CONTROLLING AND MONITORING OF SERVO MOTOR THROUGH EMBEDDED HTTP SERVER ON ARM BOARD

By

MOHD HARIZ BIN MOHD RIFAN

Submitted to the Department of Electrical & Electronic Engineering
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical and Electronics Engineering)

UniversitiTeknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Controlling and Monitoring of Servo Motor through Embedded HTTP Server on ARM Board

by

Mohd Hariz bin Mohd Rifan

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical and Electronics Engineering)

Approved by,

____

(Dr. Mohd Zuki bin Yusoff)

Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

May 2013

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.

(MOHD HARIZ BIN MOHD RIFAN)

ABSTRACT

The positioning of servo motor is controlled by sourcing the servo with PWM signal of varying pulse width. The pulse width of the PWM signal can be manipulated through a web interface hosted on ARM board programmed as an embedded HTTP server where control application by an embedded system can be monitored and managed remotely. Embedded HTTP server network stack is based on modified TCP/IP protocol suite where only important features are used. A web server for user interface is created using HTML and JavaScript is hosted together with the network stack. The ARM board is equipped with RTOS to enable real time control response between the interface and the servo motor. Two tasks namely PWM and TCP/IP task are configured to enable execution from RTOS based on their priorities. Under executions, a slider inside web interface controls the turning of servo motor with the degree of rotation is displayed on the web interface.

ACKNOWLEDGEMENTS

I would like to dedicate this page in acknowledging the help, guidance, and support, whether directly or indirectly, from project supervisors, family, and fellow friends 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 work under his supervision and providing me crucial guidelines and assistance for my project work. I would also like to thank Mr. Mohd Azman bin Zakariya, my co-supervisor, for introducing me to the wide possibilities of embedded systems with his knowledge and ideas. Not to forget, I would like to thank Mr. Duc Ngoc Tran, a UTP postgraduate student for sharing his expertise and knowledge in computer engineering besides being patient while assisting me throughout my project work.

Many thanks to Dr. Mohamad Naufal bin Mohamad Saad, Ms. Zazilah May, and Ir. Azlin Aziz for their participation in my project viva besides giving me the essential feedbacks and motivation to complete my project work. Special thanks to helpful forum users from online forum such as Cytron, Stackoverflow, and FreeRTOS for their kindness in helping me resolving problems I had encountered during project work.

I would like to take this opportunity to thank my family for their support, and to all of my friends who contributed to the project. Your support, assistance, and help are duly acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADC	Analog-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
JTAG	Joint Test Action Group
kB	Kilobyte
LED	Light Emitting Diode
MAC	Media Access Control
PC	Personal Computer
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RTOS	Real Time Operating System
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SWD	Serial Wire Debug
SWO	Square Wave Output
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus

CHAPTER 1
INTRODUCTION

1.1 Background

In recent years, the thorough development of Advance RISC Machine (ARM) architecture had make it possible for development of more complex but highly capable and powerful ARM-based products. The manufacturing of development board with ARM architecture chips on board together with on-chip memory and peripherals such as Ethernet capability had enabled developers the power to produce something that is impossible before. As a 32-bit Reduced Instruction Set Computing (RISC) architecture with high clock speed, huge memory capacity, and competitive price to other standards microcontrollers available in the market, the ARM architecture had been increasingly popular in recent years, slowly overtaking the 8-bit and 16-bit microcontrollers' popularity.

The large memory capacity of a 32-bit ARM chip enabled board had made it possible for the implementation of embedded web server on Real Time Operating System (RTOS) into a microcontroller, thus defining a new chapter in Human-Machine Interface (HMI) design which enable operator to not only control the output of an ARM board remotely from a different location, but also to gather data and information all through a web server at ease [1].

By doing this project, author would like to show the possibility of using ARM board as alternative to the conventional controller used in industrial sector such as programmable logic controller (PLC), PC based control system, electronic continuous control system, and others, in controlling, monitoring and automation for industry use.

1.2 Problem Statement

Embedded system can be used in control application where set of input and outputs are running through the system to control, for instance, several servo motors over virtual network. For real time applications, embedded systems need to be equipped with Real Time Operating System (RTOS) to manage tasks in the microprocessor by scheduling and to process data immediately upon receiving information, which is critical for critical control systems, in which for this project, for accurate tuning of servo motors.

For remote control action of the servo motors, a web server with graphical user interface (GUI) would provide input options, in form of slider bars, to the ARM microcontroller as implemented on a Hypertext Transfer Protocol (HTTP) web server in the ARM board, before a PWM signal is produced at the output pin of the ARM board to adjust the servo arm position, as a result of moving the slider bar.

