontroller tasks. The STM32f4 motherboard allows to implement multichannel ADC conversion using one ADC channel we can acquired 5 Analogue signals.

3.9 Biometric module with SPI interface.

The purpose of this module is to process the sensor data as well as panic button event and relay this data via the SPI interface to the transmitter nrf2401l of which will in turn transmit

data wirelessly to the nRF2401l receiver module, which is interfaced to the asset-tracking device. This module implements the wireless interface of the asset-tracking device.

3.9.1 Interfacing the nrf2401l Module via SPI to the Microcontroller Hardware Explained

The nrf2401l has eight pins to interface with asset tracking device namely Vcc, GND, IRQ, CE, and the four SPI-related pins (CSN, SCK, MISO, and MOSI).

- The SPI interface of the wireless module has four pins so to the asset tracking device will have four complimentary SPI pins viz, CSN, SCK, MISO, and MOSI for data transmission and reception.
- The CSN (chip select not) is basically the control pin for SPI communication, this pin is active-low, and is normally kept high. When this pin goes low, the 24L01 begins listening on its SPI port for data and processes it accordingly.
- The balance of the three SPI pins should be tied to the hardware SPI interface of the microcontroller to the same complement pins as their name suggests (SCK to SCK, MISO to MISO, and MOSI to MOSI).MOSI Master-Out Slave-In, used to shift data from the microcontroller to the device. MISO Master-In-Slave-Out, used to shift data from module to microcontroller.
- The other two pins of the nrf2401l module i.e. CE and IRQ is used to control data transmission and reception when in TX and RX modes, respectively. CE pin Chip enable pin, Active high when high the nr2401l module is either transmitting or receiving.
- IRQ is the interrupt request pin which is also is active-low. There are three internal interrupts or events that can trigger this pin to go low when they are active. Each of these bits can be masked out such that when the bit's respective interrupt becomes active, the status of the IRQ pin is not changed.
- Because SPI is serial interface for data transmission Bit/Byte order also needs to be configured to send the **Most Significant Bit(MSB) First**. Within a byte. For multiple data bytes, the **Least Significant Byte (LSB)** needs to be shifted first.

So the long and the short of the operation of the nRf2401l module interfaced to the biometric microcontroller (Arduino) is as follows: the nrf2401l module is configured to send data acquired by the ADC module of the microcontroller i.e. the biometric data at fixed intervals. Whenever data is available and the time interval is met the microcontroller will generate an interrupt bring the IRQ pin of the nRF module low instantiating an interrupt for the rf Module. Normally SPI operation follows as explained above. The other condition of instantiating an interrupt is when panic button is pressed on the biometric microcontroller.

3.8.2 Operating modes

If device is in receiving mode:

- If CE is high .Allows you to monitor the airwaves to receive packets
- If CE goes low puts device in Standby mode and no longer monitors the airwaves.

If Device in Transmitting mode:

- CE is held low except when we want to transmit a packet
- CSN is the enable pin for SPI bus. This pin is active low.
- SCK is the serial clock for SPI bus
- On the microcontroller SPI configuration CPOL and CPHA must be configured.
- SPI must be set up for 8bits
- SPI of microcontroller to be set as master and set up as slave on nRf2401l

To receive data on the nRF2401l the following must happen:

- CSN must be high to begin with
- The assert CSN low to alert nrf2401l that it will start receiving SPI data.
- This pin will stay low throughout the transaction.

The hardware is very easy to set up and the pins are self-explanatory. We will however mention and explain that this module is interfaced to the stm32f mother module via SPI interface.

The SPI interface allows the device to read/write registers, transmit data and receive data

In order to send data to or receive data from the SPI port on the 24L01, the device had to be configured to do a few things.

3.8.3 SPI Instruction set for the nRF2401

- The CSN pin on the 24L01 must be high to start out with,
- Assert the CSN pin low to alert the 24L01 that it is about to receive SPI data. Every time the device wants to send SPI commands CSN Pin needs to go low.
- Then transmit the command byte of the instruction you wish to send.
- When receiving data bytes for this instruction, send one byte to the nRF24L01 for every one byte that device wishes to get out of the nRF24L01.
- If sending the nRF24L01 data, simply send data bytes and generally don’t worry about what gets sent back to you.

- (Nordic 2016)When receiving data from the nRF24L01, it makes absolutely no difference what is contained in the data bytes that has been sent after the command byte, just so long as the correct number of them has been sent.

3.9.4 nRF2401l Data Pipes

When setting up the wireless module for communication there are a few parameters that has to be configured to be able to get that from one device to the other. The key parameters:

- Channel: which is the specific frequency channel that communication will take place on. Integers from 0 to 125(7 bits of the specific register) represent frequencies that are mapped
- Reading pipe: the reading pipe is a unique 24, 32, or 40-bit address from which the module reads data(reading and writing pipes addresses are swapped between master and slaves when setting up a single channel wireless link)
- Writing pipe: the writing pipe is a unique address to which the module writes data
- Power Amplifier (PA) level: the PA level sets the power draw of the chip and thereby the transmission power. For the purposes of this tutorial (use with the Arduino) we will use the minimum power setting.

The writing and reading pipe addresses are swapped between the two radios that are communicating with each other, as the writing pipe for each radio is the reading pipe for the other. If this is not done we would not have a successful connection.

Table 3.1 Instruction Set for nRf2401l

Instruction Name	Instruction Format [binary]	# Data Bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read registers. AAAAA = 5 bit Memory Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write registers. AAAAA = 5 bit Memory Map Address *Executable in power down or standby modes only.*
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation will always start at byte 0. Payload will be deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Used in TX mode. Write TX-payload: 1 – 32 bytes. A write operation will always start at byte 0.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, i.e. acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last sent payload. Packets will be repeatedly resent as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Scenario:

- Device wants to read the contents at specific register at a known memory address 0x10.
- First, bring CSN low and then send the command byte '**00010000**' to the 24L01. This instructs the 24L01 that the device wants to read register 0x10, which is the TX_ADDR register.
- From the table above from command "00010000"
- First three bits **000** translates to read
- Last five bits is the **10000** in binary **is 0x10** in hex.
- To sum up the instruction set bluntly, the first 3 bits of the instruction set is "**what to do"** and the next 5 bits is "**where to do"**

Then send five dummy data bytes and the 24L01 will send back the contents of the TX_ADDR register.

Finally, bring the CSN pin back high. All totalled, device will receive six bytes. When any command byte is sent, the nRF24L01 always returns the STATUS register.

After that, will have received the five bytes that are contained in the TX_ADDR register (Nordic Semiconductor Inc. 2009).

Table 3.2 Configuration Register of nRF2401L

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the **IRQ** pin 0: Reflect RX_DR as active low interrupt on the **IRQ** pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the **IRQ** pin 0: Reflect TX_DS as active low interrupt on the **IRQ** pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the **IRQ** pin 0: Reflect MAX_RT as active low interrupt on the **IRQ** pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX

3.9.5 Arduino nrf2401l Transmitter Module

Setting up this module is quiet straight forward as there are numerous libraries written for the nRf2401l module for use with the Arduino Board.

