

**EMBEDDED HTTP WEB SERVER USING FREESCALE FREEDOM PLATFORM**

By

TUN AMSYAR HAZIQ BIN MOHAMAD SUBHI

13869

Dissertation Submitted in Partial Fulfilment of

The Requirements for the

Degree Bachelor of Engineering (Hons)

(Electrical and Electronics)

MAY 2014

Universiti Teknologi PETRONAS  
Bandar Seri Iskandar  
31750 Tronoh  
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

**EMBEDDED HTTP WEB SERVER USING FREESCALE FREEDOM PLATFORM**

By

TUN AMSYAR HAZIQ BIN MOHAMAD SUBHI

13869

A project dissertation submitted to the  
Electrical and Electronics Programme  
Universiti Teknologi PETRONAS  
in partial fulfilment of the requirement for the  
BACHELOR OF ENGINEERING (Hons)  
(ELECTRICAL AND ELECTRONICS)

Approved by,

---

(Dr. Mohd Zuki Bin Yusoff)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

MAY 2014

## CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

---

(TUN AMSYAR HAZIQ BIN MOHAMAD SUBHI)

## **ABSTRACT**

Through the advancement of microprocessors, embedded web servers are applicable to assist humans in increasing productivity in terms of remote access and control. An embedded web server is the operation of a single chip to communicate with users via Ethernet or wireless connection to the internet without a PC as the medium. The Freescale Freedom Platform employs an ARM Cortex M0+ that has potential in embedded systems. This project serves an investigation on the Freedom Platform utilizing an ARM microprocessor as an embedded web server that functions as a remote data monitoring peripheral by means of a web browser as the human machine interface. A simple HTTP web page was created using HTML as the user interface to display the data.

## **ACKNOWLEDGEMENTS**

I would like to dedicate this page to acknowledge the persons involved, whether directly or indirectly, for the assistance, guidance and support from project supervisor, lecturers, family and fellow colleagues throughout the process of my dissertation.

I would like to express my deepest gratitude to my project supervisor, Dr. Mohd Zuki bin Yusoff for giving me the opportunity to undergo this project under his supervision as well as providing the guidelines throughout the duration of the project. I would also like to thank Dr Likun Xia and Dr Ho Tatt Wei for their participation in providing feedback and motivation to further progress in my project.

With this opportunity, I would like to thank my family for their encouraging support for me to push forward and to try as hard as I can to complete my project. Not to forget thanks to my colleagues for their contribution to the progression of the project.

<b>CERTIFICATION</b> .....	ii
<b>ABSTRACT</b> .....	iv
<b>ACKNOWLEDGEMENTS</b> .....	v
<b>LIST OF FIGURES</b> .....	vii
<b>LIST OF TABLES</b> .....	ix
<b>LIST OF ABBREVIATIONS</b> .....	x
<b>CHAPTER 1: INTRODUCTION</b>	
1.1 Background .....	1
1.2 Problem Statement .....	2
1.3 Objectives.....	3
1.4 Scope of Study .....	3
1.5 Relevancy of the Project.....	4
1.6 Feasibility of the Project.....	4
<b>CHAPTER 2: LITERATURE REVIEW AND THEORY</b>	
2.1 Embedded Web Server.....	6
2.2 Freescale Freedom Platform.....	8
2.3 ARM Processor.....	10
<b>CHAPTER 3: METHODOLOGY</b>	
3.1 Research Methodology.....	11
3.2 Prototype Designing.....	12
3.2.1 Hardware .....	12
3.2.2 Hardware Interfacing.....	14
3.2.3 Firmware Designing.....	15
3.3 Project Work.....	16
3.3.1 Project Design.....	16
3.3.2 Integrated Development Environment (IDE) .....	18
3.3.3 Processor Expert CodeWarrior.....	19
3.3.4 Wi-Fi Shield.....	21
3.3.5 KL26Z Programming.....	22
3.3.6 KL26Z Debugging.....	23
3.3.7 Serial Peripheral Interface (SPI) .....	24

3.3.8 Code Synthesis.....	26
3.3.9 Web Server Page Design.....	30
3.4 Key Milestones.....	32
3.4.1 Background Study.....	32
3.4.2 Project Design.....	32
3.4.3 Project Implementation.....	33
3.4.4 Documentation and Report.....	33
3.4 Gantt Chart.....	34
3.5 Hardware and Tools.....	35
<b>CHAPTER 4: RESULTS AND DISCUSSION</b>	
4.1 Wi-Fi Shield Library .....	36
4.2 CooCox Peripheral Library.....	37
4.3 Temperature Sensor Module.....	37
4.4 Results .....	38
4.4 Constraints and Problems Encountered.....	40
<b>CHAPTER 5: CONCLUSION AND RECOMMENDATION</b>	
5.1 Recommendation.....	42
5.2 Conclusion.....	43
<b>REFERENCES.....</b>	<b>44</b>
<b>APPENDICES.....</b>	<b>45</b>

## LIST OF FIGURES

1. FIGURE 2.1: Embedded Web Server System Architecture.....	7
2. FIGURE 2.2: FRDM-KL26Z Block Diagram.....	8
3. FIGURE 2.3: 64 PINS OF THE FRDM KL-26Z.....	9
4. FIGURE 2.4: ARM CORTEX M0+ Block Diagram.....	10
5. FIGURE 3.1: METHODOLOGY FLOW CHART.....	11
6. FIGURE 3.2: FRDM-KL26Z.....	12
7. FIGURE 3.3: Temperature Sensor Module.....	13
8. FIGURE 3.4: Temperature Sensor Module Diagram.....	13
9. FIGURE 3.5: Freedom Board I/O with Arduino Pin out Reference.....	14
10. FIGURE 3.6: Block Diagram of Embedded Web Server.....	15
11. FIGURE 3.7: Hardware Connections.....	16
12. FIGURE 3.8: Embedded Web Server Process Structure.....	17
13. FIGURE 3.9: Setting Up a New Project.....	18
14. FIGURE 3.10: Selecting KL26Z From Menu Selection.....	19
15. FIGURE 3.11: Header Files and Start-up Codes Generated by Processor Expert.....	20
16. FIGURE 3.12: Components Library of Processor Expert.....	20
17. FIGURE 3.13: Diagram of WizFi Shield.....	21
18. FIGURE 3.14: Block Diagram of OpenSDA.....	23
19. FIGURE 3.15: Selecting OpenSDA Connection.....	23
20. FIGURE 3.16: SPI Mode 0 Timing Diagram.....	25
21. FIGURE 3.17: Simple Web Server Page.....	31
22. FIGURE 4.1: Output Values from Temperature Sensor.....	37
23. FIGURE 4.2: KL26Z with Temperature Sensor.....	38
24. FIGURE 4.3: Systems Initialization Message.....	39
25. FIGURE 4.4: Default Interrupt Handler.....	41



## LIST OF TABLES

1. TABLE 3.1: Temperature Sensor Module.....	13
2. TABLE 3.2: WizFi Shield Technical Specifications.....	14
3. TABLE 3.3: Gantt Chart.....	34
4. TABLE 3.4: Hardware Description.....	35
5. TABLE 3.5: Software Description.....	35

## **LIST OF ABBREVIATIONS**

ADC Analogue-Digital Converter

ARM Advanced RISC Machines

CMSIS Cortex Microcontroller Software Interface Standard

CPU Central Processing Unit

FSM Finite State Machine

FYP Final Year Project

GUI Graphical User Interface

HMI Human-Machine Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IP Internet Protocol

kB Kilobyte

PC Personal Computer

RISC Reduced Instruction Set Computer

SPI Serial Peripheral Interface

SRAM Static Random-Access Memory

TCP Transmission Control Protocol

USB Universal Serial Bus

# CHAPTER 1

## INTRODUCTION

### 1.1 Background

Embedded web server or HTTP Server is a web server that is embedded on to an embedded system that has limited computing capabilities but is efficient enough to carry out a specific function especially in terms of remote processing or data management. In recent years, the development of Advance RISC Machine (ARM) chipsets have been the preferred microprocessors for embedded server implementations. The implementation of embedded web servers provides advantages in terms of cost due to its requirements which is less than the standard PC server in both hardware and software wise[1].

The Freescale Freedom Platform utilizes a 32-bit ARM Cortex-M0 with 48 MHz operation. The Freedom Platform is designed to operate at low power and has multiple on board peripherals such as GPIO, UART, I2C and SPI. It is also sold to the market at a relatively low price making it affordable for developers. The high end design of the Freedom Platform makes it suitable to operate a web server along with its compatibility of external peripherals such as LCD, Ethernet and Wi-Fi. This feature further enhances the capability of the web server to be controlled remotely in data gathering or process control.

The ARM Cortex M microprocessor group was introduced in the year 2004 and since then has developed variants of the Cortex M specifically for embedded microcontrollers. Particularly, the ARM Cortex-M0 processor was designed to cater mostly for embedded systems to operate with real-time I/O hardware[2]. This project aims to utilize and explore the potential of ARM based microcontrollers, specifically the Freedom Platform, in remote access and data monitoring that can be applied in the industrial and commercial sector.

## **1.2 Problem Statement**

Data acquirement is essential for certain industrial processes. Measured data such as temperature and pressure reading can be quite tedious and counterproductive when collecting manually. For example, in certain industries it is required for the company to send their employees to sites or stations to collect the measured data. Some of which are located far away. This would cost travel expenditures as well as precious time. Eventually it would result in decrease of productivity.

It is known fact that some industrial companies still use conventional equipment in data measurement such as a mercury thermometer rather than a state of the art equipment to save the company some money. However, such conventional ways of measuring data are susceptible to contamination of human error. With inaccurate data, it would eventually effect the company's business.

Through today's advancement in wireless technology as well as embedded systems, an operator would be able to acquire data by means of an embedded web server via a web browser as the human-machine interface. By means of the embedded web server, data is able to be obtained through the internet from a remote location. Automated data collection is also possible through the programmed ARM microcontroller unit along with the required peripherals to establish internet connectivity.

### **1.3 Objectives**

The objective of this project is to design and implement a web server using the ARM based Freedom Platform board in data collection. The particular model that shall be used in this project is the FRDM-KL26Z which is an ARM M0+ Cortex Core. The connectivity of choice for remote access is by means of Wireless Local Area Network (WLAN). This means the board would be using an external Wi-Fi peripheral to enable internet connectivity. This project also serves as exploration in prototyping with the Freescale Freedom Platform. To study and explore the potential of the Freedom Platform in pervasive systems. The Freedom Platform is relatively new and is becoming a learning platform for universities abroad studying embedded systems.

The concept of data procurement via embedded web server can be implemented in many real world scenarios. For example, the embedded web server can be attached to the blood pressure equipment to transmit the patients' blood pressure readings at the time to the nurse's station. The data collected would be up to date and increases the efficiency in data collecting. The overall objectives are:

- To remotely acquire measured data from peripherals used by the Freedom Platform
- To implement an embedded web server using the ARM based Freescale Freedom Platform particularly FRDM-KL26Z
- To introduce a potential data procurement concept that can be applied in real world applications

### **1.4 Scope of Study**

In this project, the study of the implementation and design of ARM based embedded web servers through website and journals as the fundamentals of building the idea for the prototype. To study other works in order to create an improvement of the previous study had done.

Based on ideas and suggestions, the hardware and software procurement such as Wi-Fi Shield, USB cables and software compiler. After acquiring the necessary equipment, the prototype is designed and built according the proposed plan.

Once the prototype has been built, testing and debugging is to be done in order for the prototype to meet the minimum requirements. In the end, the prototype should be able:

- Display data from input to the web site or Human Machine Interface
- To be controlled from remote access
- Automatically collect data periodically
- Data collected from the sensors

### **1.5 Relevancy of the Project**

Recent technology these days focuses more on multitasking and the ability of accomplishing work at just the end of your fingertips. These are necessary in order to accommodate the ever face-paced working environment and further increase efficiency. By means of remote controlled and monitoring peripherals, one would be able to achieve a higher efficiency at multitask and work done. Along with today's wireless connectivity, this project would introduce a data monitoring concept by means a simple human-machine interface where data can be collecting remotely without having to travel to on-site location.

### **1.6 Feasibility of the Project**

In today's industrial sector, data gathering is still done manually, means operators would have to travel to site location and gather the required readings of meters or any measured parameters. Despite today's technological advancement in pervasive systems as well as wireless technology.

This project serves to prove the capabilities of embedded systems has to offer in the industry sector especially in remote data gathering. By means of an embedded web server that enables remote connection with a peripheral to collect data through a simple web browser as the primary human-machine interface.

Today's ARM processors have evolved in such a way that it has been used in today's smartphones and tablets. The ARM processor was chosen as the base of the project for its relatively low cost price without compromising its quality and ability in multitasking processes.

Through the embedded web server and a wireless network, an operator would be able to collect measured data parameters from anywhere with just a ping of an address. This would increase in efficiency of the operator for they would be able to do more tasks at any given time.

## CHAPTER 2

### LITERATURE REVIEW AND THEORY

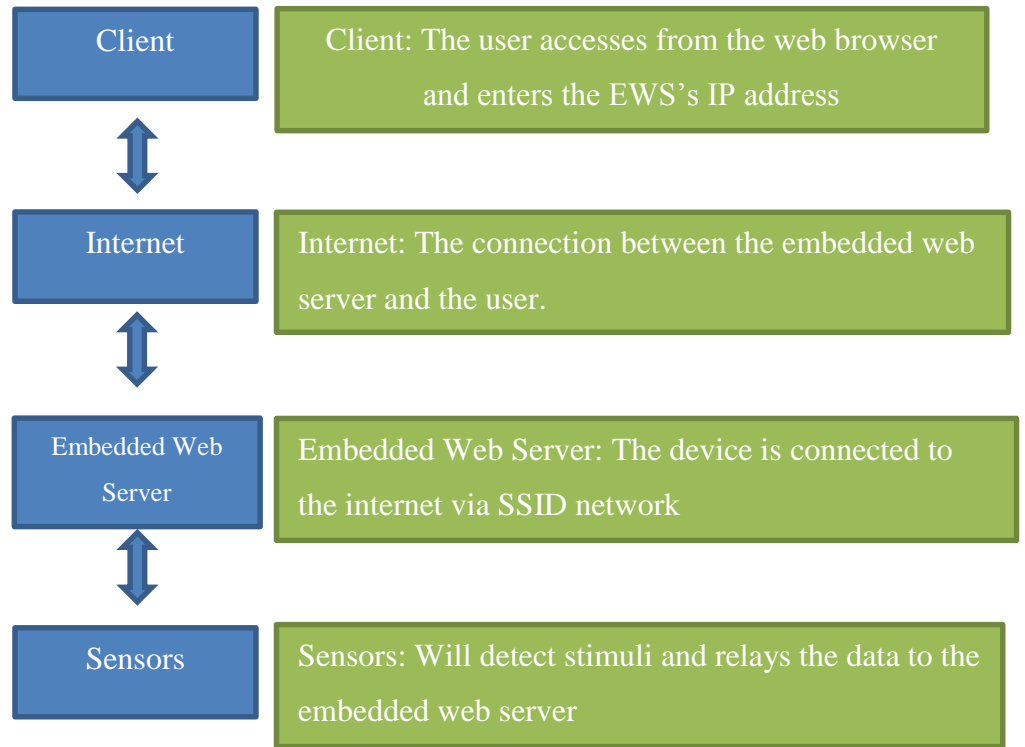
#### 2.1 Embedded Web Server

Web servers are systems that host websites and provides particular services to any requesting clients[3]. Through the modern technologies of microcontrollers, web servers are able to be implemented to these chipsets making it easier for process controlling and data wrangling via internet access. Constructing an embedded web server based on HTTP terminal, HTML pages can be embed into the server. Therefore, the terminal is able to send HTML pages to any browser to enable server to browser access anytime and anywhere[4]. Just like any typical web server, the embedded web server is capable of accepting and analyse requested by the clients and responds accordingly with results back to the client as requested[5].This shows the embedded web server is capable of providing a two-way communication between client and server. Graphic User Interface (GUI) is provided by the web browsers for a myriad browser server functions and is the default interface for many application [6]. An embedded web server is superior than the typical PC-server in terms of the low power consumption, low cost, high performance and flexible portability[7]. An embedded web server device can be implemented in our homes, businesses and even our body that can be accessed through wired or wireless connection by the user at any given time or place [8].