Web server software packages are normally made available as free open source, such as micro IP (µIP) network stack, which enables developers to modify the readily made TCP/IP network stack by other web server developers to suits application need depending on respective manufacturers' peripheral chip configuration and toolchain used in developing the program. This is also applicable to RTOS such as µOS-II and FreeRTOS, which would be used to manage different tasks processed in the microcontroller efficiently by managing the amount of memory used and the interrupt action of the microcontroller.

1.3 Objectives

The main objective for this project is to control the positioning of several servo motors, horizontally and vertically through embedded HTTP server on ARM board. The positioning of the servos can be used for tuning and controlling the position of antenna mounted on the servo motor. The ARM board used in this project is Hy-LandTiger LPC1768 equipped with 32-bit ARM Cortex-M3 microcontroller, 512 kB flash memory with 64 kB SRAM data memory with Ethernet Media Access Control (MAC) capability. The board will also have several Pulse Width Modulation (PWM) output to control the positioning of the servo motors. For further development, the project would harness the board capability to monitor the degree of rotation of the antenna through a web server. Overall, the objectives are:

- To remotely control peripherals and/or monitor processes by using ARM board
- To implement RTOS into ARM board to manage and control all the processes in the ARM board
- To implement an embedded web server on ARM board, specifically the Hy-LandTiger LPC1768

1.4 Scope of Study

In this project, there are several main subjects under investigation which would later be discussed in Chapter 3: Methodology. The scopes of study in this project are:

i. Implementation of fully functioning HTTP web server running on ARM board
ii. Controlling peripheral (servo motors) through an embedded web server on ARM board to control the vertical and horizontal positioning of an antenna
iii. Data monitoring, i.e.: degree of servo motors rotation on their respective axis, and display the data on an user interface
iv. Implementation of RTOS in Cortex-M3 ARM device

1.5 Relevancy of the Project

In these modern days, multitasking and ability to conduct work through the tips of your finger are very important to keep up with fast working environment besides increasing the effectiveness in work. By having remotely controlled peripherals besides the ability to monitor processes, one will able to do monitoring and control in a comfort of a room, or from far away. With the capability of Ethernet connection, more importantly it vast region coverage, this project would purpose a new dimension to human-machine interface where operators can work far away from machines.

1.6 Feasibility of the Project

In today's industrial sector, there are a lot of new technological advances and improvements taking places daily to cope with industry demands. One of it is in control and automation section which is important in running and controlling machineries with a little intervention from operators.

By doing this project, the author would show the possibility of using ARM board as a control system in industrial area on top of introducing new kind of human-machine interface using embedded web server that would enables remote control and monitoring to the system concerned.

On top of that, with current advancement of today's ARM processors, combined with lower cost compared to current industrial control systems, the ARM board would be a great choice in reducing costs without compromising the quality or the ability to get the automation done by the system.

With the remote controlling of servo motor, one would be able to control servos located in hard-to-reach areas such as inside vessels, piping, or high areas. This would eliminate job risks to human operator such as hazard from falling from high areas or hazard from confined spaces.

CHAPTER 2
LITERATURE REVIEW AND THEORY

2.1 Real Time Operating System (RTOS)

The major reason of using RTOS is to enable multitasking. Few years back, only one task is executed at a time. With recent years of development of multicore processor, for example LPC4300, had enable multiple task execution at a time, but what about single core processor? By using RTOS, we can manage all tasks to imitate multitask whilst in reality only one task is executed at a time.

RTOS is used in embedded application as an operating system dealing with real-time applications where tasks to be done are highly critical, deterministic, and timely. Examples of popular RTOS include µCOS, Windows CE, RTLinux, and FreeRTOS. Two notable key features of RTOS includes short thread switching latency where switching process between tasks should be done in short time and short interrupt latency where the execution of interrupt instruction should also be short. There are generally divided into three classifications [2].

i. Hard RTOS – This type of RTOS is where task execution had to strictly obey given deadline where missing deadline could lead to disaster. Usually used in critical systems such as machine emergency shutdown system.
ii. Firm RTOS – same to the hard RTOS, executions of tasks need to be done in time for firm RTOS, but missing a deadline would not lead to disastrous event but only undesirable outputs.
iii. Soft RTOS – Missing a dateline is not a problem for soft RTOS.

RTOS should be efficiently programmed and implemented in a way that supports both real time applications and time based scheduling which is the key to RTOS. The main player in RTOS is kernel. In a period of time, the kernel in RTOS forward CPU attention to specific task besides continuously checks for task priority and arranges requests from schedules and tasks.