Arduino Transmitter Software:

The software module in **Appendix C** is a simulation of the biometric data that would be transmitted via nrf2401l wireless module. The device simulates the biometric data represented by reading an analogue pot value. The Arduino together with the nRf2401L module will bet set up as a transmitter in the Biometric module. Before any coding of the Arduino it is necessary to import the Nrf2401l Library to the IDE so that can be used in the code: **Appendix D.**

STM32f4 Receiver Software:

Similarly for the stm32f4 there are also libraries already written for the nRf2401lL device. All that has to done is download the library include it in the project and use the functions at will. No forgetting to interchange the address pipes between the receive and transmit modules.

Rf Module is set data rate of 2Mbps and output Power of 0dbm.

Setting up the ATD as Receiver module for Biometric Interface:

As mentioned above there are libraries written and are free available for use with the STM32f4 microcontroller. The following pseudo code is necessary in order to get the asset tracking device to operate as the receiver.

- Import the necessary libraries into the project folder.
- Include the necessary header files to be used with Biometric module.
- Set the respective Tx address and Rx Address in the receiver module.
- Initialise the buffer in the code i.e. char Bio_ dataOut[32] and char Bio_dataIn[32]
- Looking at data sheet use appropriate pin for spi transfer and DMA mapping to choose available pins for biometric module.
- Some of the radio modules haves have pins pack function this function can be called to set up the pins for the biometric module.
- Set data rate and power level of receiver in the module it is necessary to set data rate to 2mbps and power level to -18dBm.
- The while loop for the data module will look something like the code **Appendix D.**

3.10 GPS Module Implementation

This module is implemented on the Stm32f4 board to obtain GPS data for asset tracking. This module will parse NMEA sentences to the ATD device. It was be configured to send a specific type of NMEA sentence. Data received from this was be buffered and stored at GPS_buffer from where it will be serialised into another buffer for transmission by the GSM Module. C code implementation of this module is partly covered in Appendix D. The software component of this module will be basically to set up the Uart communication of the GPS receiver to the Asset tracking device. Implement a uart interrupt handler and then to parse the serial data to a buffer GPS_Buffer.

3.10.1 Implementing the GPS Module

This module is the easiest of all the modules to set up. All that is needed to do is interface the U-Blox gps module to the Asset tracking device via UART. Once this has been done, need to pass all the data from this peripheral to one of the buffers that was set up.

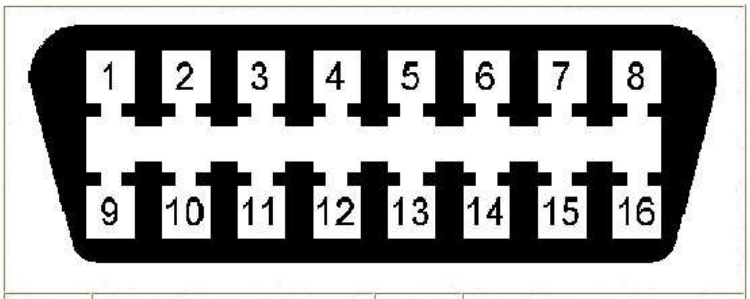
Setting up this module entails a few basic routines that we already have libraries/ functions for, the beauty of this module is that TX port is not even needed, not setting up the this TX port in the uart frees up a lot of resources for the ATD, as we will be only receiving data from this module. The data we receive at this module will be parsed to Gps_BufferRead. The setting up of the U-blox module to transmit the required sentence format can be done once-off via a terminal program using specific AT commands.

The basic steps for setting up the GPS module is.

- Set up Serial Communication interface
- Assign pins on asset tracking device for Uart
- Set up appropriate baud rate for comms we chose 38800
- There are a few finer details in terms of power requirements that has to be looked at as the U-blox module is very sensitive to the supply rail voltage.
- Once this module is initialised in the software routine
- The data from the U-blox module is sent as NMEA sentences as serial data to the Uart of the Asset tracking device.
- This data is transferred to a GPS_buffer using interrupts.
- This data is then serialised together with the other buffers by the GSM Module for transmission as UDP sentences to the Control Centre

3.11 The OBD Module

All that happens in this software module is the setup of the serial comms interface between the Obd Interpreter IC, the ELM327 and the stm32f4 discovery board. Then send a few AT commands to put the OBD module in monitoring mode to monitor the obd data of the vehicle. The obd data is transferred via interrupts to OBD_Buffer using UART. Data from OBD module will be serialised into Serial_Buffer for transmission in UDP format via the GPRS Module. This module will also handle buffer overflow of the elm327 interface.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

OBD-II Connector and Pinout

Figure 3.13 OBD connector and Pinout

In this chapter the theory, the software and hardware implementation of the obd module will be covered.

In order to implement this module, the design will need hardware to interpret the data that is received from the obd and present it as serial data to the stm32 device. In order to this we need an obd interpreter Integrated Circuit. The two most popular obd interpreter IC is the St1110 and the Elm 327 of which the details and comparison of these two are cover in chapter three.

3.11.1 Communicating with OBD Interpreter via AT Commands

AT commands are a standard in which modems are configured. At commands are the commands that the Elm327 understands and responds to; there are numerous at commands that the Elm327 responds to. These commands are there to set up the communication between the microcontroller and the vehicle. For the purpose of this research, it is intended to only monitor certain data that in turn will be buffered.

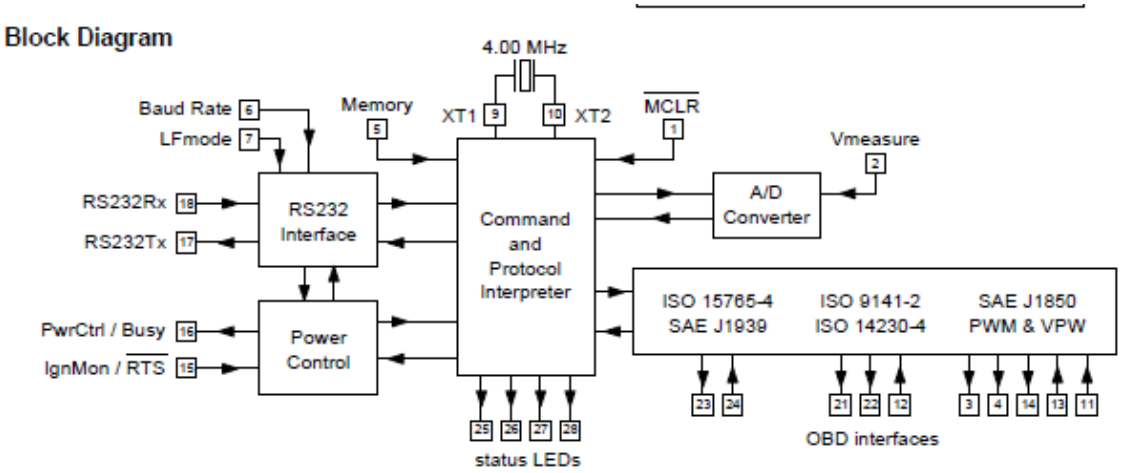


Figure 3.14 Block Diagram of OBD Interpretor

3.11.2 The Function description of the OBD Module

The elm327 and elm 329 are very popular OBD interpretor ic's and can be procured quiet easily in South Africa on the other hand the preferred obd interpretor chip the ST1110 was quite a challenge to procure so we opted to implement the OBD module with the more readily available Elm 327 obd interpretor. The data link connector shown in Figure 5.1 is a standard connector in the vehicle to which the asset tracking device will be attached/interfaced to.