By default, HTTP is actually based on a simple client and server concept. The client and server are connected through a default TCP (Transmission Control Protocol) port 80. When the client requests, the IP address of the server, the server will respond by transmitting the HTML (Hyper Text Mark-up Language) documents to the client. TCP is suitable for data transfer due to its safeguard communication between client and server. The embedded web server would obtain the IP address from the connection established with the SSID Wi-Fi network. From there, the client would use said IP address to access the embedded web server. Alternatively, the IP address can be fixed permanently by creating a private



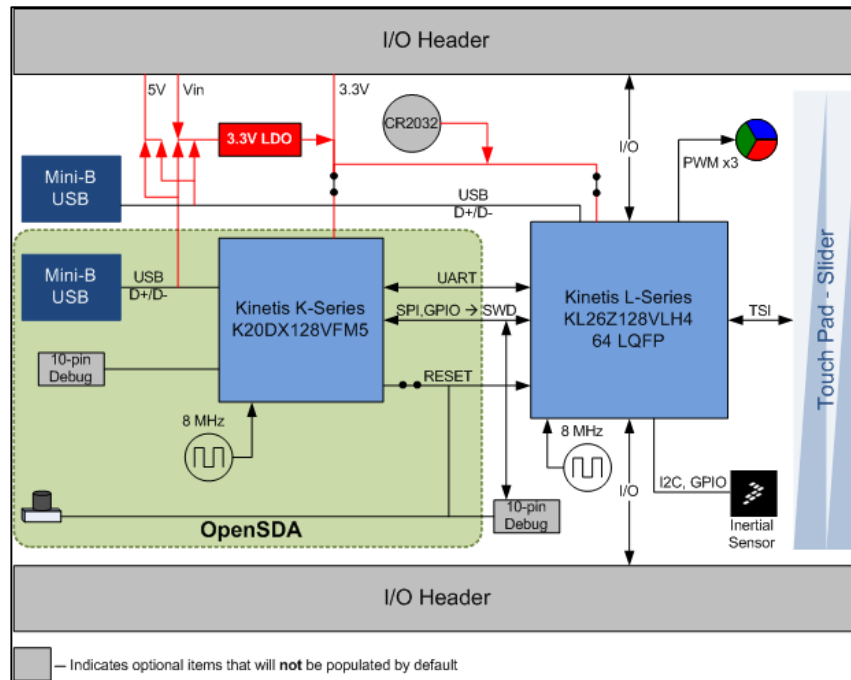
Dynamic Domain Name System (DNS) host for a domain specifically to cater the security of the embedded web servers data and connectivity.



**FIGURE 2.1: Embedded Web Server System Architecture**

## 2.2 Freescale Freedom Platform

The Freescale FRDM-KL26Z is an ultra-low-cost development platform built on ARM Cortex M0+ processor that features 128 kB flash memory and expansion board options which are compatible with a wide range of Freescale products and third-party development software and hardware. It also features low-power operated form factor as well as built-in debug interface for flash programming and run control.

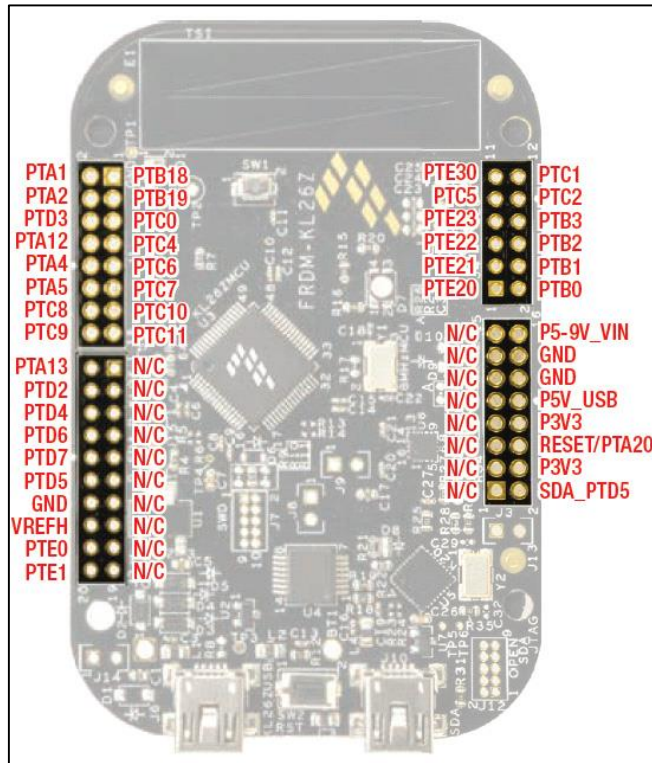


**FIGURE 2.2: FRDM-KL26Z Block Diagram**

The FRDM KL26Z has a built in OpenSDA which is an open standard serial and debug adapter that bridges the target embedded processor and USB host. The OpenSDA features as a Mass Storage Device which provides the pathway of flash programming and run-control debug interfaces rendered quick and easy.

The openness of the Freedom board enables the board to utilize third-party hardware and software. The 64 I/O pins of the Freedom board is compatible with the peripherals from Arduino shields such as Wi-Fi shield, LCD Shield and Ethernet shield. This enables for board and shield stacking with ease. The Freedom board shall be programmed using the integrated development environment provided by Freescale which is called CodeWarrior. The software has all the necessary libraries and components for developing the source codes for the Freedom board. Together with CodeWarrior, Freescale's Processor Expert Software serves as a plug-in to generate

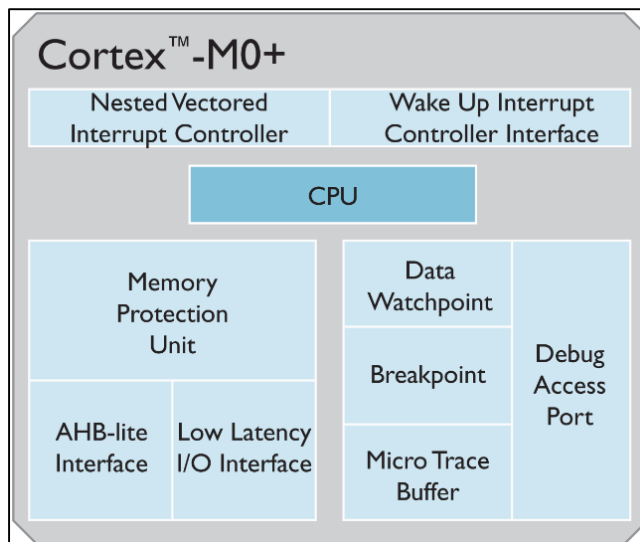
the codes written and stores libraries of the embedded components for the Freedom Board.



**FIGURE 2.3: 64 PINS OF THE FRDM-KL26Z**

### 2.3 ARM Processor

The FRDM-KL26Z utilizes the 32-bit ARM Cortex M0+ which is the most energy efficient ARM processor. Built from the ARM Cortex M0 processor, it reduces the energy consumption but increased in performance. It features the Thumb Instruction Set which provides excellent code-density for minimal system memory size and cost. It provides high level of performance and a wide code-density range of embedded applications. The ARM processor is what enables the input signals to be received and remotely control the device from a distance[9]. The ARM processor is the right choice due to its low cost and high powered which makes it capable of upload real time data and gives instructions to the sensors[10]. The ARM based embedded web server is capable of automation and controlling via online access[11].



**FIGURE 2.4: ARM CORTEX M0+ Block Diagram**

## CHAPTER 3

### METHODOLOGY

#### 3.1 Research Methodology

This project is like any other software and hardware integrated project which requires a specific approach in executing. For this project the Incremental and prototyping approach is most likely be suitable in its development. This approach emphasizes on step-by-step development by finishing one step before advancing to the other until it reaches the final stages of prototyping.

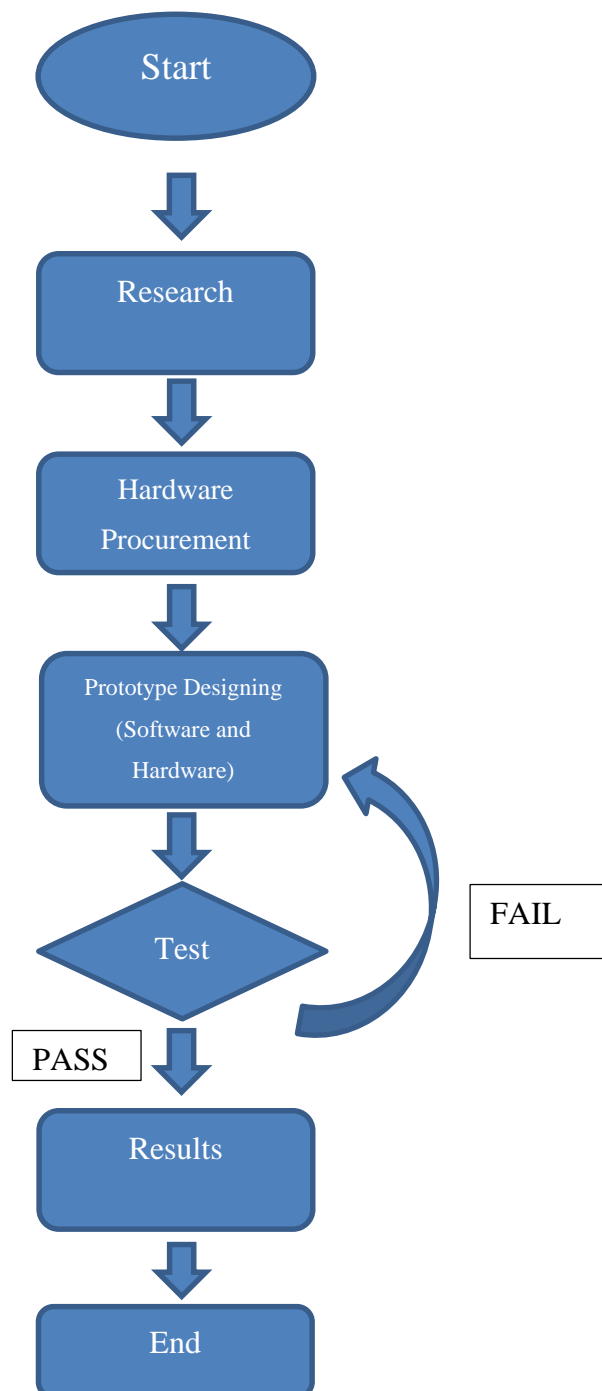
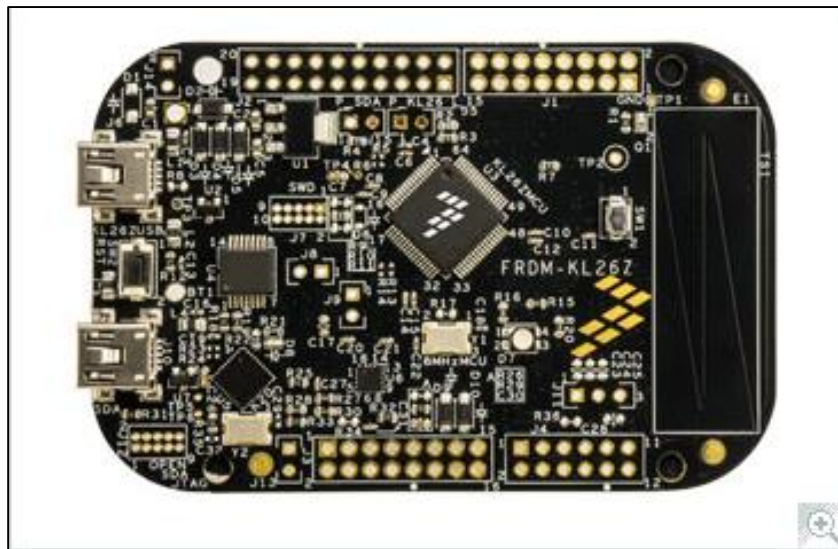


FIGURE 3.1: METHODOLOGY FLOW CHART

## 3.2 Prototype Designing

### 3.2.1 Hardware

After thorough research for the basic prototype design during research and literary reviews, several selection of hardware was made that deemed appropriate for the project. As previously stated, the microcontroller that will be used for this project is the Freedom Board KL26Z as pictured below.



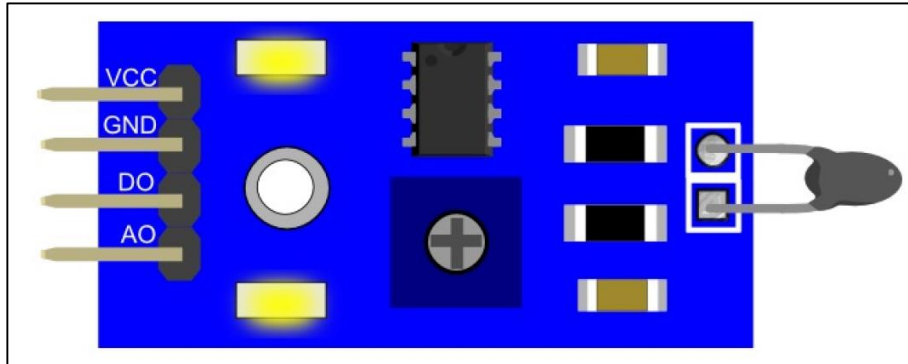
**FIGURE 3.2: FRDM-KL26Z**

This board was chosen because of its sophisticated open-sourced design, compatibility with other peripherals and the ARM processor to cater the instructions and processes as an embedded web server. The schematics of the board are shown in Appendix A. From the schematics of the board, the I/O ports of the board are compatible with those of Arduino architecture and are able to interface with many other third party modules.

The following hardware is the temperature sensor module. The Temperature Sensor Module SN-Temp-Mod is chosen because it utilizes Negative Temperature Coefficient (NTC) thermistor to detect temperature changes of the environment. It has two output, analogue and digital. NTC thermistor will change the effective resistor when there is a temperature change. The temperature is detected by measuring the voltage from a resistor network.