On top of that, the kernel should be able to do multitasking and kernel preemption, in which task currently being processed by the kernel can be interrupted and resumed later. Basically, the kernel is responsible in creating, deleting, changing the priority and state of a task, besides managing resources and hardware monitoring. Task with higher priority would be able to take over the CPU attention for execution, interrupting execution of lower priority task in its way.

There are three basic functions of a RTOS taken into design consideration. They are listed as below [3]:

1. Scheduler

   With RTOS scheduler, the processor would able to give impression that all tasks programmed into a program are running at a time while actually the scheduler would switch between each tasks execution to make it appear as if everything is running at once. The secret to timely and deterministic feature on RTOS lies on its advanced scheduling algorithm. Scheduler decides the switching operation of tasks which exists in three states:

   i. Ready to run – a state before a task is running. Task should have all the resources to run in this state.
   ii. Running – Task running on the CPU is in running state.
   iii. Blocked – A task is in blocked state when it does not have enough resources to run. For example, waiting for I/O action.

Scheduler schedules task based on three algorithms scheme:

i. Cooperative scheduling – task will be running in cooperative scheduling until execution is complete.
ii. Preemptive scheduling – Each tasks are given distinctive priority level for execution.
iii. Round Robin scheduling – Every tasks are given respective fixed time for execution where missing dateline would have a task to wait for next turn.

2. RTOS Services

   Services are provided by the kernel to support task in having CPU attention. Some of the RTOS services are interrupt handling, process management and inter-process communication services.

3. Messaging

   Messaging services are used as a way of communication between tasks and with other systems. Some of the messaging services and their usage are as following:

   i. Semaphores – synchronize access to shared resources
   ii. Event flags – synchronize inter-task operations
   iii. Mailboxes, pipes, and message queues – send messages to tasks

   Overall, the main reasons of developer using RTOS is:

   a) The capabilities to prioritize tasks in ensuring real-time constraints of the application are met. On top of that, RTOS would assists in programming events based on applications in which excessive use of pooling routines are avoided, thus reducing the number of cycles of CPU needed and improves the efficiency of the system with reduction in power consumption [4].
   b) RTOS programming is well-structured, thus it would make it easier for developer to code, debug, and maintain the program.

2.2 Embedded Web Server

An embedded web server is a microcontroller with Internet software suite and application code implemented into it for monitoring and controlling systems [5]. With an embedded web server, a microcontroller can be accessed remotely from a web browser. The embedded system is also able to host static and dynamic web documents to web browsers.

When talking about internet communication, the most significant thing would be TCP/IP, also known as the Internet Protocol Suite, which had been a standard protocol used in web page transfers, e-mail communication, and peer-to-peer networking [6]. The most common protocol in the suite is the Transmission Control Protocol (TCP) which establishes connection and enables data exchange between two machines.

In an established TCP connection, data is transferred between machines in form of raw bytes send in a group called packets. In this project, the web server loaded into the ARM board was given specified IP address. It then waits for connection on a port, also known as listening. Since the programming language we used is able to open TCP connections, there is no higher protocol than TCP in the system.

In a network, connected machines are identified by IP addresses, usually in form of 4 numbers separated by periods in between. On top of that, machines establish connection on number of ports which provide services to incoming data in which on modern computer systems, the ports are usually standardized. For instance, port 80 is always used for web server application.

With recent development where code size and memory requirement are more optimized [6], TCP/IP stack are now able to be ported into embedded systems, giving the ability for connection with other hosts in intranet or internet network. TCP/IP stack now can be minimized to important set of features only such as TCP, IP, ICMP and UDP protocols with ability to handle single network interface.

As a finite state machine (FSM), embedded web server processes HTTP request as a sequence of discrete flow. Fig. 1 below shows FSM flow.