The OBD II protocol interface the ELM327 shown figure 5.2 converts the data presented to it as serial data to the UART peripheral of the microcontroller whereby the Asset tracking device will buffer this data via the OBD interrupt handler to a buffer called OBD_Buffer after which sour GSM/GPRS module will serial this buffer for transmission as udp sentences. Conversion of data in text format to voltage levels from the processor to the on-board system and vice versa is performed by the ELM 327based on the AT Commands received by it from the Asset tracking device. PID data is requested and performs the processing of the data received from the ECU via the OBD II protocol interface

During the hardware interfacing of the Elm 327 ELM327 to the STM32F4 microcontroller it was necessary to use a seral convertor IC the max232 serial interface to connect **to the** asset tracking device. Most of- the- shelf ELM 327 kits have either a USB interface or a Bluetooth interface as their standard interface

The nine diagnostic modes and all PIDs are described in detail in international standard such as SAE J1979 and ISO 15031-5.

The implementation of the Obd module to the asset-tracking device can be broken down into the following stages

- The data presented to the asset-tracking device by the OBD interpreter is presented in the form of serial data as described by SAE as responses messages and PID's. at a specific baud rate.
- The OBD function in the code on the ATD device has to handle this data presented to it at specific baud rate.
- The asset-tracking device has to configure the uart port for communication to the OBD intrepretor. Create a buffer to store the data received.
- For this module, interrupts were used to handle the data acquisition from obd interpretor to the ATD's OBD_ Buffer.
- The OBD module has two main routines to take care of for us one being to extract the VIN number of the vehicle for use in the GPS tracking software as well as for asset management software. The other routine, which the ODB module will do continuously, is to monitor the OBD data. This is done by putting the obd intrepretor into monitoring mode.
- The Obd buffer of the elm 327 is not very large and in the code it required a function/routine clear the buffer of the obd intrepretor when it overflows.

3.12 Putting all Together

The heart of the asset-tracking device lies in the principle of buffering data and resending the data in a controlled format to a remote control centre. The asset-tracking device has numerous serial communication interfaces all of which were used to implement this design. Most of the modules interfaced to the microcontroller were interfaced using UART using interrupts and SPI. All of these modules have been extensively covered in the preceding chapters. This chapter predominantly focusses on the principles and theory of buffering the data as well as the different mechanism like Interrupts and DMA employed to get the data into these buffers.

In a nutshell the asset tracking device is a device that acquires data from the different modules stores this acquired data into buffers; these buffers are then serialised into one UDP sentence which is then transmitted by the GPRS interface of the asset tracking device base on the different finite states of the asset tracking device.

There are numerous variations of the theme in terms of buffering data all of which have their pro and cons but all buffering techniques endeavour to resolve a few fundamental issues which we will covered extensively in this chapter.

3.12.1 The Control Centre

The whole benefit of this design lies in the Control Centre/Remote Server and the remote Database engine. The Control server will serve four primary Functions:

- To de-serialize that incoming UDP data and store into different buffers.
- From these buffers applications will write this data into appropriate tables into Database as well as convert to appropriate file formats like KML and CSV.
- Different high-level application to use this data for different application programs.
  All maintenance management functions and Life Cycle functions are implemented at this level as all the relevant data from the primary asset gets relayed to this server for applications to use.
- All the data that was acquired from the numerous unique addressed ATD,s will be stored in the Database from where all the applications will use to drive their systems and applications like the asset management systems, e-fuel systems ,GIS systems and so on.

  For the purpose this thesis the control centre software was not implemented; all we implemented was a multi-threaded sever/client socket that was implemented in C#. This was done purely to test the ATD in terms of data sent by it to control center.

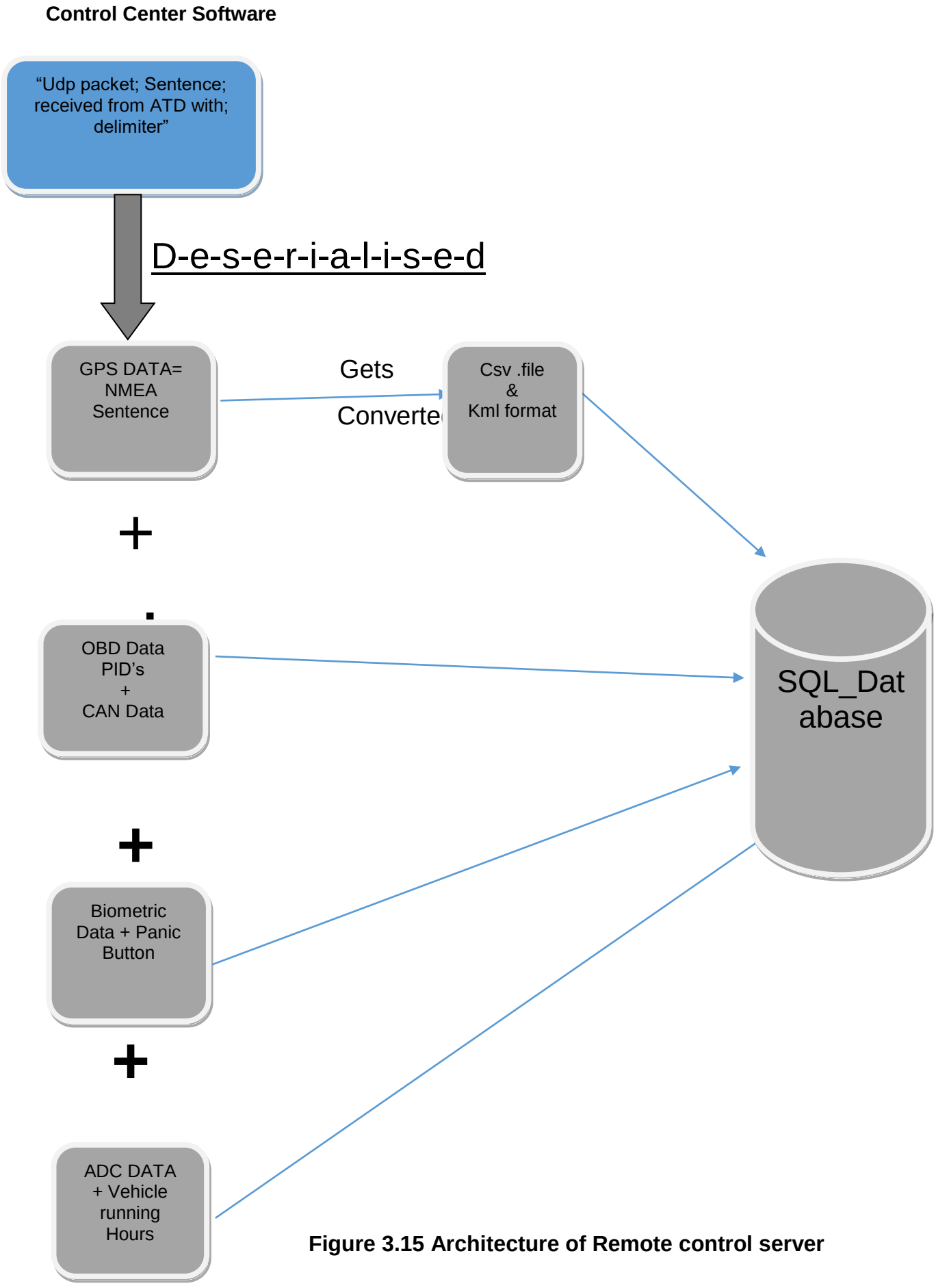


Figure 3.15 Architecture of Remote control server

3.12.2.1 Multithreaded Server Application

The remote control server center is a server that will be handling multiple ATD device clients. The software will be implemented with Multithreaded Socket Program.