**FIGURE 3.3: Temperature Sensor**



**FIGURE 3.4: Temperature Sensor Module Diagram**

Pins	Notes
VCC	3.5V to 5V Operating Voltage
GND	Ground of power and signal
DO	Digital output
AO	Analogue Output

**TABLE 3.1: Temperature Sensor Module**

The hardware to establish connection for the embedded web server is the Wi-Fi shield. Particularly the WizFi210 which is a low power-consuming Wi-Fi Module that can be set to Standby mode and be woken up when the shield needed to work.

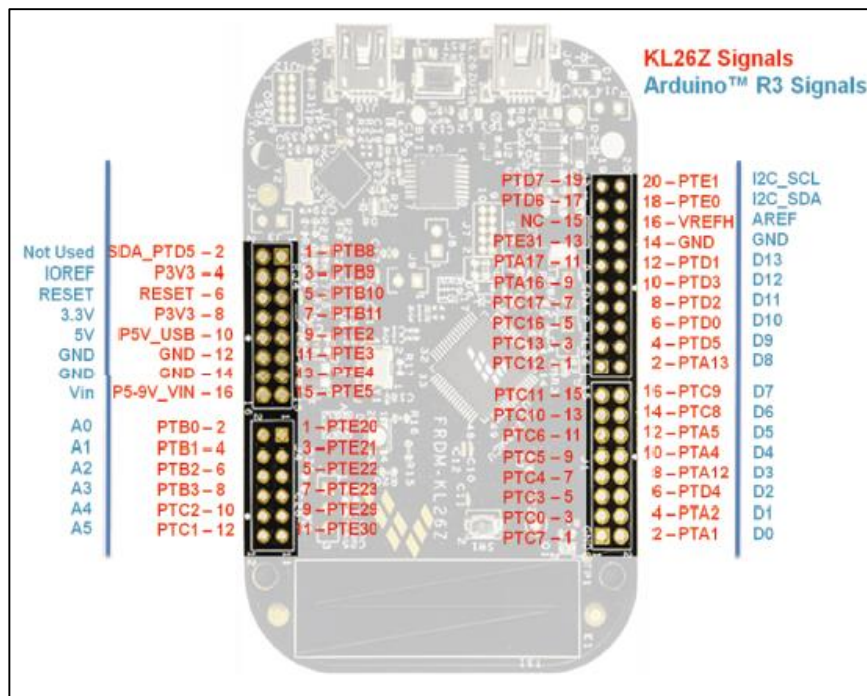
<b>Wi-Fi Chip</b>	WizFi210
<b>Radio Protocol</b>	IEEE 802.11b/g/n Compatible
<b>Supported Data Rates</b>	11, 5.5, 2, 1 Mbps (IEEE 802.11b)
<b>Modulation</b>	DSSS and CCK
<b>RF Operation Frequency</b>	2.4 - 2.497 GHz
<b>Antenna Options</b>	Chip antenna and U.FL connector for external antenna
<b>Networking Protocols</b>	UDP, TCP/IP (IPv4), DHCP, ARP, DNS, HTTP/HTTPS Client and Server(*)

<b>Power Consumption</b>	Standby = 34.0 $\mu$ A Receive = 125.0 mA Transmit = 135.0 mA
<b>RF Output Power</b>	8dBm $\pm$ 1dBm
<b>Security Protocols</b>	WEP, WPA/WPA2-PSK, Enterprise, EAP-FAST, EAP-TLS, EAP-TTLS, PEAP
<b>I/O Interface</b>	UART, SPI(*), I2C(*), WAKE, ALARM, GPIOs
<b>System Working Voltage</b>	3.3V

**TABLE 3.2: WizFi Shield Technical Specifications**

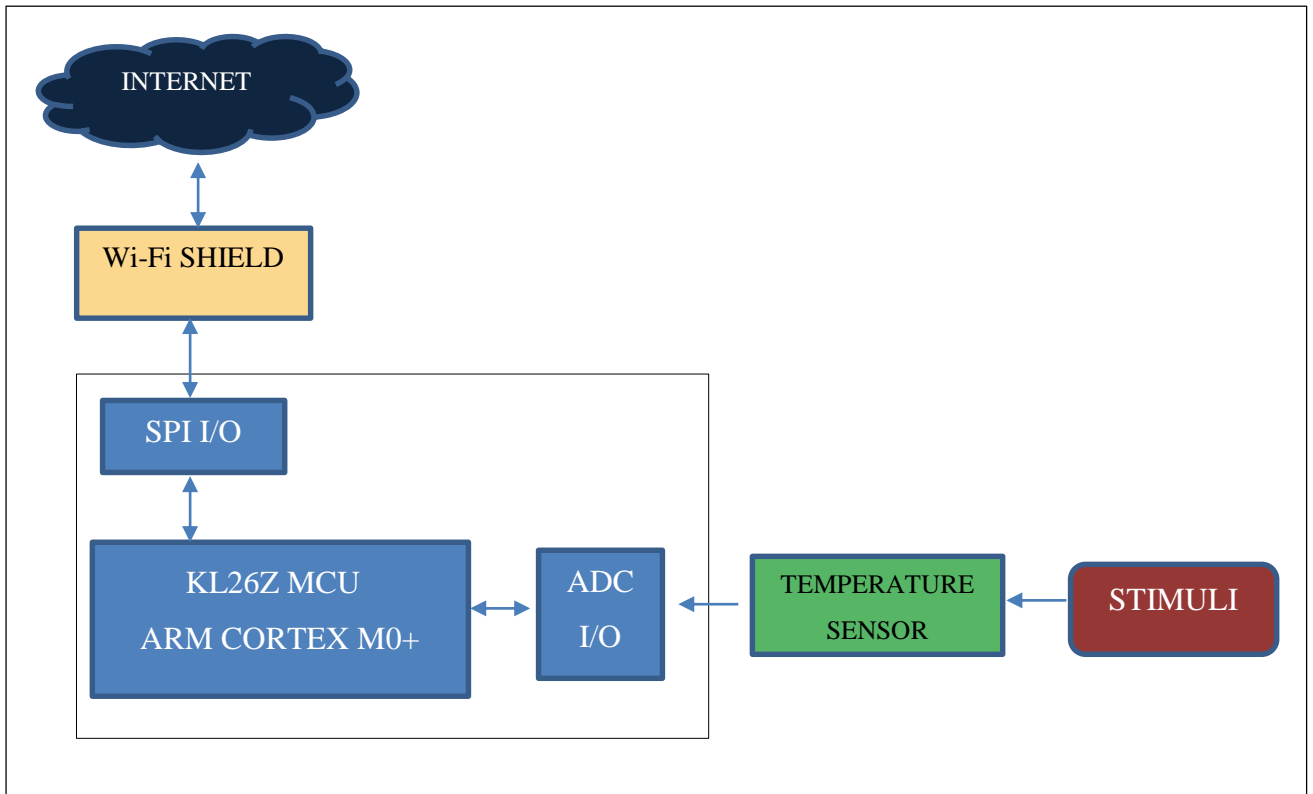
### 3.2.2 Hardware Interfacing

During this part of the project, the author relied on the schematics of the hardware in order to understand the I/O ports goes to which ports. Given the nature of the Freedom Board that is compatible with the Arduino architecture, the I/O port interface proves to be quite easy in the following figure. The Freedom platform utilizes Serial Port Interface (SPI) peripheral in order to communicate with the Wi-Fi Shield. For the temperature sensor, the Analogue Digital Converter (ADC) peripheral of the Freedom Platform is used.



**FIGURE 3.5: Freedom Board I/O with Arduino Pinout Reference**





**FIGURE 3.6: Block Diagram of Embedded Web Server**

Figure 3.6 illustrates the connections that are involved in the embedded web server. The Freedom platform is the centre of it all. The temperature sensor is interfaced through the ADC peripheral. From the values obtained, the KL26Z will relay the data to the Wi-Fi shield connected via SPI peripheral. The Wi-Fi shield establishes connection to the internet and displays the obtained data through the User Interface via web browser. Every data value obtained has to pass through the KL26Z before it is transmitted to the user via Wi-Fi shield and the internet.

### ***3.2.3 Firmware Designing***

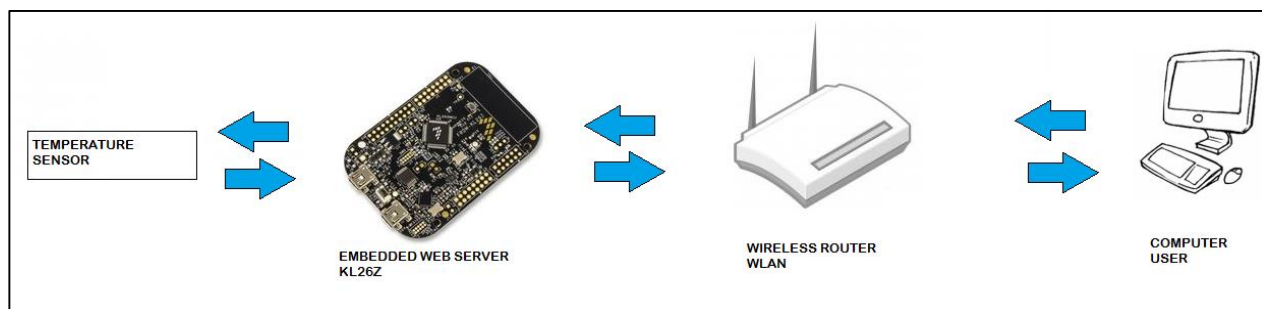
Designing the firmware for the project is as not easy as the hardware assembly. Various sources had to be referred to in order to construct the program's coding. Understanding how the coding works as well as working out the bugs that have occurred during compiling the program. Some parts of the code are acquired through examples found on the internet and had to be modified in order to fit with the programming and hardware. Refer to Appendix D and E for the codes acquired from Arduino library and the modified codes respectively. Both codes function the same way but for different hardware. Appendix D works for Arduino WiFi Shield whereas Appendix E works for WizFi Shield. The author would have to use both codes in order to assimilate with the Freedom Board.

### 3.3 Project Work

During the course of the project, the author had thoroughly researched the internet regarding the creation of embedded web servers and the Freedom platform. Despite using the FRDM-KL26Z, resources that were found were mostly regarding an earlier model of the Freedom platform, the FRDM-KL25Z. However, both Freedom platforms are quite similar and the author had used said resources as reference nonetheless.

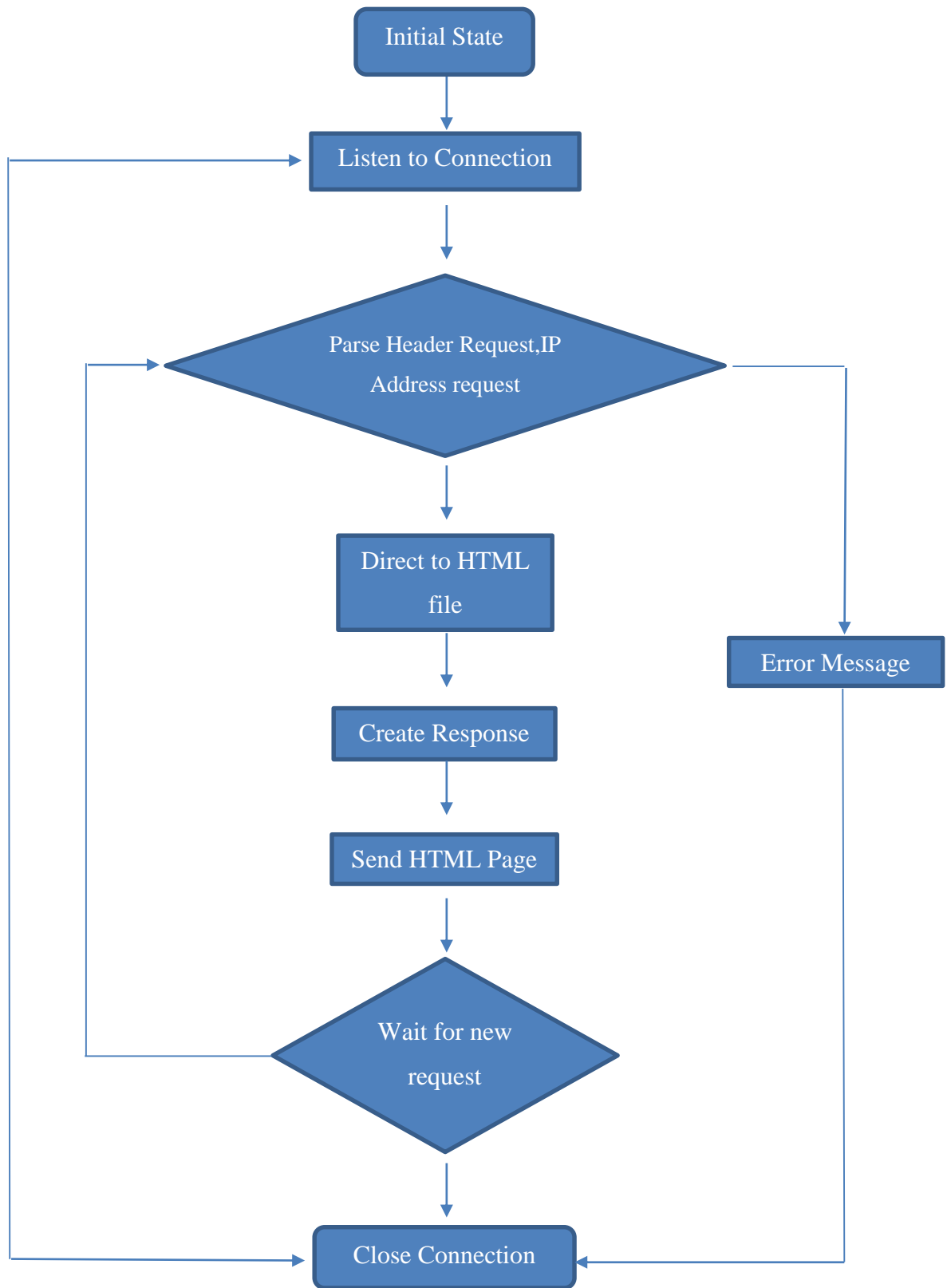
#### 3.3.1 Project Design

The user would connect to the WLAN and enter the IP address of the embedded web server. The web browser then would display the user interface (web page) served by the embedded web server. From the figure below, the user would not necessarily be connected from only a computer. A web browser from a smartphone or tablet would also be able to access the embedded web server.