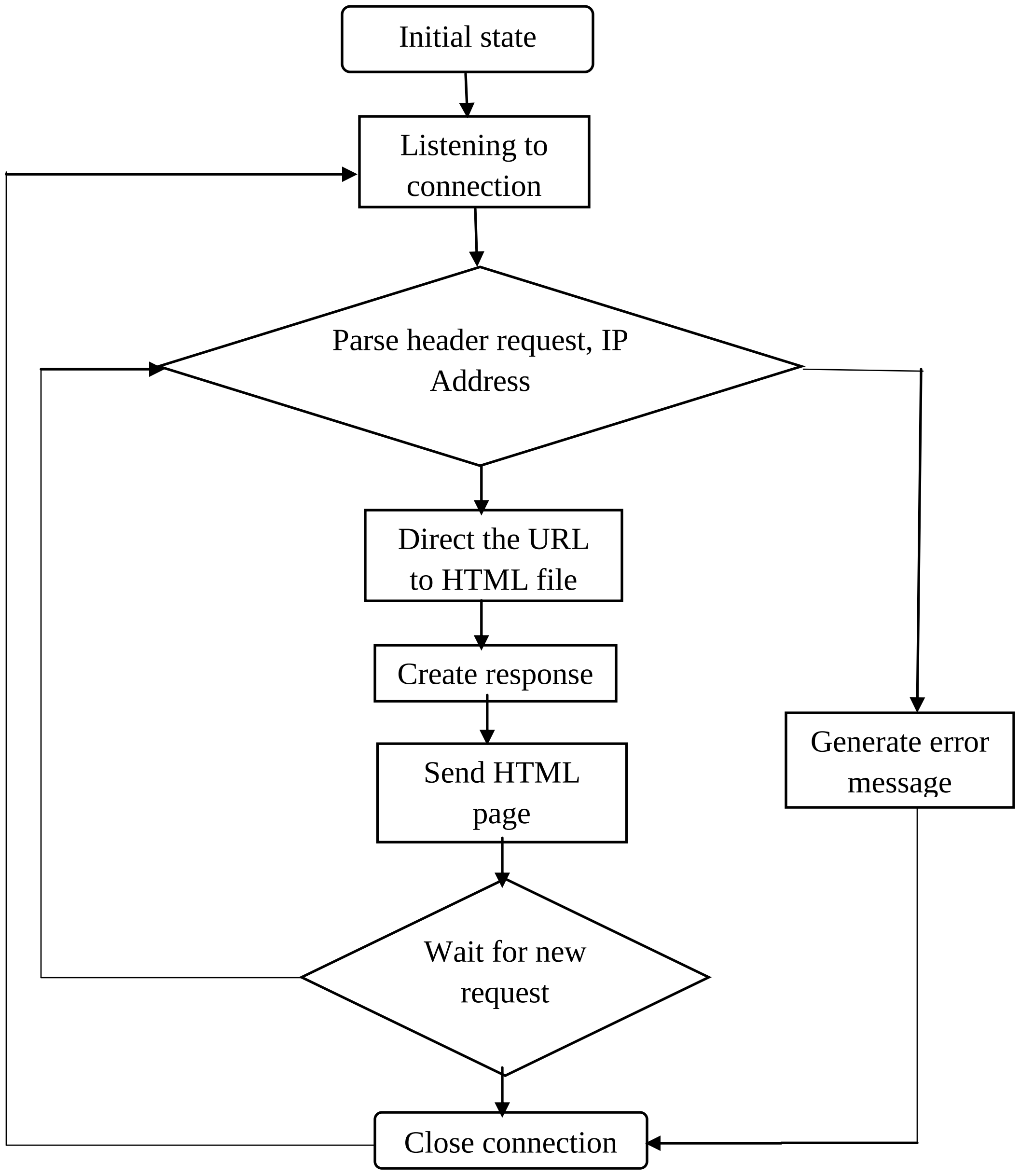


Figure 1: Embedded web server process

2.3 ARM board

Powerful microcontroller board are required to handle not only heavy task of controlling servo motor positioning but also to handle large processing load involving networking and embedded web server without distracting microcontroller primary application. The LPC1768 board not only are equipped with powerful 32-bit ARM Cortex-M3 microcontroller, but also the Ethernet media access controller to suit up for remote application requirements which could be achieved by connecting to a network through RJ45 jack.

In addition, the Cortex-M3 had advantages of improved code density and program execution efficiency by 25% over other 32-bit RISC, allowing more performance while using less memory [7]. With the high ratio of 64 kB SRAM data memory to 512 kB flash memory, the LPC1768 should be able to handle various complex networking and data-processing routines. The 68 kB SRAM is further divided into 32 kB of SRAM on the CPU equipped with local code for fast CPU access and two 16 kB SRAM for general purpose storage and Ethernet use [8].

On top of that, the LPC1768 is also equipped with several channels of pulse-width modulators (PWM) with timer synchronization for accurate edges positioning and quadrature encoder inputs to read precise positioning of motor together with integrated fault protection for safety of operation in low latency conditions.

The performance of the LPC1768 is highly attributed to the architecture features that is mostly can be found in RISC microprocessors. Fig. 2 below shows what are there in the Cortex-M3 based LPC1768 board.