The data will be received in a serialized format; after which it will be deserialized by the de-serialize program that will be local to the server in the control centre. DE-Serialization entails writing into different buffers as seen in Figure 11.1. From the data will be converted to the relevant formats for the different applications. Data will also be converted to CSV file format and bound to a SQL database. Once the data is written to a database the higher levels applications can retrieve data and write data to the SQL database. This server will host the Asset tracking software as well as host the database for the asset management software and Inter- Active Control Centre Software.

Chapter 4
Testing Results and Discussion

This chapter describes the results and findings after successful completion/building of the asset tracking device with all modules operating as simultaneously. The design was implemented on a proof of concept basis with each of the daughter modules hard-wired to the core mother module except of course for the Biometric module which is wirelessly connected to the mother module. The Asset tracking device once initialized connects to the remote control centre which for the purpose of test was multi-socket sockets application installed on a server. For the purpose of the proof of concept all that was tested for was the successful reception of the serialised data from ATD as UDP packets.
The specifics that was tested was\;

- Successful serialisation of the data into udp packets.
- The different data transmission modes based on the different acquired states, these values were simulated via pot connected to the ADC Module.
- Latency of the data transmission as well as initialisation routines was not measured and captured per say, but was observed by means of a standard stopwatch.

4.1 Testing and analysis of GPRS

During initial setup of the GSM/GPRS module the sim900 module was directly interfaced to the Stm32f4 ATD device via uart but during testing of the modules it was found that the asset tracking device had considerable latency issues when the GSM module lost connections and a lots of time was wasted to re-initialise the GSM-module, also found that GSM module engages the main processor for long periods of time causing a considerable loss of data.
For this reason and an interim solution the GSM Module was interfaced as an entirely separate unit with its own STM32f4 motherboard in other words what basically done was interface an sim900 module to STM32f4 development board and interface that development board to the Asset tracking device and interface the second stm32f4 to main stm32f4 ATD this gave us the options to share the task between these two devices, then tested the original setup to test the capabilities of the stm32f4 in terms of handling a number of different peripherals.
During the implementation of the GSM/GPRS, a few issues with regard to transmission of UDP packets was encountered but no problems with regard to transmitting TCP packets. After researching on many forums, it was realised that on the MTN Network as matter of fact all SA cell networks there is barring of UDP Packets/datagrams. In order to use this service it has to be arranged with the service provider in question by means of a formal process that could take several months. Fortunately, a fellow associate involved in telemetry products

allowed for the testing of the transmission of udp packets via their IP address(which is approved for UDP) obviously this being the simulated/dummy IP address of the remote control centre for this research.

4.2 Analysis and Interpretations of the research Conducted

Asset lifecycle is greatly impacted if real time data can be collated. Asset tracking device provides a means to collate data in an efficient and cost effective way changing the perception that data cost in terms of asset tracking could still be a distant future. We saw with the implementation of serialised UDP that data cost can be drastically reduced.

It can be seen that by serialising the data into long UDP sentences and putting the GPRS module in transparent mode that we greatly reduce packet overhead. With further improvements in design there is opportunity to implement state machines to further streamline data transmissions into more intelligent process; for example in the present design all the buffers are serialised into one buffer for transmission to control centre.

The implementation of state machine with regard to different states of the asset tracking device also result in a reduction of data cost as the state machine controls the frequency of data transmitted.

In future development data could be written to different buffers determined by different conditions and transmitted according to those conditions to optimise data transmissions and costs.

During the research it was found that IOT devices do not only impact lifecycle management but also provides a platform for other applications that could provide benefit to the asset Manager, the adding of a biometric interface demonstrated this.

The GSM Module, during the initialisation procedure, had some latency issues but once it was up and running there was no further latency issues; solution to this is to have a separate microcontroller handling the GPRS module to reduce this latency. We also appreciated the significance in the difference on how udp and tcp is handled. We also saw that there are cost differences with regard to data; in TCP applications which are mainly HTTP driven applications the cost of data is significantly higher because of header overhead; it can also be seen that in the South African context that the consumers pay for total payload whether is just one byte of payload or 80-bytes of payload. We saw that globally there is an increase in udp data and this of course can be directly related to IOT.(CAIDA n.d) There is increase in

the number software routines written to negate the shortcomings of udp in terms of reliability. It was also found that the 32-microcontroller is more than capable of handling numerous peripherals simultaneously because of the DMA capability, the nested vector interrupt controller that the stm32f4 microcontroller has. The successful implementation of the asset tracking device is proof of this capability.

During this research it was seen that that is a dramatic increase in use of the UDP protocol and this is mainly attributed to the increase in IOT applications. There is however, increase in web base applications for IOT devices, which means a need for TCP as opposed to UDP to drive these web-based solutions.

Also it is important to understand that in the present information age there is a convergence to transfer data as opposed to signals.

During literature review of this research it was found that vehicle manufacturers are making a lot more data available with the use of CAN and that very soon that OBD will be totally phased out or incorporated into CAN which will mean that more data will be made available for AI and data Analytics. It is a strong belief that if more data is made available to secondary systems that life cycle cost of assets can be extended quiet significantly. There is a lot of research currently extended to lifecycle asset management that rely on data analytics thus increasing the need for IOT devices like the asset tracking device. Devices could even monitor driver behaviour and driving patterns that affect road safety and improve on this.

One to the main objectives of this research was to determine the 32-microcontrollers ability to handle numerous peripherals without any stability or latency issues. The research found that the stm32f4 device proved to be more than capable without any complications. All latency issues we related to peripheral devices response to AT Commands issue or the response time of the network to establish a UDP/TCP connection and this was only confined to initialisation routines of the device or modules.

Chapter 5
Conclusion and Future Work

5.1 Conclusion

The primary purpose of this thesis was to implement Asset tracking device with GPRS interface so that a device can transmit data to a remote call centre for asset optimisation and application. It was the purpose of this thesis to realise an understanding with regard to the cost implications of transferring acquired data to a remote call centre. Even though that no cost analysis was conducted in this research it is irrefutable that there is direct correlation in terms of amount and frequency of data transmitted and costs incurred. It is safe to say that optimising transmission rates and reducing data overhead that this design was successful in reducing data cost, there is however no data to demonstrate the exact cost impact. This design successfully implemented a mechanism to control the frequency of data transmitted using a state machine.