**FIGURE 3.7: Hardware Connections**

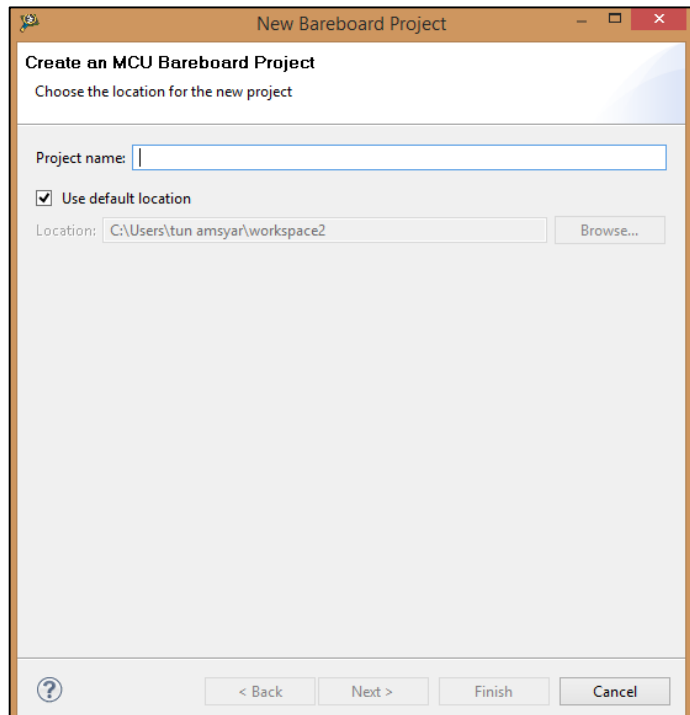
Generally, an embedded web server is a finite state machine that administers HTML requests in a step by step sequence [12]. Below is the embedded web server process structure. It is from this process structure, the author would be able to write the codes enabling the functionality of the embedded web server.



**FIGURE 3.8: Embedded Web Server Process Structure**

### 3.3.2 *Integrated Development Environment (IDE)*

For the development of the project, the author used CodeWarrior Development Studio version 10.5. This particular IDE was chosen due to its cohesiveness with the Freedom Board. It has the required specification to cater the Freedom Board architecture application development. To create a new project, go to File > New> Bareboard Project and it will pop out a window.

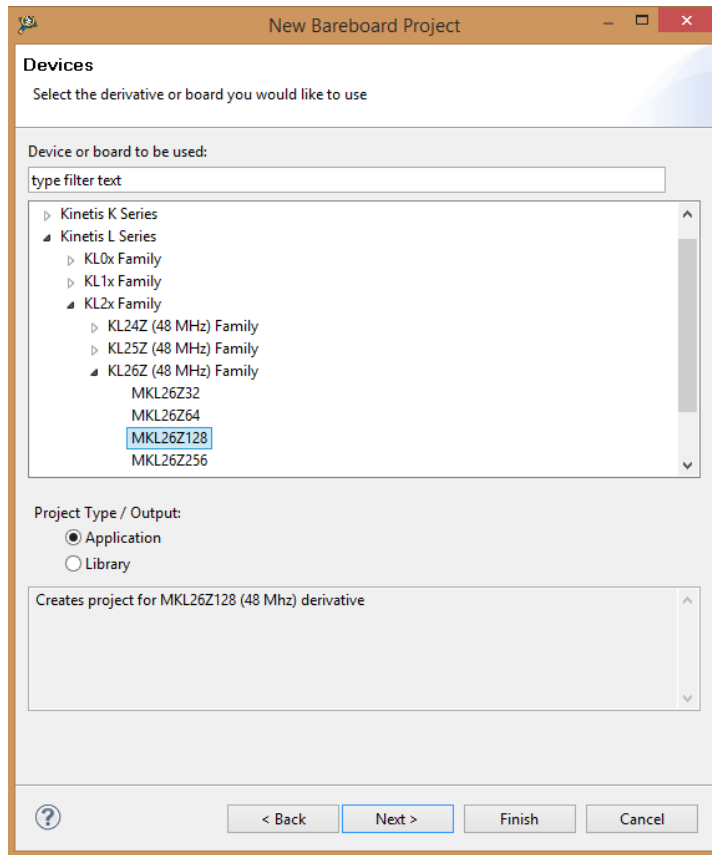


**FIGURE 3.9: Setting Up a New Project**

After which the author has to select the target processor in which case is the Kinetis Microcontroller Unit, MKL26Z128. If the wrong processor is chosen the IDE will prepare the wrong board and components. The IDE will also load the wrong source files and header files incompatible with the board.

In addition to CodeWarrior, the author also used Arduino IDE 1.5 to refer Arduino source codes in reference to the KL26Z programming. The author used the source codes to understand how Arduino based web servers' work and to apply the information gained in the programming for the KL26Z. Since both Arduino and Freedom platforms use C language, there would not be any problem translating the

languages. The problem was to understand and implement the communication between the microcontrollers and the peripherals.

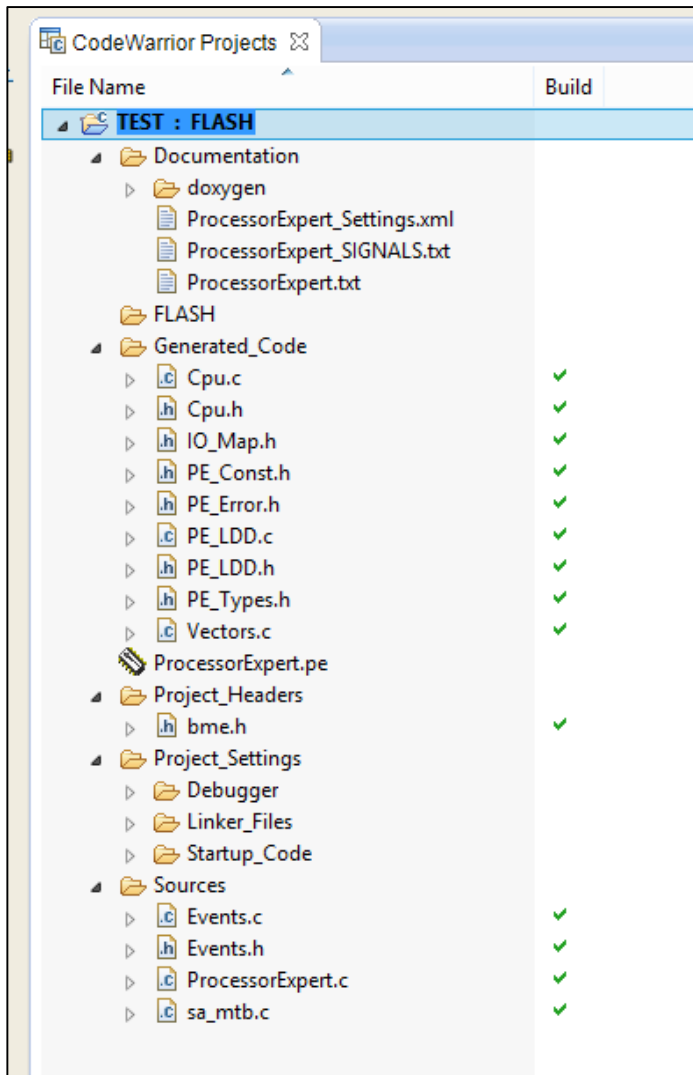


**FIGURE 3.10: Selecting KL26Z from Menu Selection**

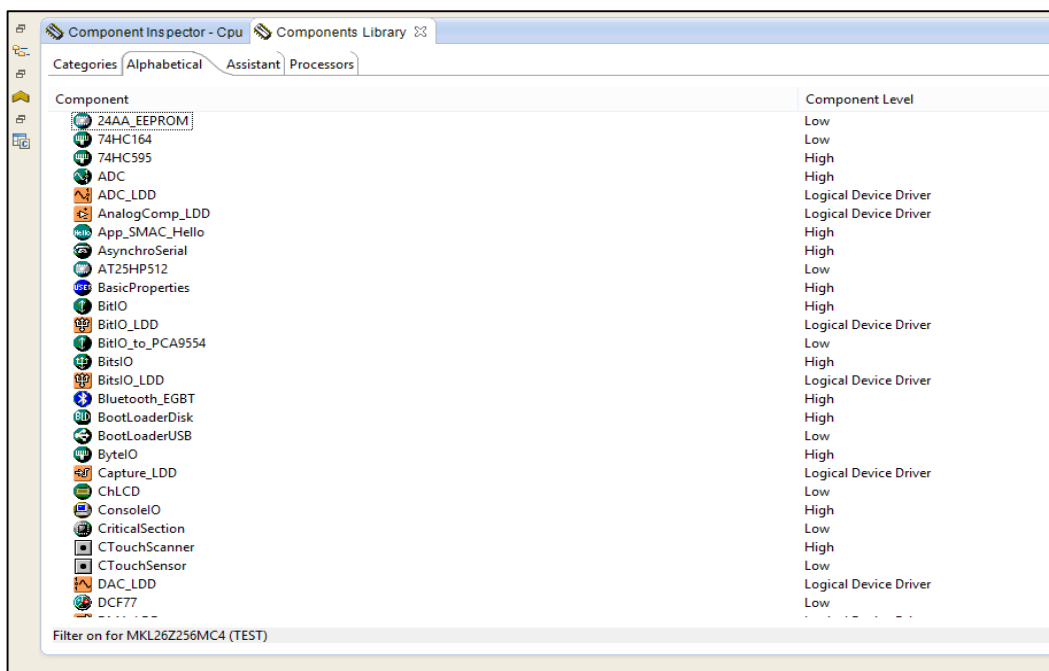
### ***3.3.3 Processor Expert CodeWarrior***

The Processor Expert (PEX) is a plug-in for CodeWarrior designed for rapid application development of embedded applications. The PEX plug-in generates the codes from the embedded components while the CodeWarrior IDE manages the project files, compilation and debug processes. The PEX has internal definition of the microcontroller along with all of the integrated peripherals making it much easier to understand and writing the program for the project.

The PEX works with user choosing the desired peripheral from the list of components provided along with the required settings of the peripheral. After which, the PEX would automatically generate the initialization codes for the peripheral. From there, the user is able utilize the peripheral codes for its desired function.



**FIGURE 3.11: Header Files and Start-up Codes Generated by Processor**



**FIGURE 3.12: Components Library of Processor**

### 3.3.4 Wi-Fi Shield

This particular shield utilizes WIZnet's Wi-Fi module, WizFi210 for the implementation of wireless communication in Arduino development environments. Note that even the shield is built for Arduino environments, the Freedom Board is made compatible with Arduino peripherals. Therefore, there should not be any compatibility issues between the peripheral and the Freedom Board. Although, there are certain library files for the Wi-Fi shield had to be searched on the internet in order to program the Wi-Fi shield. Some files had to be taken from the Arduino IDE.

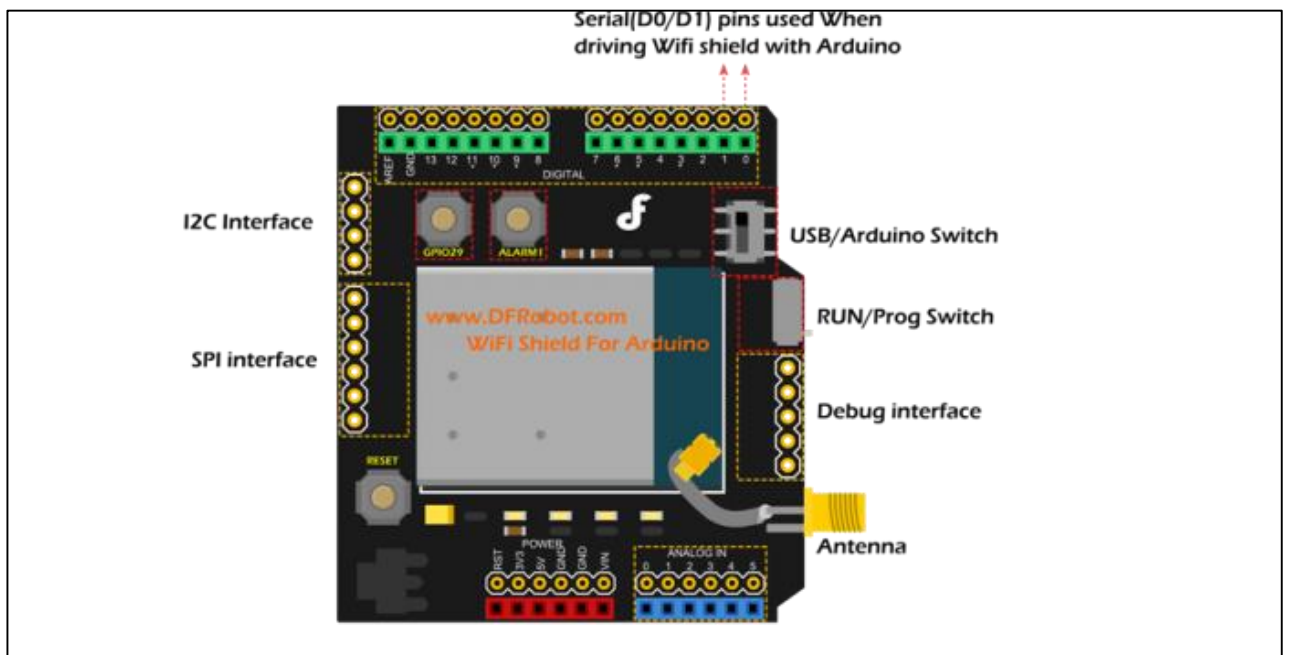


FIGURE 3.13: Diagram of WizFi

### 3.3.5 KL26Z Programming

For the programming of the KL26Z, the author had used C programming language for this project. There are two ways of programming the KL26Z. One is utilizing the Processor Expert that would generate codes for the on board pins of the KL26Z board according to the desired pin peripheral. The other is manually coding the pins in accordance to the desired pin peripheral. The author had used both in order to learn and to generate the program codes for the project.

There are several important initialization and declaration that are included in the program:

- **MKL26Z4.h** – KL26Z Peripheral Memory Map Implementation Header File
- **\_\_arm\_start.c** – Entry point for ARM programs Source File
- **\_\_arm\_end.c** – Interface for board-level termination Source File
- **kinetis\_sysinit.c** – Default initialization routines for Kinetis ARM systems Source File
- **kinetis\_sysinit.h** – Default initialization routines for Kinetis ARM systems Header File
- **main.c** – Main project program
- **makefile** – Compile project

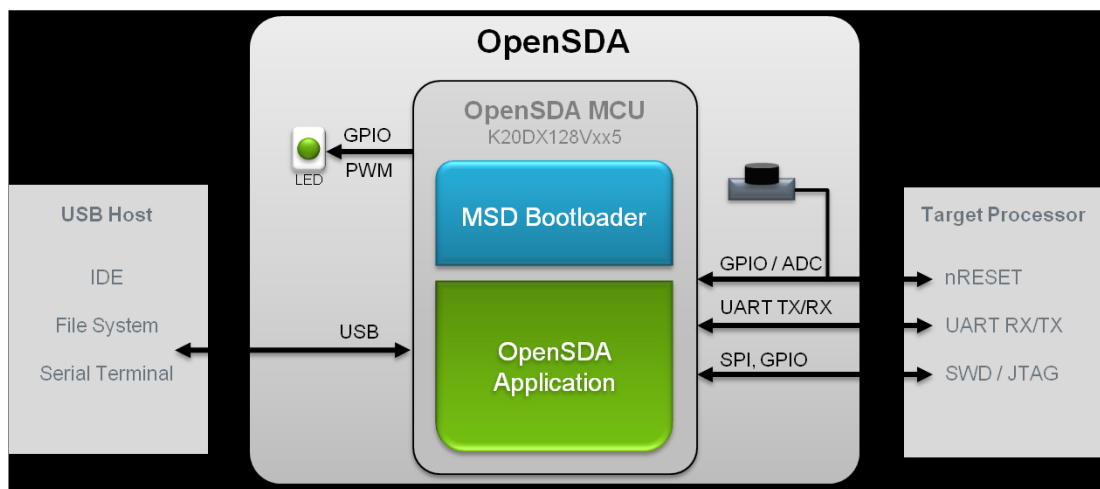
Before setting the peripherals of the KL26Z, the system clock has to be initialized. For this project, the author decided to use the maximum frequency clock speed which is 48MHz. The KL26Z uses clock gating for each peripheral. Therefore for each peripheral used, the PORT has to be gated on in order to use. This will be further discussed below on SPI initialization.