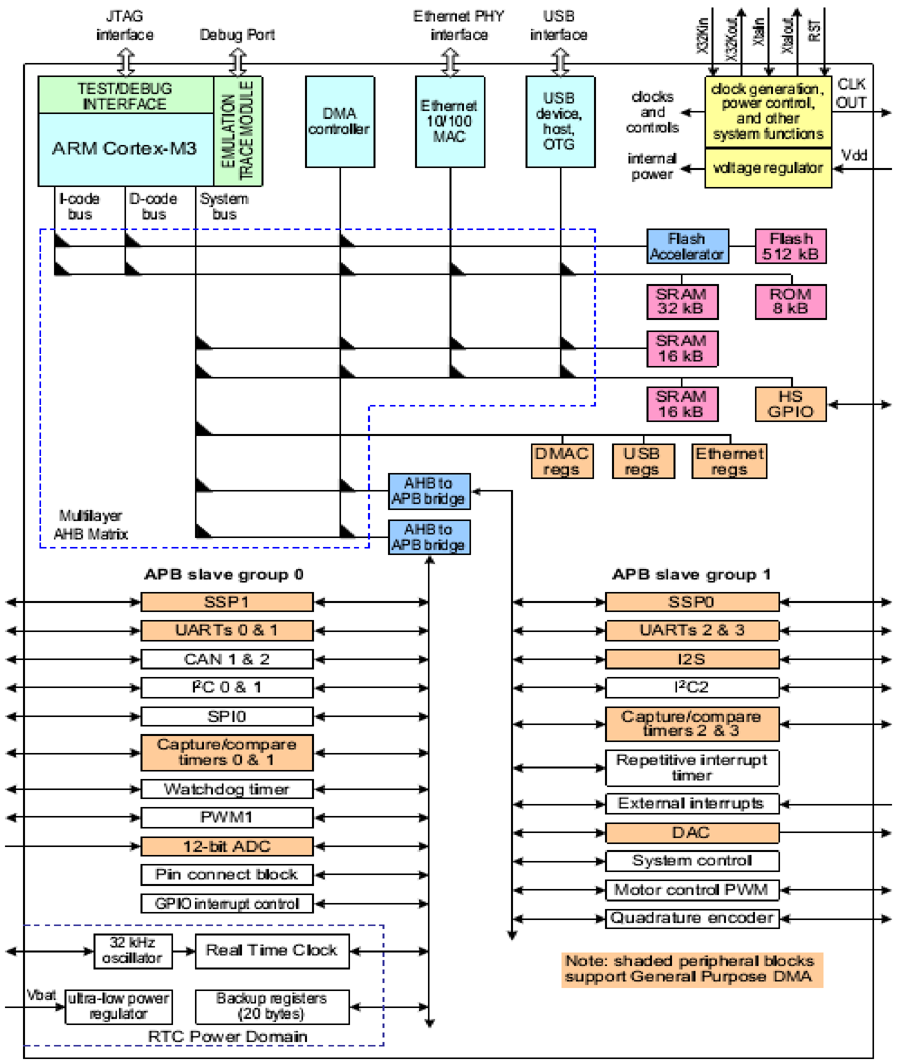


Figure 2: Simplified block diagram for LPC1768

First and foremost, the LPC1768 heart is the Cortex-M3 microprocessor which is of Harvard architecture, meaning that the instruction and data bus are separated, thus allowing instructions and data access to take place at the same time from separate memories. However, the two buses share the same memory space, meaning that there are now two separate 32-bit buses that allow 8GB of memory space.

The separation of these two buses promotes improvement on bandwidth comparing to the traditional von Neumann architecture where program and data are fetched from the same memory through same bus. Besides separating instruction and data bus allow instructions to be sized differently than the 8-bit wide data word.

2.4 Peripherals

2.4.1 Servo Motor

Servomotors integrate motor with dedicated rotary encoder. Industrial servomotors are more focused into servomechanism using feedback from encoder, usually a PID controller that determines the position or rotation of the electrical motor component. The closed-loop control is the sector that differentiates between servomotors and stepper motors, which uses open-loop configuration although both motors embrace servomechanisms.

For this project, the author is more focused into lower end servomotors type called the radio-controlled servo motor, which has an electrical motor that is connected to a potentiometer, which acts as the position sensor for the motor. A microcontroller provides the servo with PWM signal. Electronic components inside the servo then converts the pulse width of the signal into a position, which is fed to the potentiometer that responded by adjusting its value to the input accordingly. The electrical motor moves as it is powered by the potentiometer until the desired value in the potentiometer is reached.

2.4.2 JTAG Debugger

Standard Test Access Port and Boundary-Scan Architecture, or commonly known as Joint Test Action Group (JTAG) is developed to test printed circuit boards using boundary scan [9] where pins-out of ICs soldered on multi layered boards can be viewed to scan board for solder joint fault on the ICs.