Due to successful realisation of the proof of concept of the asset tracking device, it was determined that stm32f4 microcontroller achieved the objective of interfacing numerous peripheral modules simultaneously at different baud rates with excellent data latency.
In-depth understanding of the nested interrupt controller and DMA controller of the microcontroller was achieved which formed the foundation of implementing the various modules without any interrupt conflicts and incredible latency rates, this acquired knowledge can be incorporated into new designs when working with 32-bit ARM microcontrollers. The various buffering and serialisation techniques investigated also contributed significantly to the objectives of this research.

The other objective of this thesis was to accomplish this design using the barebones approach as opposed to using rapid development platforms that are available for the design applications of this nature. After implementing this design and also having being exposed to the rapid development platforms It was realised that the knowledge gained from the bare bones approach is definitely worth the effort, after all, the libraries written can be re-used in other applications and will belong to the embedded designer or asset owner but at the same time rapid development platforms have a place in quick prototyping. It was also realised in this research that time taken with the barebones approach can easily be under-estimated because not only does it requires time to write code of complex designs It also is required that embedded designer be familiar with the microcontroller using the barebones approach; however the knowledge gained is irreplaceable

5.2 Future Work

- Having being exposed to CAN at a low level, a great deal of knowledge was acquired with regard to this protocol; this understanding exposes the power of CAN and the possibilities especially with regard to the Internet of things. In this device CAN was implemented in getting vehicle data but CAN is not confined to motor vehicles its applications can be extended to the mining industry in fact to any microcontroller with CAN peripherals.
- CAN Bridge is the natural/logical progression because most microcontrollers are equipped with at least two CAN peripherals as well as the fact that all microcontroller vendors are making CAN a standard peripheral in their products. The applications born from this bridge will be incredible and can be extended to other industries like the Military, Marine and heavy duty Automotive Industries.
- Extending the asset tracking device to including video transmission which will enhance the application of the device. E.g. GPS-stamping of video images which could assist with fault reporting software applications.
- To appreciate the concept of asset tracking the control centre software can be further developed to give the full benefit with regard to asset lifecycle management.

References

Anon.1991. *CAN Specification* Version 2.0.Bosch Gmbh

Anon.1998.*Controller Area Network*. Siemens Microelectronics :CANPRES Version 2.0

Anon.2014.*ELM327DSJ*.http://www.elmelectronics.com.[17 Mar 2018].

Bangali S.A & Shah S.K Dr. 2015.*Real time School Bus Tracking System with Biometrics, GPS and GPRS Using ARM Controller*. International Journal of Advanced research in Electrical, Electronics and Instrumentation Engineering.Vol 4,Issue 8.

Barrenscheen J, Dr.1998. *Application Note Siemens AP_Note 2921*.Siemens Canpress.

Cai J & Goodman D.1997. *General Packet Radio Service in GSM*. Rutger University,IEEE Communications Magazine.

CAN in Automation. n.d. *Can Knowledge*. http://www.can-cia.org/can-knowledge/can/can-data.[28 Jan 2018]

Center for Applied Internet Data Analysis(CAIDA) n.d. *Traffic Analysis of UDP/TCP*. https://www.caida.org/research/traffic-analysis. [10 Feb 2018].

Cypress. n.d. *ADC using dma*. http://www.cypress.com//An61102/adc[2 Aug 2017].

Digi.com. *Efficient Data Transfer over Cellular Networks: White Paper.* http://www.digi.com. [Jun 2017]

Github.com n.d. *Hitex Reference Manual*. https://github.com/pmaupin/pdfrw/files/1594407/STM32HitexRefManual.pdf [12 Apr 2018]

Godavarty S, Broyles S and Parten M.(2000), *Interfacing to the On-board Diagnostic System*, Proceedings Vehicular Technology Conference, vol 4.

Gps Information.org n.d. http://www.gpsinformation.org/dale/nmea.html[12 Aug 2017]

Hanks S, Li R, Farinacci D,Traina P.2015.*Generic Routing Encapsulation (GRE).*RFC 2784.

Heath S.2002. *Embedded Systems Design*. Second Edition. Oxford Boston. Newnes.

Jenkins W.2006. *Real time performance monitoring with data integrity*, Master Thesis, Missisipi State University.

K. Tindell, A.Burns and A.J. Wellings. 1995.*Calculating Controller Area Network (CAN) Message Response Times*. Control Engineering Practice, Volume 3, Issue 8. UK.

Kurose J.F & Ross K.W. 2013. *Computer Networking :A Top Down Approach*, Sixth Edition, Pearson.

Murphy N. n.d. *Introduction to Controller Area Network (CAN)*. http://www.netrino.com/Embedded-Systems/How-To/Controller-Area-Network-CAN-Robustness. [Jul 2017]

Navet N, Song Y,Simonot-Lion F, and Wilwert C.2005. *Trends in Automotive Communication* Systems Proceedings of the IEEE,Vol.93,No 6. Pages:1204-1223.

NORDIC SEMICONDUCTOR Inc.2009. *nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0* .Norway: Nordic Semiconductor .

Passemard M. 2011. *Microcontrollers for Controller Area Network* (CAN).Atmel. htttp://www.scribd.com/doc/80457440/Can.[Jun 2018]

Perkings, C. 1996. *IP Encapsulation within IP*. IEEE Communications magazine.

Rhode & Schwarz. Training Center (2005), *Transmission Planes Ver 2.0*,http://rohde-schwarz.com.[Aug 2018].

Simcom n.d. https://simcom.ee/.../SIM900/SIM900_TCPIP_Application%20Note_V1.02.pdf [20 Apr 2018].

Simcom n.d. *SIM900_AN_TCPIP_V100*. http:// www.sim.com/wm [Jun 2018]

ST Microelectronics. 2014. *STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM®-based 32-bit MCUs*.RM0090. http://www.st.com , [2 Aug 2017].

STMicroelectronics. 2011. *Reference manual stm32f100xx advanced ARM-based 32-bit MCUs* rev. 4. RM0041.

Techtarget.com. n.d. *User Datagram Protocol.* https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol [12 Apr 2018].

Van Sickle T.2003. *Programming Microcontrollers in C*.2nd ed. Elsevier. USA: Newnes.

VisualGDB n.d. *ADC*. https://visualgdb.com/tutorials/arm/stm32/adc/ [10 Jun 2018]

Ward G. 2014.Design of a small Form Factor control system. Unpublished Masters Thesis.Virginia Commonwealth University.Virginia,USA.

Yui J. 2014. *The Definitive Guide to Arm Cortex M3 and Arm Cortex M4 Processors*.Newnes,2014

Zhang Y, Zhang Q & Jiang J. 2014. *The controller Development of Multi-Layer Parking bay based on STM32*.International Journal of Smart Home.Vol 8.pp,303-310.

Zuberi K.M. & Shink K.G.1997. *Scheduling Messages on Controller Area Network for Real-Time CIM Applications*. IEEE Transactions on Robotics and Automation. pp. 310-316.