For the Wi-Fi shield peripheral, the author had to use Wi-Fi libraries from Arduino IDE as reference in order to generate the codes and functions for wireless connection. All the necessary header files would be included in main program. The functions used are already included in the Wi-Fi libraries. So in the main program the author only had to call said functions and modify the required parameters for the function. Changes to the functions used were to be done in the Wi-Fi libraries' source files. The most heavily changed codes were regarding the interface between the Wi-Fi shield and the Freedom board.

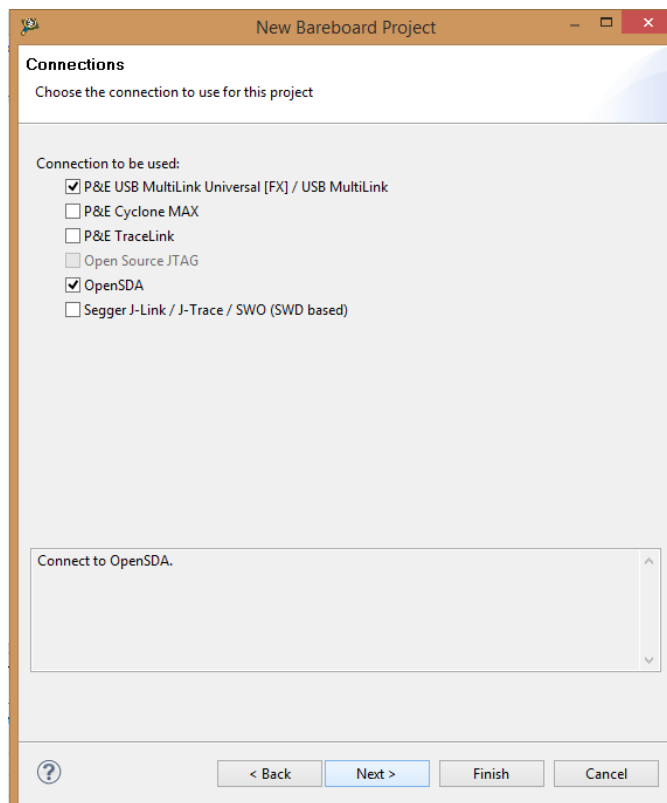


### 3.3.6 KL26Z Debugging

The Freedom Board KL26Z has a built-in OpenSDA. OpenSDA is an open-standard serial and debug adapter that acts as a bridge for serial and debug communications between a USB host and an embedded target processor as shown in Figure. The author used the P&E Debug Application as the OpenSDA Application that provides debugging and a virtual serial port in one application. The P&E Debug Application is designed to debug the resident target processor in the OpenSDA system with limited support for off-board devices within the same processor family as the resident target processor.



**FIGURE 3.14: Block Diagram of OpenSDA**



**FIGURE 3.15: Selecting OpenSDA Connection**

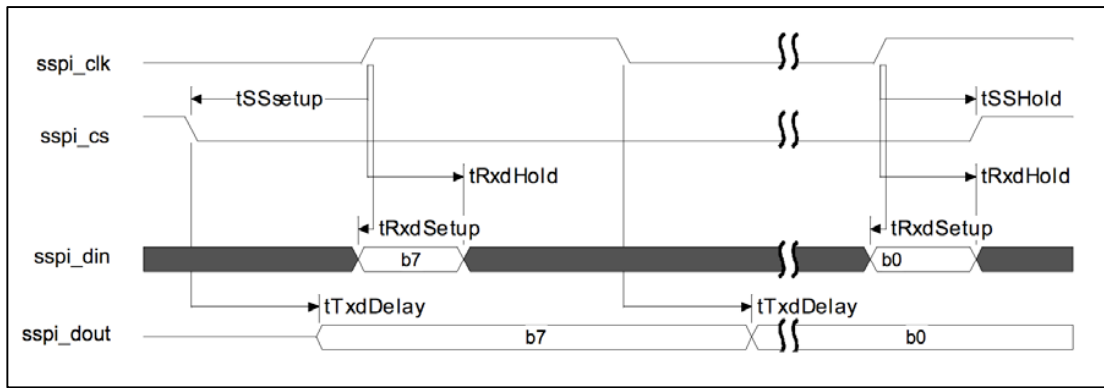
During the debugging and compiling process, several errors had occurred that rendered compiling the program unsuccessful. Most common error that occurred was no definition of the used function. This is a result of the function was not properly linked in the header files. At the same time of errors, there were also warnings that appear mostly related to redefinition of variables that were already declared in the header files. Despite the program has warning flags, the programs was still able to be compiled and execute.

### ***3.3.7 Serial Peripheral Interface (SPI)***

Each I/O pins of the KL26Z have alternative modes that are to be set by the developer depending on its usages. Refer to **APPENDIX E** for pin out modes. For this project, SPI is the method of interfacing of the KL26Z with the Wi-Fi Shield. Most Arduino shields utilize SPI protocol in communication with microcontrollers. Arduino IDE supplies the user with **SPI.h** for initializing I/O pins of the Arduino boards. However, it is the opposite for the Freedom Boards.

The KL26Z board has to be manually initialized by the user according to the pins with SPI capabilities. A total of four pins had to be used of the KL26Z to initiate SPI. One pin for Master Output Slave Input (MOSI), one pin for Master Input Slave Output (MISO), one pin for Slave Select or Chip Select (SS) and one pin for Serial Clock (SCK). Per the norm, the microcontroller device (KL26Z) is usually set as the Master and the secondary device (Wi-Fi Shield) as the Slave.

Utilizing the Wi-Fi Shield, an additional General Purpose I/O (GPIO) is needed to indicate the Wi-Fi Shield is awake and is sending data to the Master device. The Wi-Fi Shield also requires the SPI to run only at Motorola mode which means Clock Polarity (CPOL) = 0 and Clock Phase (CPHA) = 0. This mode allows data to be captured on the falling edge of the clock and propagates the data during the rising edge of the clock.



**FIGURE 3.16: SPI Mode 0 Timing Diagram**

### 3.3.8 Code Synthesis

#### Peripheral Initialization

This is to illustrate how to initialize a peripheral of the Freedom Platform. In this case is the SPI peripherals. As previously stated, the SPI of the KL26Z has to be manually initiated by the developer in order the pins to act accordingly. The following are the steps to initialize the SPI pins:

1. Enable the clock to the ports involved
  - a. The SPI pins are located in PORT D and therefore the clock is to be enable to the entire PORT D.
  - b. The SPI clock must also be enabled
  - c. The clock is controlled by the System Clock Gating Registers of the KL26Z.
  - d. There are specific bits on the registers must be enabled (Logic 1) that controls specific ports. For example, PORT D is located at System Clock Gating Register 5(SIM\_SCGC5) bit number 12. By default, the bit is set to 0 and therefore must be enabled to 1 in order to enable clock at PORT D only. SIM\_SCGC5 only caters for PORT pins.
  - e. Clock for the SPI is also controlled by the System Clock Gating Register. But it is on System Clock Gating Register 4(SIM\_SCGC4) bit 23 and must also be enabled to 1. SIM\_SCGC4 only caters for interfacing modes.
  - f. The specified bits in the registers must be the only bit to be enabled using the specific register mask e.g.: SIM\_SCGC5\_PORTD\_MASK provided by the KL26Z header file, **MKL26Z4.h**.

Code:

```
SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; //Enable Clock on
PORTD
SIM_SCGC4 |= SIM_SCGC4_SPI0_MASK; //Enable Clock on SPI0
```

2. Enable the pins to be in the SPI mode
  - a. The mode of the pins are controlled by the PORT Control Registers (PORT<sub>x</sub>\_PCRN)
  - b. In this case, the pins used are in PORT D, PORT Control Registers 0,1,2 and 3 e.g.: PORTD\_PCR0.
  - c. To configure the pins, the Pin Mux Control must be set according to the specific pin's available options.
  - d. In this case for example, to activate SPI mode in PORT D Pin 0, the Pin Mux Control must be set to 2 e.g.: PORT\_PCR\_MUX(2). For PORT D Pin 0, it is specified as the SPI0 Chip Select pin.

Code:

```

PORTD_PCR0 = PORT_PCR_MUX(2);           //PTD0 TO MUX 2
[SPI0_PCS0]

PORTD_PCR1 = PORT_PCR_MUX(2);           //PTD1 TO MUX 2
[SPI0_SCK]

PORTD_PCR2 = PORT_PCR_MUX(2);           //PTD2 TO MUX 2
[SPI0_MOSI]

PORTD_PCR3 = PORT_PCR_MUX(2);           //PTD3 TO MUX 2
[SPI0_MISO]

```

3. Setting KL26Z as the SPI Master and other options of SPI
  - a. SPI mode is controlled by SPI Control Register 1 (SPI<sub>x</sub>\_C1)
  - b. To configure as SPI Master, the SPI<sub>x</sub>\_C1 bit 4 has to be set as 1.
  - c. To set the Clock Polarity and Clock Phase to 0 for Motorola mode 0 respectively, the bit 3 has to be 0 and bit 2 as 1 respectively in SPI<sub>x</sub>\_C1.
  - d. However, by default the logic bit setting of the bits 3 and 2 equals to 0 and 1 respectively. Therefore, no need to configure the register.
  - e. To configure SPI<sub>x</sub>\_C1 to master we must use the Master Mask (SPI\_C1\_MSTR\_MASK)

- f. SPI Control Register 2 (SPI<sub>x</sub>\_C2) controls the optional functions of the SPI
- g. Configuring SPI Master mode-fault function enable is to enable the master SS pin as the slave select output.
- h. The Master mode-fault function is located in SPI Control Register 2 (SPI<sub>x</sub>\_C2) bit 4 and must be set to logic 1.
- i. To configure SPI<sub>x</sub>\_C2 to mode-fault function, the MODFEN MASK IS USED (SPI\_C2\_MODFEN\_MASK)
- j. The Baud rate is configured in the SPI Baud Rate Register (SPI<sub>x</sub>\_BR)
- k. The SPI Baud rate prescale divisor (SPPR) is set to 3 (SPI\_BR\_SPPR(0x02)) and SPI baud rate divisor (SPR) is set to 256 (SPI\_BR\_SPR(0x08)).

Code:

```

SPI0_C1 = SPI_C1_MSTR_MASK ;           //SET SPI0 TO
MASTER & SS PIN TO AUTO SS

SPI0_C2 = SPI_C2_MODFEN_MASK;         //MASTER SS PIN
ACTS AS SLAVE SELECT OUTPUT

SPI0_BR = (SPI_BR_SPPR(0x02) | SPI_BR_SPR(0x08)); // SET
BAUD RATE

SPI0_C1 |= SPI_C1_SPE_MASK;           // ENABLE SPI0

```

#### 4. SPI Initializing function

Code:

```

void spi_init(void)
{
    SIM_SCGC4 |= SIM_SCGC4_SPI0_MASK;    //ENABLE SPI0
    CLOCK
    SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK;    //ENABLE CLOCK
    ON PORTD

    PORTD_PCR0 = PORT_PCR_MUX(2);         //PTD0 TO MUX 2
    [SPI0_PCS0]
}

```

```

    PORTD_PCR1 = PORT_PCR_MUX(2);           //PTD1 TO MUX 2
[SPIO_SCK]
    PORTD_PCR2 = PORT_PCR_MUX(2);           //PTD2 TO MUX 2 [SPIO_MOSI]
    PORTD_PCR3 = PORT_PCR_MUX(2);           //PTD3 TO MUX 2 [SPIO_MISO]

    SPI0_C1 = SPI_C1_MSTR_MASK ;             //SET SPIO TO MASTER & SS
PIN TO AUTO SS

    SPI0_C2 = SPI_C2_MODFEN_MASK;           //MASTER SS PIN ACTS AS
SLAVE SELECT OUTPUT

    SPI0_BR = (SPI_BR_SPPR(0x02) | SPI_BR_SPR(0x08)); // SET BAUD
RATE

    SPI0_C1 |= SPI_C1_SPE_MASK;             // ENABLE SPIO
}
int main(void)
{
    spi_init();
}

```

## 5. SPI Write and Read function

Code:

```

uint8_t SPI_status(void) {
    return SPI0_S;
}

// Write out all characters in supplied buffer to register at
address
void SPI_write(uint8_t* p, int size, uint8_t addr) {
    int i;

    for (i = 0; i < size; ++i) {
        // poll until empty
        while ((SPI_status() & 0x20) != 0x20);
        SPI0->D = p[i];
    }
}

// Read size number of characters into buffer p from register at
address
void SPI_read(uint8_t* p, int size, uint8_t addr) {
    int i;

    for (i = 0; i < size; ++i) {
        // poll until full
        SPI0->D = 0x00;
        while ((SPI_status() & 0x80) != 0x80);
        p[i] = SPI0->D;
    }
}

```

### 3.3.9 Web Server Page Design

The webpage was designed in the simplest way as possible. The webpage was created using the HyperText Markup Language (HTML) and was done in text editor Notepad.exe. The data is displayed in a tabular form to make it easier for viewing. Prior to the project, the author has had experience with HTML by self-experimenting with the language.

```
<!doctype html>
<html>
  <head><title> Freedom Embedded Web Server</title></head>
  <body>
    <div>
      <center><h1>Freedom Web Server</h1></center>
      <center>
<style>table,th,td
{
border: solid black;
}
</style>
<table>
  <thead>
    <tr>
      <th>Parameter</th>
      <th>Status</th>
    </tr>
  </thead>

  <tbody>
    <tr>
      <td>Temperature</td>
      <td>(sensor_reading)</td>
```

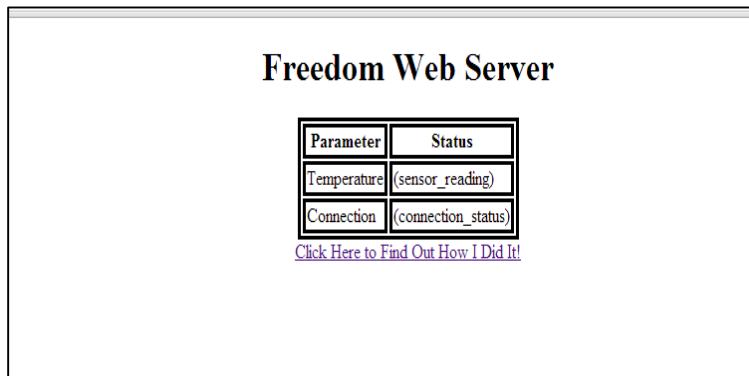


```

</tr>
<tr>
  <td>Connection</td>
  <td>(connection_status)</td>
</tr>
</tbody>
</table>
<center><a href="http://mcuoneclipse.com/2014/01/25/frdm-with-
arduino-ethernet-shield-r3-part-3-embedded-web-server/">Click Here
to Find Out How I Did It!</a></center>
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
<br><br><br><br><center><h6><b>Created by: Tun
Amsyar</b></h6></center>
</div>
</body>
</html>

```

The web server should serve a simple website like this:



**FIGURE 3.17: Simple Web Server Page**

The (sensor\_reading) and (connection\_status) is meant for the output from the collected data to be displayed in the respective spaces. The webpage is to refresh automatically to provide real time data display.