In debugging, JTAG is used due to ability to access sub-blocks of integrated circuits on embedded systems which has limited options of other communications channel with debug capability. With JTAG, a direct connection from host-based debugger software to debug logic inside CPU is established, enabling software debugging of an embedded system to be targeted directly at the machine instruction level when needed, where assertions of debug exception retargets the processor to fetch instruction from logic registers instead of the program counter [10]. The connection is shown as in Fig. 3.

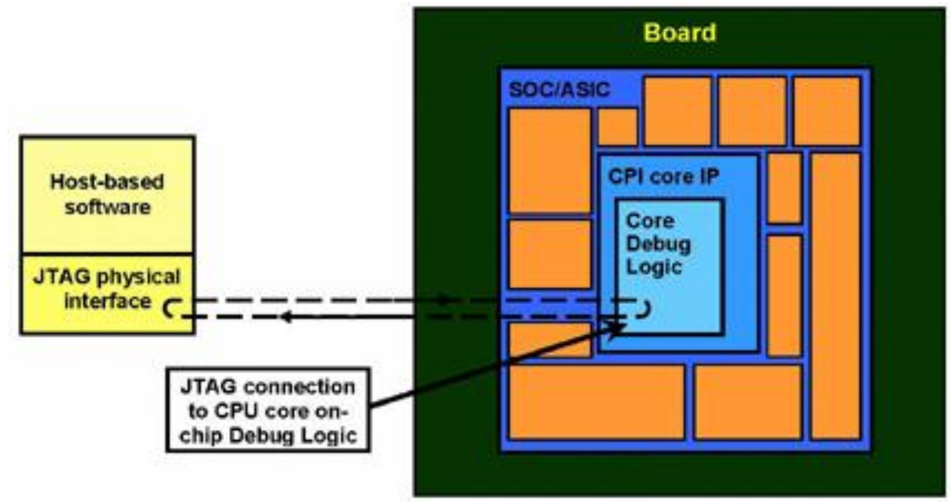


Figure 3: [10] JTAG connection for debugging

The significance of using a JTAG debugger is more obvious when involving projects with complex structures. One example is in tracing fault in occurrence of chip halting in HardFauld handler, where with a debugger, ‘breakpoints’ can be inserted into code to temporarily halt code execution until instructed to run again, in single-step, to trace the source of fault. When used with GNU tools, or IDE, or any commercial toolchain, the J-Link JTAG debugger can reset a device, start program execution and halting the program on main() for user to run, with just a click to a button. This eliminates the inconvenience of using UART or USB bootloader programmed to the device for debugging [11]

2.5 Related Literatures

No.	Title	Author & Year	Summary
1.	Design and Development of ARM Processor Based Web Server [5]	Roy, B.R., Dessai, S., and Shiva Prasad Yaday, S.G., 2009.	- A project to develop an embedded web server using ARM9 processor and µC/OS-II as RTOS. - µC/OS-II is used to monitor all the tasks of the web server. - The embedded web server is tested for its working, using a data acquisition web application hosted over a network of PC's.
2.	High Speed Data Acquisition and Processing System Design of Power Transformer [12]	Xing-tao, S., Wen-rui, Z., 2009.	- Project on gathering transformer body and power system data. - Used NXP LPC1768 for data acquisition and Ethernet to send data to PC.
3.	Design and Implementation of Software Architecture Behavioral-Based Robot Control System Using Active Object Computing Model [13]	Berry Perdana, P., Kusprasapta, M., Widyawardana, A., 2011.	- Developed a software using Active Object Computing Model which is successfully tested on an autonomous mobile robot running on NXP LPC1768 demonstrating obstacle avoidance and object following.

4.	The Design of Embedded Web Server For Remote Laboratories Microcontroller System Experiment [14]	F. Yudi, L., Harry, S., AnakAgungPutri, R., AjibSetyo, A., 2009.	- Design of embedded web server using 8-bit microcontroller AT89S52. - System consists of a computer server, experiment module microcontroller system. - Embedded web server serves as the user interface to control the lab module microcontroller via the internet.
5.	Industrial Process Parameter Control using Ethernet [15]	N. U. Chipde, V.R. Raut, 2013.	- Implementation of industrial automation using ARM processor with Ethernet controller to provide network interface capability. - Configuring the ARM board register and memory, with design consists of SPI communication, processor and Ethernet interface module to access the ENC 28J60. - Provide high performance solution compared to conventional industrial monitoring and control system.