APPENDICES

Appendix A: C Code for Multichannel ADC Module on the Stm32f4 Microcontroller

```
#include "mcu_init.h"
#include <stm32f4xx.h>
#include <stm32f4xx_adc.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_rcc.h>

#include <misc.h>



//=============================================================================
===
//   Configuring TIM3 to trigger at 2kHz which is the ADC sampling rate
//=============================================================================
===
void TIM3_Config(void)
{
  TIM_TimeBaseInitTypeDef TIM3_TimeBaseSetUp;

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

  TIM_TimeBaseStructInit(&TIM3_TimeBaseSetUp);
  TIM3_TimeBaseSetUp.TIM_Period       = (uint16_t)49; // Trigger = CK_CNT/(49+1) = 2kHz
  TIM3_TimeBaseSetUp.TIM_Prescaler    = 420;         // CK_CNT = 42MHz/420 = 100kHz
  TIM3_TimeBaseSetUp.TIM_ClockDivision = 0;
  TIM3_TimeBaseSetUp.TIM_CounterMode   = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM3, &TIM3_TimeBaseSetUp);
  TIM_SelectOutputTrigger(TIM3, TIM_TRGOSource_Update);

  TIM_Cmd(TIM3, ENABLE);
}
```

```
//=============================================================================
===
//   Configuring ADC with DMA for 5 Channels
//=============================================================================
===
void ADC_Config(void)
{
 ADC_SetUpTypeDef      ADC_SETUP;
 ADC_COMMONSetInitTypeDef ADC_COMMONSET;
 DMA_SetUPTypeDef      DMA_SETUP;
          GPIO_SETUPTypeDef     GPIO_SETUP;
          NVIC_SETUPTypeDef     NVIC_SETUP;

 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);



 RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

 DMA_SETUP.DMA_Channel = DMA_Channel_0;
 DMA_SETUP.DMA_PeripheralBaseAddr = (uint32_t)ADC1_RDR;
 DMA_SETUP.DMA_Memory0BaseAddr    = (uint32_t)&ADC_Raw[0];
 DMA_SETUP.DMA_DIR              = DMA_DIR_PeripheralToMemory;
 DMA_SETUP.DMA_BufferSize       = 5*4;
 DMA_SETUP.DMA_PeripheralInc     = DMA_PeripheralInc_Disable;
 DMA_SETUP.DMA_MemoryInc         = DMA_MemoryInc_Enable;
 DMA_SETUP.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
 DMA_SETUP.DMA_MemoryDataSize     = DMA_MemoryDataSize_HalfWord;
 DMA_SETUP.DMA_Mode             = DMA_Mode_Circular;
 DMA_SETUP.DMA_Priority         = DMA_Priority_High;
 DMA_SETUP.DMA_FIFOMode          = DMA_FIFOMode_Disable;
 DMA_SETUP.DMA_FIFOThreshold     = DMA_FIFOThreshold_HalfFull;
 DMA_SETUP.DMA_MemoryBurst       = DMA_MemoryBurst_Single;
 DMA_SETUP.DMA_PeripheralBurst    = DMA_PeripheralBurst_Single;
```

```
 DMA_Init(DMA2_Stream0, &DMA_SETUP);
 DMA_Cmd(DMA2_Stream0, ENABLE);


NVIC_SETUP.NVIC_IRQChannel = DMA2_Stream0_IRQn;
NVIC_SETUP.NVIC_IRQChannelPreemptionPriority =0;
NVIC_SETUP.NVIC_IRQChannelSubPriority =0;
NVIC_SETUP.NVIC_IRQChannelCmd =ENABLE;
NVIC_Init(&NVIC_SETUP);


GPIO_SETUP.GPIO_Pin = GPIO_Pin_0 |GPIO_Pin_1 |GPIO_Pin_2 |GPIO_Pin_3 |GPIO_Pin_4;
GPIO_SETUP.GPIO_Mode = GPIO_Mode_AN;
GPIO_SETUP.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOC, &GPIO_SETUP);



 ADC_COMMONSET.ADC_Mode           = ADC_Mode_Independent;
 ADC_COMMONSET.ADC_Prescaler      = ADC_Prescaler_Div2;
 ADC_COMMONSET.ADC_DMAAccessMode   = ADC_DMAAccessMode_Disabled;
 ADC_COMMONSET.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
 ADC_COMMONSetInit(&ADC_COMMONSET);

 ADC_SETUP.ADC_Resolution         = ADC_Resolution_12b;
 ADC_SETUP.ADC_ScanConvMode       = ENABLE;
 ADC_SETUP.ADC_ContinuousConvMode  = ENABLE; // ENABLE for max ADC sampling frequency
 ADC_SETUP.ADC_ExternalTrigConvEdge                                                    =
ADC_ExternalTrigConvEdge_Rising;//ADC_ExternalTrigConvEdge_Rising;
 ADC_SETUP.ADC_ExternalTrigConv    = ADC_ExternalTrigConv_T3_TRGO;
 ADC_SETUP.ADC_DataAlign          = ADC_DataAlign_Right;
 ADC_SETUP.ADC_NbrOfConversion     = 5;
          //ADC_SETUP.ADC_NbrOfChannel       = 5;
 ADC_SetUp(ADC1, &ADC_SETUP);

 ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_15Cycles);
```

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2, ADC_SampleTime_15Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 3, ADC_SampleTime_15Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_13, 4, ADC_SampleTime_15Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 5, ADC_SampleTime_15Cycles);