### **3.4 Key Milestones**

To be able to achieve the project's objectives, several key milestones had been underlined in order to meet the desired requirements. Below are the key milestones set:

#### ***3.4.1 Background Study***

Using various sources such as online journals and articles to understand more about the problem statements as well as design ideas to overcome said problem statements. The concept of embedded web server is able to be fathomed through in-depth study and analysis. The author had also spent time reviewing C and HTML languages in order to create essential parts of the project. Some online references were also used such as Freescale Community forum in order to understand more about the Freedom platform.

#### ***3.4.2 Project Design***

Throughout FYP 1, the author had outlined the necessary procedures, hardware, software and tools required for designing the project. The author initially experimented with the Freedom board in order to comprehend the usage of CodeWarrior and the functionality of the Freedom board. From the basic 'Hello World' program, to the complexity of interfacing with peripherals. The designing of the prototype program was purely by trial and error. Of course some heavy modifications of the original guidelines had to be done in order to fix several problems that had occurred.

Due to time constraints, the project had to be amended to suit the given time. Rather than being able to connect to the embedded web server from anywhere in the world, the project's programming had to be amended to be able to access the embedded web server if the user was in the same wireless local area network. Initially the project was to utilize a Dynamic Domain Name System (DDNS) service which would ultimately provide security and enable connectivity with the user from anywhere in the world.

### ***3.4.3 Project Implementation***

Freescale Codewarrior IDE would be used to create the codes and load the program to the KL26Z. The loaded code would configure the KL26Z to be the embedded web server. The connection of the KL26Z to the internet is manually configured by the user. The user manually configures the IP address, Wi-Fi network name and password of the embedded web server in the source code.

To confirm the connection of the embedded web server and the user, the IP address of the embedded web server would be pinged by means of the command prompt. Once the connection established and confirmed, the user would use the web browser to enter the IP address and requests the server connection of the KL26Z. Through the web browser, the user interface, in the form of HTTP page, would display the data acquired by the sensors attached to the KL26Z.

### ***3.4.4 Documentation and Report***

Each progress of the project is documented. Every complications encountered is recorded for future reference. Findings and development are discussed and analysed thoroughly. Any lessons learned shall be recorded for future improvement. Relevant sources such as website links, video tutorials and user manuals shall be included in a CD-ROM for future references as well as the project work, source codes and header files.

### 3.5 Gantt Chart

No	Activities	Week																											
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	Title Selection	█	█																										
2	Literary Research		█	█	█	█	█																						
3	Prototype Designing					█	█	█	█																				
4	Hardware Procurement									█	█																		
5	Prototype Building										█	█	█																
6	Firmware Designing											█	█	█	█	█	█	█	█	█	█	█	█	█					
7	Firmware Testing And Troubleshooting																█	█	█	█	█	█	█	█	█	█			
8	Prototype and Firmware Completion																						█	█	█	█	█		
9	Report and Thesis																█	█	█	█	█	█	█	█	█	█	█	█	█

TABLE 3.3: Gantt Chart

### 3.6 Hardware and Tools

<b>HARDWARE</b>	<b>DESCRIPTION</b>
Laptop	<i>For every programming and compiling purposes of the project</i>
FRDM-KL26Z Board	<i>The platform for the embedded web server</i>
Mini USB Cable	<i>The wired link between the Freedom Board and laptop</i>
Temperature Sensor	<i>The sensor will be used to collect temperature reading as the data parameters</i>
LED	<i>To indicate the sensor is running and others</i>
WiFi Shield	<i>The peripheral to enable internet connectivity for the Freedom board</i>

**TABLE 3.4: Hardware Description**

<b>SOFTWARE</b>	<b>DESCRIPTION</b>
CodeWarrior Development Tools	<ul style="list-style-type: none"> <li>• <i>The Integrated Development Environment for the Freedom board.</i></li> <li>• <i>The debugging and coding of the program is made using this software by Freescale</i></li> <li>• <i>Compatible with the FRDM-KL26Z</i></li> <li>• <i>Made free for evaluation by Freescale</i></li> </ul>
Processor Expert Software	<ul style="list-style-type: none"> <li>• <i>Provides the embedded components library to CodeWarrior</i></li> <li>• <i>External libraries may be installed as well</i></li> <li>• <i>Generates the code for the Freedom Board</i></li> </ul>

**TABLE 3.5: Software Description**

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Wi-Fi Shield Library

The Wi-Fi shield relies heavily on the library files of the Arduino IDE. This meant several modifications of the related Arduino header files had to be done to enable the Wi-Fi shield to work with the KL26Z. Modifications were mostly done to the driver source files of the Wi-Fi shield. The following are the source code files used by the Wi-Fi shield that were modified:

- **server\_drv.cpp** – Server Driver Source File
- **server\_drv.h** – Server Driver Header File
- **spi\_drv.cpp** - SPI Driver Source File
- **spi\_drv.h** – SPI Driver Header File
- **wifi\_drv.cpp** – Wi-Fi Driver Source File
- **wifi\_drv.h** – Wi-Fi Driver Header File
- **WiFi.cpp** – Wi-Fi Functions Source File
- **WiFi.h** – Wi-Fi Functions Header File
- **WiFiClient.cpp** – Wi-Fi Client Functions Source File
- **WiFiClient.h** – Wi-Fi Client Functions Header File
- **WiFiServer.cpp** – Wi-Fi Server Functions Source File
- **WiFiServer.h** – Wi-Fi Server Functions Header File

The author had use references from the internet particularly microcontroller development forums in order to apply the changes to the files. The aforementioned Wi-Fi shield library contains the functions and declarations of variables for the Wi-Fi shield to carry out the necessary steps for wireless network connectivity.

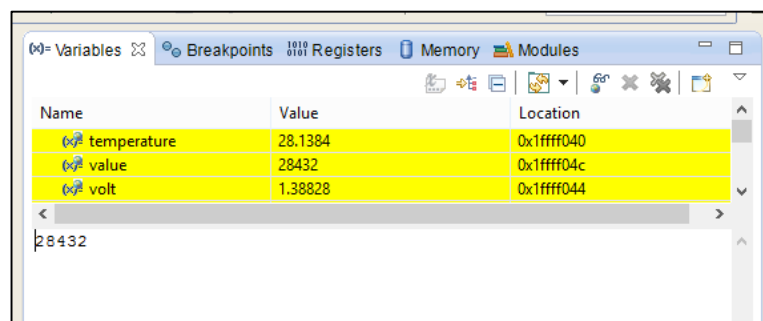
## 4.2 Coocox Peripheral Library

The Coocox Peripheral Library or CoX is a group of interface function definition. CoX defines the functional access functions of common MCU peripherals such as UART, SPI and I2C. The author utilized the CoX library due to its comprehensive functions that is compatible with most ARM chip processors especially the Cortex M0. It also provides special APIs for specific MCUs' features. The CoX library officially caters for the KL25Z MCU. However, since the KL26Z is very similar to the KL25Z, it is possible that the CoX library is compatible with the KL26Z.

The author had used the CoX library as a guideline to write the code for the KL26Z and the Wi-Fi shield. The CoX also has several source codes function of the Arduino Wi-Fi shield which was later used and referenced for interfacing between the Wi-Fi shield and the Freedom platform. Using both CoX library and Arduino IDE examples to create the codes for the embedded web server. The CoX would later be included in the project documentations as references for future use.

## 4.3 Temperature Sensor Module

As previously stated, the temperature sensor used is a thermistor whereby the resistivity changes with temperature and the voltage value is used to be converted into temperature by means of a formula. The temperature sensor is interfaced with the Freedom platform through the Analogue-Digital-Converter (ADC) peripheral. The ADC functions to acquire the raw voltage value from the temperature sensor and is converted through several equations in order to acquire the real voltage of the temperature sensor detected.



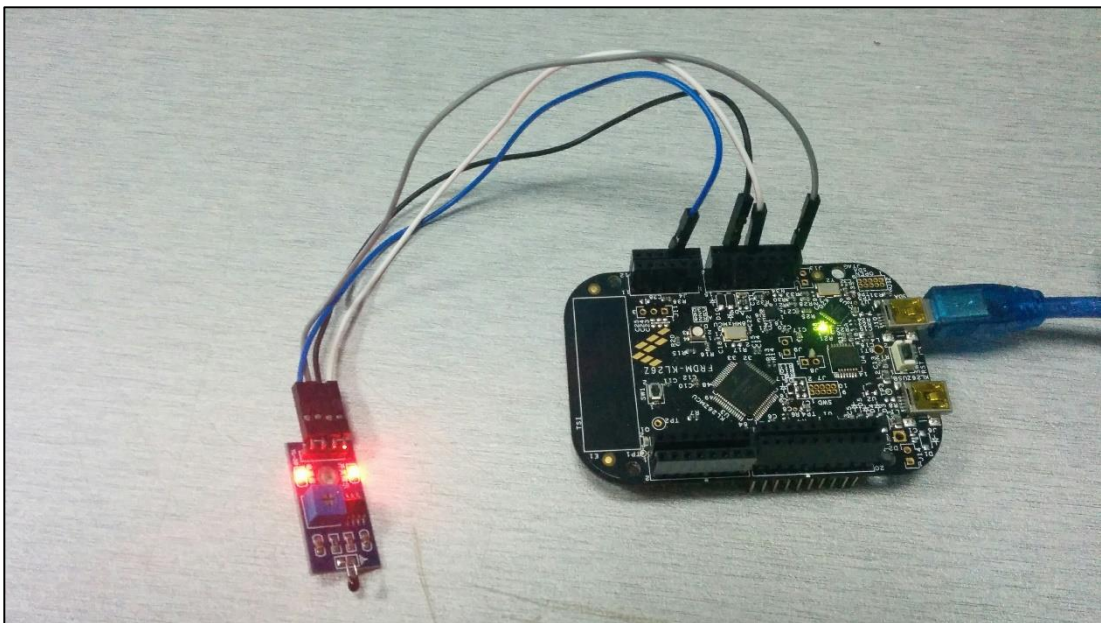
Name	Value	Location
temperature	28.1384	0x1ffff040
value	28432	0x1ffff04c
volt	1.38828	0x1ffff044

**FIGURE 4.1: Output Values from Temperature Sensor**

From the real voltage, it is able to be used in the equation to convert from voltage to temperature (Celsius). The temperature measured has a 3% percentage of error from the actual temperature.

$$Temp = 27 - ((V - 1.45) \div 0.0537)$$

The value 27 is the temperature when the voltage is at 1.45V. The value 0.0537 is the temperature sensor slope according to the temperature sensor's datasheet. Variable V is the converted voltage from the raw voltage acquired.



**FIGURE 4.2: KL26Z with Temperature Sensor**

#### **4.4 Results**

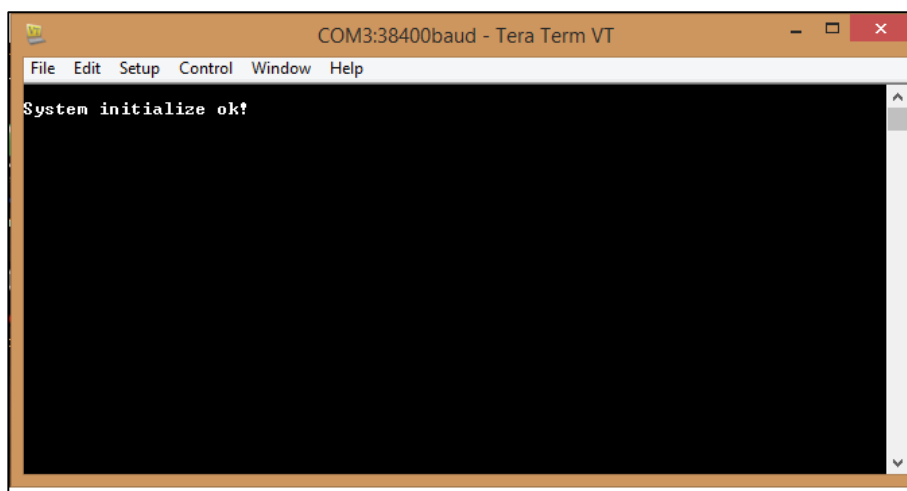
During the course of FYP I, the author had spent most of the time as possible to understand the functionality of embedded web servers and learning about coding the Freedom board. Throughout the course, the author was able to create the necessary header files for the Freedom board with the help of CoX library of course. The hardest during which was trying to accommodate the Arduino library with the Freedom Board. Most time was spent to understand the interfacing algorithms between the Wi-Fi Shield and Freedom Platform.

Several attempts were made in testing the functionality the Wi-Fi shield and the Freedom platform which had led to repeated errors. Such example of errors will be discussed below in the following section.



The author had based the source codes of the project from various examples found in Freescale Community Forums as well as MCU on Eclipse blog. Several demonstration codes were also found and attempted to run on the KL26Z despite the demonstration codes were meant for the KL25Z.

In the end, the source codes for the project was able to be generated and compiled with success. The execution of the program however was unsatisfactory. Supposedly, the KL26Z and Wi-Fi shield was instructed to initialize the system before being connected to the wireless network. However, after said system initialization, the program seems to have no further execution of the program. The systems initialization is the initialization of the drivers for the SPI peripheral of the KL26Z. Should system initialization was done, a message would appear in the console as shown in Figure 4.1.



**FIGURE 4.3: Systems Initialization Message**

The author had spent most of the time during FYP II attempting to rectify the problem by sieving through every line of code to find what had caused the program not to function. The author had even modified pin connections of the board to ensure the hardware connection was fine. However, it had proven to no avail whatsoever as the time was not forgiving and still the quandary was not resolved. Therefore, the author decided to make due of what has been achieved so far and use this project as a baseline for future projects to come regarding the Freedom platform and embedded web servers.

#### **4.5 Constraints and Problems Encountered**

During the beginning of the project, the author had difficulties of starting the project in terms of the software and hardware. Despite having undergone Structured Programming course previously, the author was not exposed to the microcontroller programming environment thoroughly. To overcome these difficulties, the author had to research on ARM based, C language programming and embedded web server from scratch. The author had use various references to learn such as example projects, related literature and online forums and tutorials.

Freescale Freedom boards are all open sourced to third-party applications for its development. Therefore, there are many Integrated Development Environments (IDEs) that are compatible with the Freedom Board. The author had difficulty of choosing which IDE to use for this project. It was a choice between IAR Embedded Workbench and CodeWarrior Development Studio. Both IDEs support the Freedom Board and have the necessary library files needed for the project. However, the author decided to use the CodeWarrior Development Studio due to its user friendly interface and the software itself provides step-by-step tutorials on how use the development studio.