Table 1: Related literatures with the project

CHAPTER 3: METHODOLOGY

3.1 Research Methodology

As this project is a prototype-based project, the author is adapting waterfall development and prototyping method at the same time to achieve the objective of producing a working prototype from the project. Both of this method had emphasized on breaking the project into smaller tasks with waterfall method suggests on sequential flow of project work, solving one task to another depending on criticality. Finally, the author would finalize the configuration of RTOS incorporated with configured embedded web server together with of servo motors controlling task for antenna positioning. Fig. 4 below shows project activities:

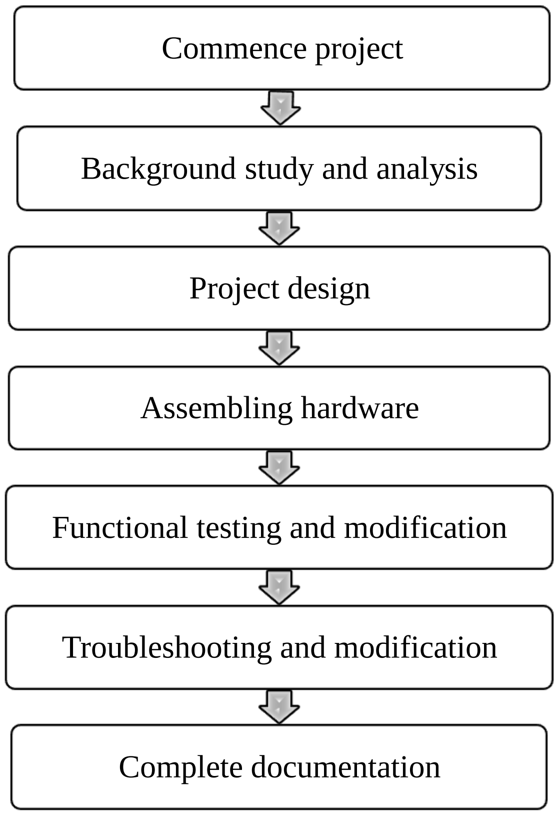

Commence project
Background study and analysis
Project design
Assembling hardware
Functional testing and modification
Troubleshooting and modification
Complete documentation

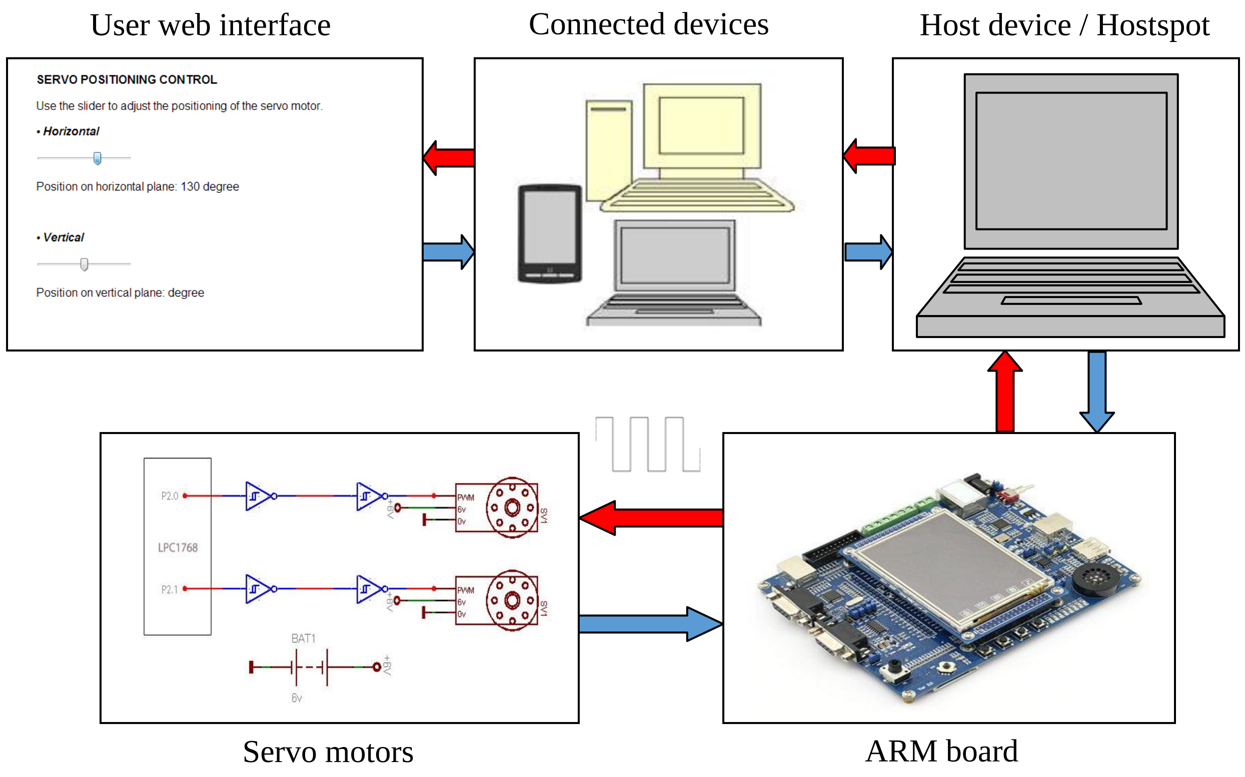
Figure 4: Waterfall methodology development activities