  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);      // (for testing)
  ADC_DMACmd(ADC1, ENABLE);
  ADC_Cmd(ADC1, ENABLE);
  ADC_SoftwareStartConv(ADC1);
  //ADC_TempSensorVrefintCmd(ENABLE);
}
```

Appendix B: Implementation C code for GPS Module

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_usart.h"
#include "string.h"
#include "stdio.h"

#define LENGHT 7
#define MAX_LENGHT 5
#define BUFFER_SIZE 8


enum BufferState{BUFFER_OK,BUFFER_EMPTY,BUFFER_FULL};
struct Buffer{
        uint8_t data[BUFFER_SIZE];
        uint8_t new_idx;
        uint8_t old_idx;
};
struct Buffer buffer;
volatile struct Buffer GPS_Buffer ={{0},0,0} ;
volatile struct Buffer OBD_Buffer ={{0},0,0} ;

enum BufferState bufferWrite(volatile struct Buffer*buffer,uint8_t byte);
enum BufferState bufferRead(volatile struct Buffer*buffer,uint8_t *byte);
enum BufferState bufferPeek(volatile struct Buffer *buffer,uint8_t *byte);



void GPIOD_INIT();
        void USART_Config(void);
void USART2_Config();
        void USART_PutChar(char c);
        void USART2_PutChar(char c);
```

```
        void USART_PutString(char *s);
        void USART2_PutString(char *s);

        uint16_t USART_GetChar();
        uint16_t USART2_GetChar();
        void  USART1_IRQHandler(void);
        void  USART2_IRQHandler(void);
        unsigned char receive_data(char *a,int max);
        char* USART_ReceiveString(char* String);
void get(char *a,int n);
int getchara(void);

uint16_t USART_GetChar();

        volatile uint16_t chi ;

        char incoming_byte;
        char received_byte;
        char *data;

char dat[LENGHT];
volatile char  receive_str[MAX_LENGHT+1];


int main(void)

{

                enum BufferStatus status;

 //GPIO_SETUPTypeDef GPIO_SetUp;


                // Enable clock for GPIOD (for orange LED)
```

```
GPIOD_INIT();

        // Call USART1 configuration
        USART_Config();
        USART2_Config();

        USART1_IRQHandler();
        USART2_IRQHandler();
        USART2_PutString("AIDON!\n");
                        //USART_PutString(data);
                        //get(data,LENGHT);

                        //char *name;
                         //usart1_isr();



        while (1)
        {
                        status=bufferRead(&OBD_Buffer,&received_byte);
                        if(status == BUFFER_OK){

                                        if((strcmp((char*) received_byte,"hello"))==0)


          {
                       GPIO_SetBits(GPIOD, GPIO_Pin_13);
          }
         // else
          {
                      GPIO_ResetBits(GPIOD, GPIO_Pin_13);


          }

                                }
```

```
        }

            }

void GPIOD_INIT(void)
{
    GPIO_SETUPTypeDef GPIO_SetUp;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

      // Initialization of GPIOD (for orange LED)

      GPIO_SetUp.GPIO_Pin = GPIO_Pin_13;
      GPIO_SetUp.GPIO_Mode = GPIO_Mode_OUT;
      GPIO_SetUp.GPIO_OType = GPIO_OType_PP;
      GPIO_SetUp.GPIO_PuPd = GPIO_PuPd_NOPULL;
      GPIO_SetUp.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_Init(GPIOD, &GPIO_SetUp);

}
    void USART_Config(void)
    {
        GPIO_SETUPTypeDef GPIO_SETUPSetUp;
        USART_InitTypeDef USART_InitSetUp;
        NVIC_SETUPTypeDef NVIC_SETUPStruct;

      // Enable clock for GPIOB
      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
      // Enable clock for USART1
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);



      // Connect PB6 to USART1_Tx
      GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);
      // Connect PB7 to USART1_Rx
      GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);
```

```
        // Initialization of GPIOB

        GPIO_SETUPSetUp.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
        GPIO_SETUPSetUp.GPIO_Mode = GPIO_Mode_AF;
        GPIO_SETUPSetUp.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_SETUPSetUp.GPIO_OType = GPIO_OType_PP;
        GPIO_SETUPSetUp.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_SETUP(GPIOB, &GPIO_SETUPSetUp);

        // Initialization of USART1

        USART_InitSetUp.USART_BaudRate = 9600;
        USART_InitSetUp.USART_HardwareFlowControl =
             USART_HardwareFlowControl_None;
        USART_InitSetUp.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
        USART_InitSetUp.USART_Parity = USART_Parity_No;
        USART_InitSetUp.USART_StopBits = USART_StopBits_1;
        USART_InitSetUp.USART_WordLength = USART_WordLength_8b;
        USART_Init(USART1, &USART_InitSetUp);

        // Enable USART1
        USART_Cmd(USART1, ENABLE);
                        //Enable the USART RX Interrupt
                        USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
                        NVIC_SETUPStruct.NVIC_IRQChannel=USART1_IRQn;
                        NVIC_SETUPStruct.NVIC_IRQChannelPreemptionPriority=0;
                        NVIC_SETUPStruct.NVIC_IRQChannelSubPriority=0;
                        NVIC_SETUPStruct.NVIC_IRQChannelCmd=ENABLE;
                        NVIC_SETUP(&NVIC_SETUPStruct);

    }

void USART2_Config(void)
    {
            GPIO_SETUPTypeDef GPIO_SETUPSetUp;
```

```
        USART_InitTypeDef USART2_InitSetUp;
        NVIC_SETUPTypeDef NVIC2_InitSetUp;

// Enable clock for GPIOB
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
// Enable c1ock for USART1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);



// Connect PA9 to USART2_Tx
GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);
// Connect PA10 to USART2_Rx
GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);

// Initialization of GPIOB

GPIO_SETUPSetUp.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
GPIO_SETUPSetUp.GPIO_Mode = GPIO_Mode_AF;
GPIO_SETUPSetUp.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_SETUPSetUp.GPIO_OType = GPIO_OType_PP;
GPIO_SETUPSetUp.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_SETUPSetUp);

// Initialization of USART2

USART2_InitSetUp.USART_BaudRate = 9600;
USART2_InitSetUp.USART_HardwareFlowControl =
     USART_HardwareFlowControl_None;
USART2_InitSetUp.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART2_InitSetUp.USART_Parity = USART_Parity_No;
USART2_InitSetUp.USART_StopBits = USART_StopBits_1;
USART2_InitSetUp.USART_WordLength = USART_WordLength_8b;
USART_Init(USART2, &USART2_InitSetUp);

// Enable USART2
```

```
    USART_Cmd(USART2, ENABLE);
                    //Enable the USART RX Interrupt
                    USART_ITConfig(USART2,USART_IT_RXNE,ENABLE);
                    NVIC2_InitSetUp.NVIC_IRQChannel=USART2_IRQn;
                    NVIC2_InitSetUp.NVIC_IRQChannelPreemptionPriority=0;
                    NVIC2_InitSetUp.NVIC_IRQChannelSubPriority=0;
                    NVIC2_InitSetUp.NVIC_IRQChannelCmd=ENABLE;
                    NVIC_Init(&NVIC2_InitSetUp);

                }
//The second function is USART_PutChar(). This function is to send a character.
        void USART_PutChar(char c)
        {
          // Wait until transmit data register is empty
          while (!USART_GetFlagStatus(USART1, USART_FLAG_TXE));
          // Send a char using USART1
          USART_SendData(USART1, c);
                USART_ClearITPendingBit(USART1,USART_IT_TC);

        }
        void USART2_PutChar(char c)
        {
                  USART_SendData(USART2, c);
                USART_ClearITPendingBit(USART2,USART_IT_TC);

        }

        void USART_PutString(char *s)
        {
          // Send a string
          while (*s)
          {
             USART_PutChar(*s++);
          }
        }
        void USART2_PutString(char *s)
```

```
        {
          // Send a string
          while (*s)
          {
            USART2_PutChar(*s++);
          }
        }
//The last function  is USART_GetChar(). This function is for read a character.




        void usart1_IRQHandler(void){

                //int status;
                if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
                {
                        USART_ClearITPendingBit(USART1,USART_IT_RXNE);
                //while(ch != '\r' || ch !='\n')
                        chi =(uint8_t)USART_ReceiveData(USART1);

                        {

                                USART_ITConfig(USART1 , USART_IT_TXE , ENABLE);


                                        }if(USART_GetITStatus(USART1 , USART_IT_TXE) != RESET)
{
uint8_t data;

USART_SendData(USART1 , data);


USART_ITConfig(USART1 , USART_IT_TXE , DISABLE);

}
```

```
	}
}
	void get(char *a,int n)
{
	int i=0,c;

	while((c=USART_GetChar())!='\n'&& i<(n-1))
		a[i++]=c;
	if(i<n)
	{
		a[i++]='\n';
		//a[i]='\0';
	}
	else
		for (i=0;i<n;i++)

	a[i]=0;
}

		void USART1_IRQHandler(void) {



			if(USART_GetITStatus(USART1,USART_IT_RXNE))
			{
				//static int cnt=0;
									char chi=USART1->DR;
				bufferWrite(& GPS_Buffer,chi);
				USART_ClearITPendingBit(USART1, USART_IT_TXE);

				// volatile char receive_str[MAX_LENGHT+1];
			}
if(USART_GetITStatus(USART1 , USART_IT_TXE) != RESET)
{
uint8_t datr;
 bufferRead (&GPS_Buffer , &datr);
```

```
USART_PutString((char*)&datr);

}
else {

USART_ITConfig(USART1 , USART_IT_TXE , DISABLE);

}

                }


                void USART2_IRQHandler(void) {



                        if(USART_GetITStatus(USART2,USART_IT_RXNE))
                        {
                                //static int cnt=0;
                                                                char chi=USART2->DR;
                                bufferWrite(&OBD_Buffer,chi);
                                USART_ClearITPendingBit(USART2, USART_IT_TXE);

                                // volatile char receive_str[MAX_LENGHT+1];
                        }
if(USART_GetITStatus(USART2, USART_IT_TXE) != RESET)
{
uint8_t datr;

 bufferRead (&OBD_Buffer , &datr);
USART2_PutString((char*)&datr);

}
else {
```

```
USART_ITConfig(USART2 , USART_IT_TXE , DISABLE);

}

                }

                uint16_t USART_GetChar()
        {
           // Wait until data is received
           while (!USART_GetFlagStatus(USART1, USART_IT_RXNE));
           // Read received char
           return USART_ReceiveData(USART1);
        }
        uint16_t USART2_GetChar()
        {
           // Wait until data is received
           while (!USART_GetFlagStatus(USART2, USART_IT_RXNE));
           // Read received char
           return USART_ReceiveData(USART2);
        }
        enum BufferStatus bufferWrite(volatile struct Buffer*buffer,uint8_t byte)
        {
                uint8_t next_idx=(((buffer->new_idx)+1)%BUFFER_SIZE);
                if(next_idx==buffer->old_idx){
                        return BUFFER_FULL;
                }
                buffer->data[buffer->new_idx]=byte;
                buffer->new_idx=next_idx;
                return BUFFER_OK;
        }
        enum BufferStatus bufferRead(volatile struct Buffer*buffer,uint8_t *byte)
        {

                if(buffer->new_idx==buffer->old_idx){
                        return BUFFER_EMPTY;
```

```
                }
                *byte =buffer->data[buffer->old_idx];
                buffer->old_idx=((buffer->old_idx+1)%BUFFER_SIZE);


                return BUFFER_OK;
        }

enum BufferStatus bufferPeek(volatile struct Buffer *buffer,uint8_t *byte)
{
        uint8_t last_index=((BUFFER_SIZE +(buffer->new_idx)-1)%BUFFER_SIZE);
        if (buffer->new_idx ==buffer->old_idx)
        {return BUFFER_EMPTY;}
        *byte =buffer->data[last_index];
        return BUFFER_OK;

}
```