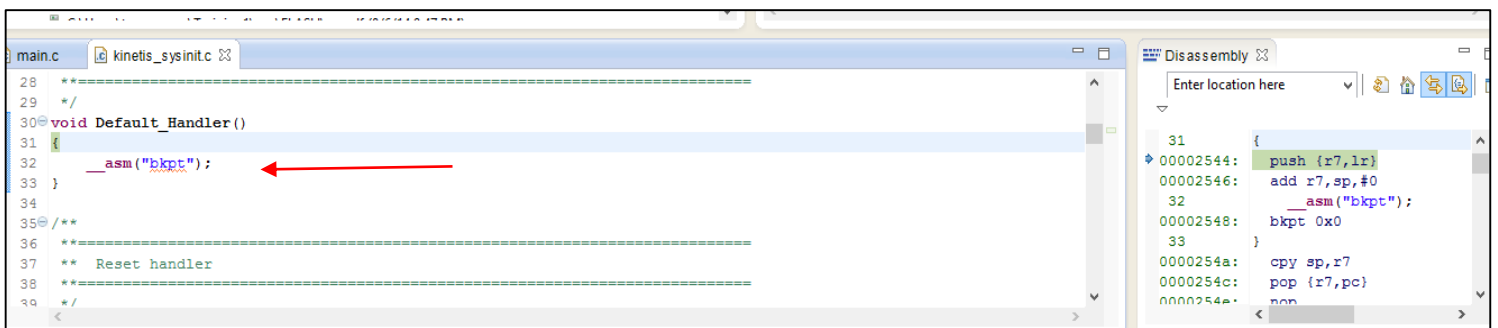
Among other complications that aroused was the references about the Freedom Board. Since the Freedom Board KL26Z is still new in its development, there are not many tutorials and references regarding the board. However, there are a lot of references regarding the KL25Z board. Both boards are similar and have the same ARM architecture but different pin outs and certain on-board components. The author had to refer to KL25Z examples and tutorials as the baseline of creating the codes for the KL26Z. From the KL25Z examples, the author translated and modified certain codes to match the suitability of the KL26Z especially regarding the pin outs. The author used the schematics of the KL25Z and KL26Z to cross-reference the pin outs of the boards. This causes the author to spend a lot of trial and error in order to get the correct code.

During the compiling and debugging process, the author encountered several error messages that were unknown. To rectify these errors, the author referred to online forums by Freescale Freedom Board developers to understand the solution to the problem and to avoid them from reoccurring.

Several problems encountered also when utilizing the libraries regarding the Wi-Fi Shield and Arduino IDE. The Wi-Fi Shield relies much of its functions and header files on Official Arduino libraries. This made it difficult to use said header files and libraries. Particularly the Arduino's **SPI.h**, is heavily used in the Wi-Fi Shield's header files. More time is required to synthesize an alternative for the header file so that would enable to interface the Wi-Fi Shield and KL26Z through SPI.

In addition to the header files of the Wi-Fi Shield, the header files also are dependent on the Arduino Library **avr/pgmspace.h** which is Arduino board program space utilities. Some modifications had to be done in order to suit compatibility for the KL26Z. Since most Arduino shields are compatible with Freedom Boards, modifications to libraries have to be made or synthesize alternatives to accommodate functionality between the shield and the Freedom board.

One particular complication that had occurred was during the debugging process, the program would stop half way. It was discovered from the disassembly of the program, that the program had entered a Default Interrupt Handler which led to a **breakpoint (bkpt)** instruction. This instruction causes the ARM to enter a HardFault.



**FIGURE 4.4: Default Interrupt Handler**

The probable causes of this fault are enabling improper interrupt declaration and mathematical error.

This complication had cost the author a substantial amount of time. The author had to go through every line of code used in the program in order find the root cause of the problem. After thorough searching the codes and online forums, it was found that the Freedom board was not able to execute **printf** without having to initialize a low level Universal Asynchronous Receiver Transmitter (UART). The low level UART is needed to display printf in a console.

## CHAPTER 5

### CONCLUSION AND RECOMMENDATION

#### 5.1 Recommendation

Given the lack of resources and examples for KL26Z, most of the steps taken during the duration of the project are referred from projects and tutorials from the internet regarding the KL25Z. Therefore the author would document the relevant sources that might help for future projects regarding the Freescale Freedom Board Kinetis series. The sources include sample projects, video tutorials and relevant data sheets of the KL25Z and KL26Z.

It is also recommended to use an external real time clock with an independent power source such as a coin battery in order to add a real time clock feature to the embedded web server. If an on-board real time clock function is implemented, it would only work until to the point where the board is disconnected from a power source. When the board restarts the time is reset and the user would have to reset the clock again which proves to be a tedious chore to do. The external real time clock serves to solve this predicament.

As a suggestion for future projects regarding the Freedom Platform and embedded web servers to create a project to use the embedded web server to remotely control a servo along with other peripherals. This is to further explore the capabilities of the Freedom platform in handling multiple peripherals in a single application. For example, using the Freedom platform and servos to create a remote controlled robot by means of an embedded web server and web browser as the user interface to control the robot.

For security and remote access purposes, it is recommended for the embedded web server to use a Dynamic Domain Name System (DDNS) service as a replacement for the user fixed IP address. By hosting a domain, the operator would be able to access the web server without a having to be in the same wireless network. The operator would only have to enter the host name to access the web server.

The user interface can also be improved by using elements and coding styles of Cascading Style Sheets (CSS) as well as JavaScript. This is to give the user interface much more functionality and complexity. The aforementioned coding elements should be supported by current generation of web browsers.

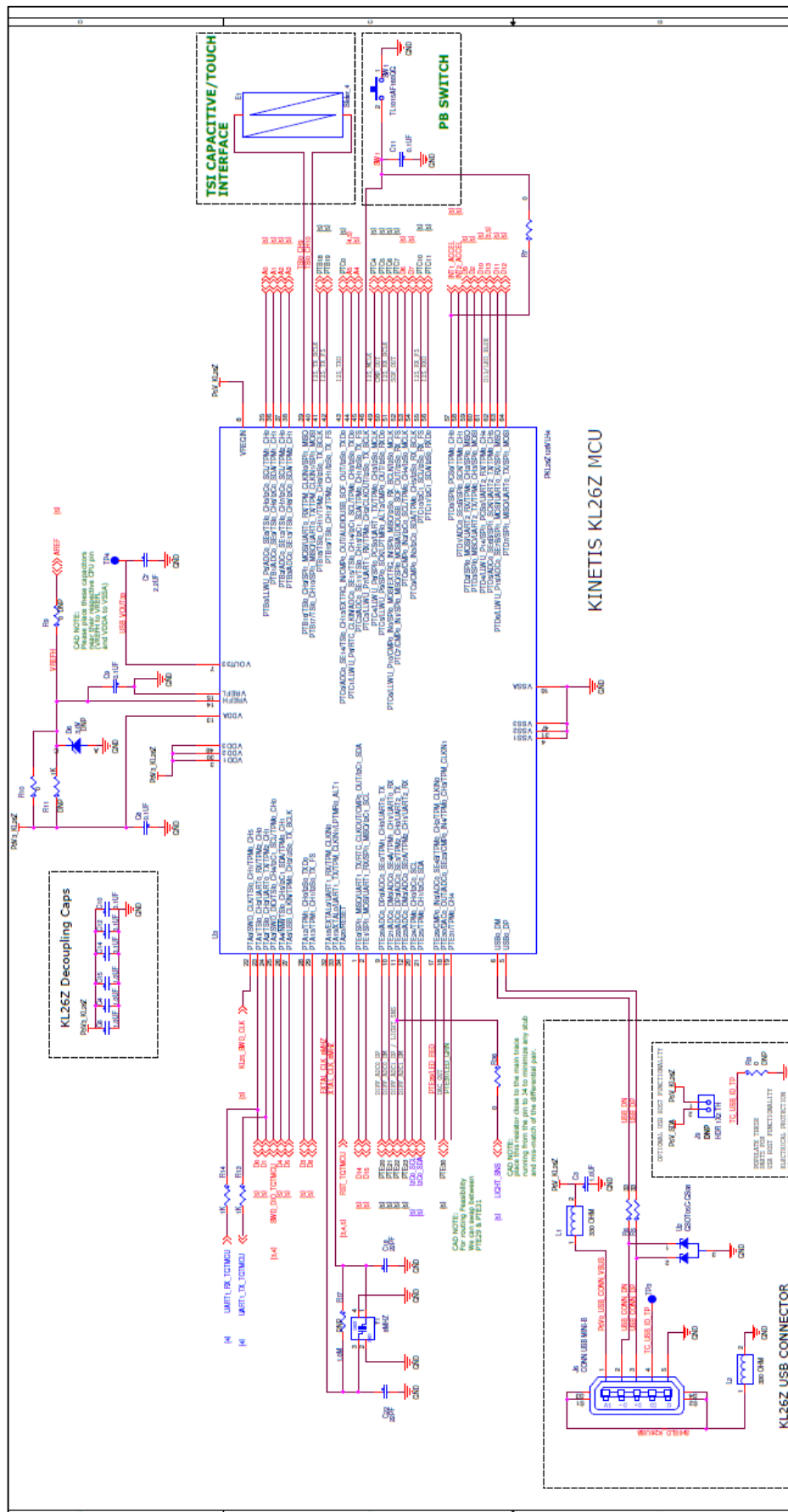
## **5.2 Conclusion**

The project was not able to operate successfully and there are substantial room for improvement. This paper serves as a reference point for future applications regarding the Freedom platform. Despite the lack of experience in embedded systems, the author had achieved an understanding regarding the technical aspects involved in creating an embedded system such as an embedded web server. Although the objectives of the project was not achieved, hopefully this paper would assist future students in continuing this project.

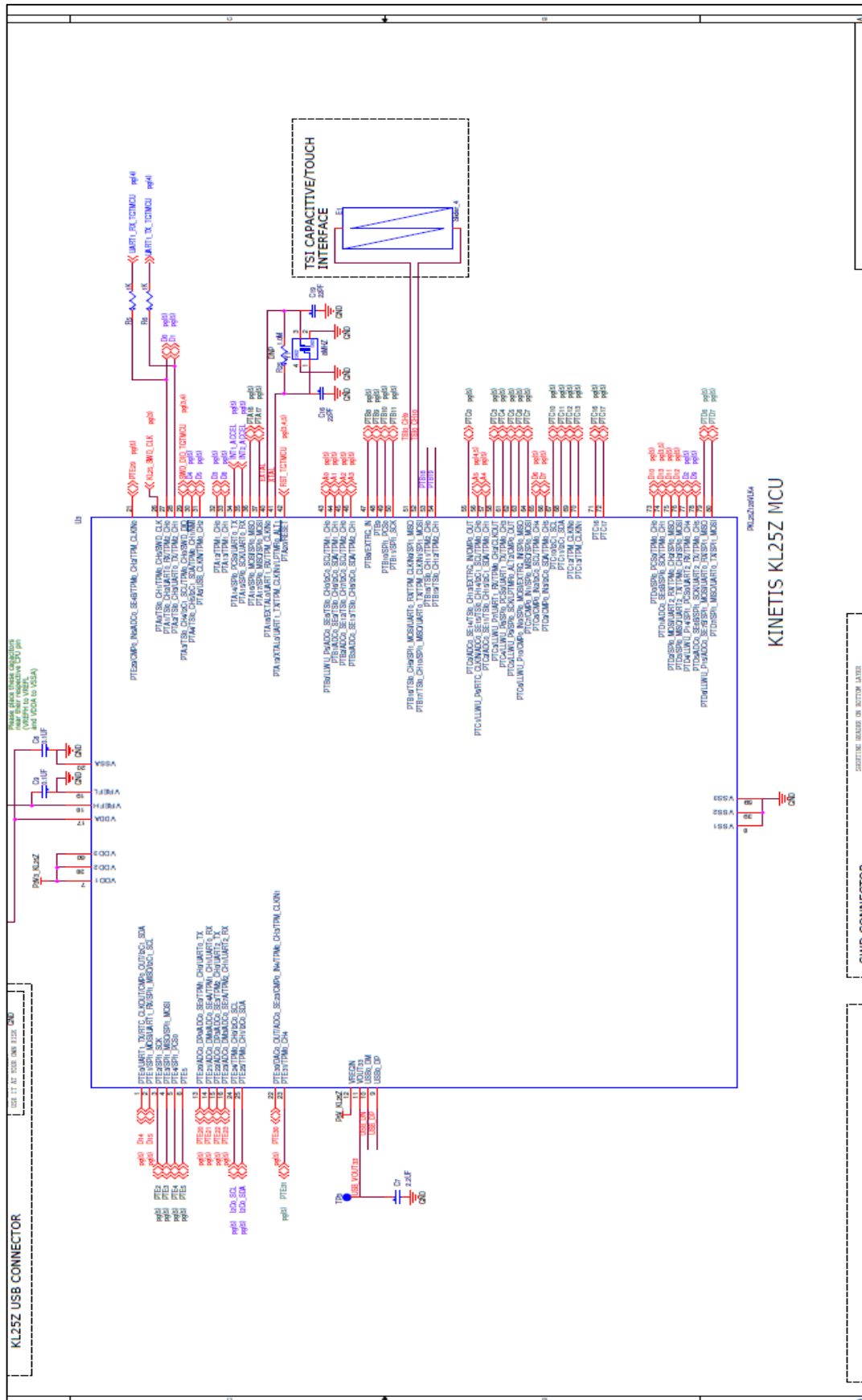
## REFERENCES

- [1] F. Y. Limpraptono, H. Sudibyo, A. A. P. Ratna, and A. S. Arifin, "The design of embedded web server for remote laboratories microcontroller system experiment." pp. 1198-1202.
- [2] K. Takaya, "Transputer-like multicore parallel processing on the array of ARM Cortex-M0 microprocessors." pp. 1-4.
- [3] M. Can Filibeli, O. Ozkasap, and M. Reha Civanlar, "Embedded web server-based home appliance networks," *Journal of Network and Computer Applications*, vol. 30, no. 2, pp. 499-514, 2007.
- [4] Y. Guangyou, L. Ming, and Z. Shuangqing, "Remote measuring and control terminal based on linux platform and embedded web server." pp. 3-104-3-108.
- [5] L. Yakun, and C. Xiaodong, "Design and implementation of embedded Web server based on arm and Linux." pp. 316-319.
- [6] F. Lidong, Z. Peiqiang, and D. Qian, "A lightweight embedded Web server for non-PC device." pp. 4756-4758.
- [7] M.-h. Wu, "Research for the Embedded WEB Server." pp. 776-779.
- [8] K. Qinma, H. Hong, and W. Hongrun, "Study on Embedded Web Server and Realization." pp. 675-678.
- [9] M. Poongothai, "ARM Embedded Web Server Based on DAC System." pp. 1-5.
- [10] L. Jian-feng, W. Chun-yi, and H. Jie, "A High Performance Data Storage Method for Embedded Linux Real-time Database in Power Systems," *Energy Procedia*, vol. 16, pp. 883-888, 2012.
- [11] D. C. Karia, V. Adajania, M. Agrawal, and S. Dandekar, "Embedded web server application based automation and monitoring system." pp. 634-637.
- [12] P. A. C. R. Gopi Krishna S, K. Gerard Joe Nigel, "Web based Remote Accessing of Medical Devices with ARM Cortex-M3," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 2, no. 3, pp. 51-54, July 2013, 2013.

# APPENDIX A: KL26Z SCHEMATIC

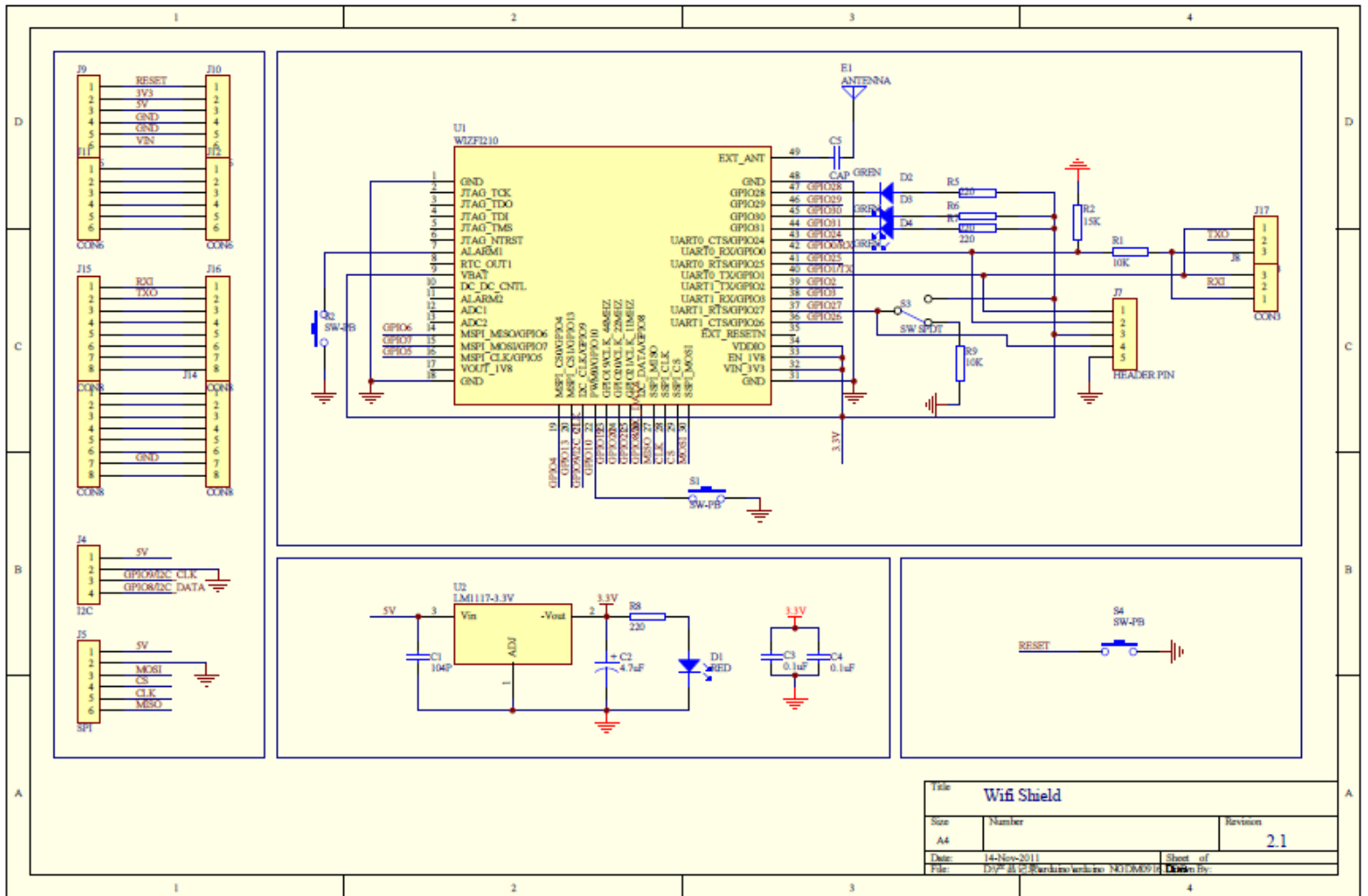


# APPENDIX B: KL25Z SCHEMATIC





# APPENDIX C: WIZFI SHIELD SCHEMATIC



## APPENDIX D: ARDUINO LIBRARY CODE

```
#include <SPI.h>
#include <WiFi.h>

char ssid[] = "yourNetwork"; // your network SSID (name)
char pass[] = "secretPassword"; // your network password
int keyIndex = 0; // your network key Index number (needed only for WEP)

int status = WL_IDLE_STATUS;

WiFiServer server(80);

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }
  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }
  String fv = WiFi.firmwareVersion();
  if ( fv != "1.1.0" )
    Serial.println("Please upgrade the firmware");
  // attempt to connect to Wifi network:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);
    // wait 10 seconds for connection:
```

```

    delay(10000);
}
server.begin();
// you're connected now, so print out the status:
printWifiStatus();
}
void loop() {
    // listen for incoming clients
    WiFiClient client = server.available();
    if (client) {
        Serial.println("new client");
        // an http request ends with a blank line
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                Serial.write(c);
                // if you've gotten to the end of the line (received a newline
                // character) and the line is blank, the http request has ended,
                // so you can send a reply
                if (c == '\n' && currentLineIsBlank) {
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println("Connection: close"); // the connection will be closed after completion of the
response
                    client.println("Refresh: 5"); // refresh the page automatically every 5 sec
                    client.println();
                    client.println("<!DOCTYPE HTML>");
                    client.println("<html>");
                    // output the value of each analog input pin
                    for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
                        int sensorReading = analogRead(analogChannel);
                        client.print("analog input ");
                        client.print(analogChannel);

```

```

        client.print(" is ");
        client.print(sensorReading);
        client.println("<br />");
    }
    client.println("</html>");
    break;
}
if (c == '\n') {
    // you're starting a new line
    currentLineIsBlank = true;
}
else if (c != '\r') {
    // you've gotten a character on the current line
    currentLineIsBlank = false;
} } }

// give the web browser time to receive the data
delay(1);

// close the connection:
client.stop();
Serial.println("client disconnected");
}
}

void printWifiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your WiFi shield's IP address:
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI):");
    Serial.print(rssi);
    Serial.println(" dBm");}

```

## APPENDIX D: MODIFIED CODE FOR WIZFI SHIELD

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <SPI.h>

#include <WizFi2x0.h>

#include <WizFiClient.h>

#include <WizFiServer.h>

#include <TimerOne.h>

#define SSID "" // SSID of your AP

#define Key "" // Key or Passphrase

// Wi-Fi security option (NO_SECURITY, WEP_SECURITY, WPA_SECURITY, WPA2PSK_SECURITY)

//#define Security WPA_SECURITY

#define MAX_SOCKET_NUM 4

unsigned int SrcPort = 80;

WizFi2x0Class myWizFi;

WizFiClient myClient[MAX_SOCKET_NUM]; // (SIP, ServerPort);

WizFiServer myServer(SrcPort);

boolean Wifi_setup = false;

// 1msec Timer

void Timer1_ISR()

{

    myWizFi.ReplyCheckTimer.CheckIsTimeout();

}

void setup() {

    byte retval, i;

    Serial.begin(9600);

    Serial.println("\r\nSerial Init");

    for(i=0; i<MAX_SOCKET_NUM; i++)

        myClient[i] = WizFiClient();
```

```

myWizFi.begin();

// Timer1 Initialize
Timer1.initialize(1000); // 1msec

Timer1.attachInterrupt(Timer1_ISR);

myWizFi.SendSync();

myWizFi.ReplyCheckTimer.TimerStart(3000);

Serial.println("Send Sync data");

while(1)
{
    if(myWizFi.CheckSyncReply())
    {
        myWizFi.ReplyCheckTimer.TimerStop();

        Serial.println("Rcvd Sync data");

        break;
    }

    if(myWizFi.ReplyCheckTimer.GetIsTimeout())
    {
        Serial.println("Rcving Sync Timeout!!");

        return;
    } }

// AP association

while(1)
{
    byte tmpstr[32];

    retval = myWizFi.associate(SSID, Key, Security, true);

    if(retval == 1){

        myWizFi.GetSrcIPAddr(tmpstr);

        Serial.println("WizFi2xo AP Associated");

        Serial.print("MY IPAddress: ");

        Serial.println((char *)tmpstr);

        Wifi_setup = true;

        break;

    }else{

```

```

        Serial.println("AP association Failed");} }
if(myServer.begin())
    Serial.println("Server Listen OK");
else
    Serial.println("Server Listen Failed");
}
void loop()
{
    uint8_t retval, i;
    byte rcvdBuf[129];
    memset(rcvdBuf, 0, 129);
    if(Wifi_setup)
    {
        myWizFi.RcvPacket();
        for(i=0; i<MAX SOCK_NUM; i++)
        {
            if(myClient[i].available()){
                retval = myClient[i].read(rcvdBuf);
                if(retval > 0)
                {
                    Serial.print("CID[");
                    Serial.print((char)myClient[i].GetCID());
                    Serial.print("]");
                    Serial.println((char *)rcvdBuf);

                    if((rcvdBuf[retval - 1] == 0x0A) && (rcvdBuf[retval - 2] == 0x0D) && (rcvdBuf[retval - 3] == 0x0A)
&& (rcvdBuf[retval - 4] == 0x0D))
                    {
                        Serial.print("Receiving Completed");

                        myClient[i].write((byte *)"HTTP/1.1 200 OK\r\n");
                        myClient[i].write((byte *)"Content-Type: text/html\r\n");
                        myClient[i].write((byte *)"\r\n");
                        myClient[i].write((byte *)"Hello World !\r\n");

                        delay(100);

                        myClient[i].disconnect();    } } } } }

```

# APPENDIX E: KL26Z PINOUTS

FRDM-KL26Z Pins				KL26Z Pins # (K1 UCP)									
Onboard Usage	IO Header & Pin Num	Arduino™ R3 Pin Name	FRDM-KL26Z Pin Name	ALTB	ALTI	ALTZ	ALT3	ALT4	ALTS	ALT6	ALTT	Reset Status/Function	
Power	J2 20	D15	PT21	VDD	PT20	SPI_MISO	UART1_TX	TRQ_CLKOUT	CIAPQ_OUT	IC21_SQA		DISABLED	
Power	–	–	VDD	VSS	PT21	SPI_MISO	UART1_RX			IC21_SQA		DISABLED	
Power	–	–	VSS	VSS									
USB	–	–	USB0_DP	USB0_DP								USB0_DP	
USB	–	–	USB0_DMI	USB0_DMI								USB0_DMI	
2.2µF cap	–	–	VOUT33	VOUT33								VOUT33	
USB (VBUS (SV)	–	–	VRESIN	VRESIN								VRESIN	
–	J4 01	–	PT20	AD0Q_DP/AD0Q_SE0	PT20		TPM1_CH0	UART1_TX				AD0Q_DP/AD0Q_SE0	
–	J4 03	–	PT21	AD0Q_DM/AD0Q_SE4	PT21		TPM1_CH1	UART1_RX				AD0Q_DM/AD0Q_SE4	
–	J4 05	–	PT22	AD0Q_DP/AD0Q_SE3	PT22		TPM2_CH0	UART2_TX				AD0Q_DP/AD0Q_SE3	
–	J4 07	–	PT23	AD0Q_DM/AD0Q_SE7	PT23		TPM2_CH1	UART2_RX				AD0Q_DM/AD0Q_SE7	
Power	–	–	VDDA	VDDA								VDDA	
Power	J2 16	AREF*	VREFH	VREFH								VREFH	
Power	–	–	VREFL	VREFL								VREFL	
Power	–	–	VSSA	VSSA								VSSA	
Red LED	–	–	PT29	CIAPQ_IN/AD0Q_SE4D	PT29		TPM0_CH2	TPM_CLKIN0				CIAPQ_IN/AD0Q_SE4D	
–	J4 11	–	PT30	DMQ0_OUT/AD0Q_SE29/CIAPQ_IN4	PT30		TPM0_CH3	TPM_CLKIN1				DMQ0_OUT/AD0Q_SE29/CIAPQ_IN4	
Green LED	–	–	PT31		PT31		TPM0_CH4					DISABLED	
Inertial Sensor I2C	–	–	PT24		PT24		TPM0_CH0					DISABLED	
Inertial Sensor I2C	–	–	PT25		PT25		TPM0_CH1					DISABLED	
Debug (SMD_CLK)	–	–	PT40	TS10_CH1	PT40		TPM0_CH5					DISABLED	
–	J1 02	D0	PTA1	TS10_CH2	PTA1	UART0_RX	TPM2_CH0					DISABLED	
–	J1 04	D1	PTA2	TS10_CH3	PTA2	UART0_TX	TPM2_CH1					DISABLED	
Debug (SMD_DIO)	–	–	PTA3	TS10_CH4	PTA3	IC21_SQA	TPM0_CH0					SMD_DIO	
–	J1 10	D4	PTA4	TS10_CH5	PTA4	IC21_SQA	TPM0_CH1					ENABLE	
–	J1 12	D5	PTA5		PTA5	USB_CLKIN	TPM0_CH2					DISABLED	
–	J1 08	D3	PTA12		PTA12		TPM1_CH0					DISABLED	
–	J2 02	D8	PTA15		PTA15		TPM1_CH1					DISABLED	
Power	–	–	VDD	VDD								VDD	
Power	–	–	VSS	VSS								VSS	
8MHz XTAL	–	–	PTA18	EXTAL0	PTA18	UART1_RX	TPM_CLKIN0					EXTAL0	
8MHz XTAL	–	–	PTA19	XTAL0	PTA19	UART1_TX	TPM_CLKIN1					XTAL0	
Reset	J3 05	–	PTA20		PTA20							RESET_B	
–	J4 02	A0	PTB0	AD0Q_SE8/TS10_CH0	PTB0/LLWUP_P5	IC20_SQA	TPM1_CH0					AD0Q_SE8/TS10_CH0	
–	J4 04	A1	PTB1	AD0Q_SE9/TS10_CH6	PTB1	IC20_SQA	TPM1_CH1					AD0Q_SE9/TS10_CH6	
–	J4 06	A2	PTB2	AD0Q_SE10/TS10_CH7	PTB2	IC20_SQA	TPM2_CH0					AD0Q_SE10/TS10_CH7	
–	J4 08	A3	PTB3	AD0Q_SE13/TS10_CH8	PTB3	IC20_SQA	TPM2_CH1					AD0Q_SE13/TS10_CH8	
Touch Slider	–	–	PTB16	TS10_CH9	PTB16	SPI_MOSI	UART0_RX	TPM_CLKIN0	SPI_MISO			TS10_CH9	
Touch Slider	–	–	PTB17	TS10_CH10	PTB17	SPI_MISO	UART0_TX	TPM_CLKIN1	SPI_MOSI			TS10_CH10	
–	J1 01	–	PTB18	TS10_CH11	PTB18		TPM2_CH0					TS10_CH11	
–	J1 03	–	PTB19	TS10_CH12	PTB19		TPM2_CH1					TS10_CH12	