Appendix C: Pseudo code and state machine to implement the different GSM Modes

```
enum monitorState {VEH_DORMANT,VEH_IDLING,VEH_MOVING};
enum monitorState AcqVehState(int a,int b);
void GSM();
void GSM_Mode1();
void GSM_Mode2();
void GSM_Mode3();

enum monitorState veh_status;
int ign=0,vspd=0;

int main (void)
{
   GSM_Init();
            while(1)
            {
                        Veh_status = AcqVehState(ign,vspd);
                        GSM();
            }
   return 0;
}
void GSM(void)
{
            switch(veh_status)
            {
            case VEH_DORMANT:
            {
                        GSM_Mode1();
            }
            break;
            case VEH_IDLING:
            {
                        GSM_Mode2();
            }
            break;
            case VEH_MOVING:
```

```
		{
			GSM_Mode3();
		}
		break;



		}
}
void GSM_Mode1()
{
   //*Will serialise the data from the just the ADC_Buffer and GPS_Buffer. In this mode data is transmitted the less frequently*//
}

void GSM_Mode2()
{
//*Will serialise the data from all the buffers ie The ADC_Buffer,GPS_Buffer,OBD_Buffer and the BIOMETRIC_Buffer. In this mode data is transmitted the more frequently than GSM_Mode1 and Less frequently than GSM_Mode3 *//



}



void GSM_Mode3()
{
   //*Will serialise the data from all the buffers ie The ADC_Buffer,GPS_Buffer_OBD,Buffer and the BIOMETRIC_Buffer. In this mode data is transmitted the most frequently*//

}

enum monitorState AcqVehState(int a, int b){

		if(ign>=9 && vspd >=0)
		{
			return VEH_MOVING;
		}
```

```
        else if(ign>=9 && vspd<=0)
        {
                return VEH_IDLING;
        }
        else
        {
                return VEH_DORMANT;

        }

}
```

Explaining the pseudo code above we can see that we made use of some very clever coding tricks that saves an incredible amount of memory and microcontroller resources.

The crux of the pseudo code is as follows:

- Declared an enum monitorState {VEH_DORMANT,VEH_IDLING,VEH_MOVING};

Internally the compiler is substituting a zero for VEH_DORMANT,a one for VEH_IDLING and a two for VEH_MOVING.

- The return type for the Function enum monitorState AcqVehState(int a,int b) is enum monitorState which means whenever we call this function we should expect an enum returned which invariably means we have monitored the vehicle states which can be used in the state machine.
- void GSM(enum s) is the function that handle the state machine. The switch condition switch (veh_status) test the different states and implements accordingly.
- GSM_Mode1(),GSM_Mode2()and GSM_Mode3() are functions that implement the different transmission modes of the GSM module.

The GSM Module is also responsible for serialisation of data from all the different buffers. Also based on the above GSM state machine the data will also be serialised differently based on vehicle states. For clarity say that the vehicle is in Vehicle_Dormant state only GPS_Buffer and ADC_Buffer will be serialised for transmission.

Appendix D: Biometric Software Module Code

Pseudo code for Arduino is as below

- Include the necessary header files like SPI.H and RF24.h
- Configure and Initialise SPI and Radio module

  //SCK -> 13//MISO -> 12//MOSI -> 11//CSN -> 7//CE -> 8

  RF24 radio(8,7);

- Assign address pipe for Transmission

  const uint64_t tx_pipe_address = 0xE8E8F0F0E1LL;

- Define the necessary variables and arrays to hold the data

- In setup routine will call the necessary function first put module in transmitting mode by calling

  Radio.begin();

  Radio.openWrintingPipe(tx_pipe_address);

- The main loop will like:

```
void loop(void)
{
    val = analogRead(sim_bio);          /* read value from analogue
    pot i.e simulation of biometric data*/

    val = map(val, 0, 1023, 0, 179);    /*convert */
    tx_Buffer[0] = val;                 /*write value to tx_Buffer*/
    radio.write(tx_Buffer, 1);
}
```

```
while (1) {
      /* If data is ready on wireless biometric module */
      if (Bio_Wireless_DataReady()) {
         /* Get data from NRF24L0_Bio+ */
         NRF24L01_Bio_GetData(dataIn);


        Bio_NRF24L01_Transmit(Bio_dataIn);

         /* Start send  turn tx_LED On*/
         Bio_LedOn(LED_GREEN);
         /* Wait for data to be sent */
         do {
            txStatus = Bio_Wireless_GetTransmissionStatus();
         } while (txStatus == Bio_Wireless_Transmit_Status_Sending);
         /* Send done */
         Bio_LedOff(LED_GREEN);

         /* Go back to RX Mode */
         Bio_Wireless_PowerUpRx();
      }
```