3.2 Project Activities

The project is started with the author reviewing related literatures on the concerned topics such as RTOS, and embedded internet network. The author had also discussed with his supervisor and project assistant on the hardware, software, project implementation strategy, and expected result from a working prototype from this project.

3.2.1 Hardware Connections and Design

A user graphical interface would be displayed from any web browsers accessing the web server. The input from the interface will be sent to the ARM board through Ethernet. The data would be processed and output in form of PWM signal is produced at two output pins named P2.0 and P2.1. Each of the output pins are assigned to two different servo motors which are responsible for horizontal and vertical positioning respectively. Fig. 5 below shows the overall hardware connections:



SERVO POSITIONING CONTROL

Use the slider to adjust the positioning of the servo motor.

- *Horizontal*

Position on horizontal plane: 130 degree

- *Vertical*

Position on vertical plane: degree

Figure 5: Hardware connections

3.2.2 *Integrated Development Environment (IDE) / Toolchains*

For the project development, the author used IAR Embedded Workbench for ARM version 6.40.2.3992. This IDE provides the author with comprehensive software encompassing source code editor, build automation tools, and debugger for project development. To create a project, go to Project > Create New Project and a window will pop up. The author then selects C++ as basic project template.

After that, the author had to setup for types of processor target, in which for this project is the LPC1768 processor. This is important as wrongly set target would render an error named Fatal Error[Pe1696] where the IDE are unable to open source files during compiling due to clashes of the loaded setup files which is intended to the LPC1768 with the configured setting. After that, the IDE is ready to be used.

In addition to IAR, the author also used KEIL µVision and Rowley CrossWorks for ARM as an alternative to IAR. The reason is that the downloaded µIP network stack and FreeRTOS kernel are configured to work on development board with different peripheral and port design (refer to **Appendix A**) although they share the same ARM core (LPC1768). So as the work over to this problem, the author had to use these toolchains to configure the µIP and FreeRTOS to work on LandTiger LPC1768 board before porting the whole program into IAR systems

3.2.3 LPC1768 Programming

The author uses C and C++ programming language for the project. In addition, the author utilizes CMSIS function or Cortex Microcontroller Software Interface Standard library for the programming part which is regarded as standardization attempt by ARM with the distribution of startup code, linker script, and low-level initialization details for used with LPC17XX series microprocessor.

There are few important initializations and declarations that have to be included in the program. They are:

- **core_cm3.c**– CMSIS Cortex-M3 Core Peripheral Access Layer Source File
- **core_cm3.h**– CMSIS Cortex-M3 Core Peripheral Access Layer Header File
- **core_cmFunc.h**– CMSIS Cortex-M Core Function Access Header File
- **core_cmInstr.h**– Cortex-M Core Instruction Access Header File
- **system_LPC17xx.c**– CMSIS Cortex-M3 Device Peripheral Access Layer Source File for the NXP LPC17xx Device Series
- **system_LPC17xx.h**– CMSIS Cortex-M3 Device Peripheral Access Layer Header File for the NXP LPC17xx Device Series
- **LPC17xx.h**–Needed for function device struct and interrupt function
- **startup_LPC17xx.s –** Startup assembly bootstrap
- **ldscript_rom_gnu.ld** – Linker script
- **main.c**– Main project program
- **Makefile –** To compile projects

To use CMSIS function, the author has to enable the function from the program options (General Options > Library Configuration tab > CMSIS > Tick Use CMSIS) or else compiling the project would result in Fatal Error[Pe1696] where the IDE are unable to open source files for CMSIS.

Before starting with the PWM setting, the author had to identify the correct CPU frequency clock setting for the microcontroller. Inside the LPC1768 is a peripheral named Phase Locked Loop (PLL) that outputs multiple frequencies from reference frequency sources available [16]. As stated earlier, LPC1768 is capable to work up to 100MHz of frequency which is the output as the result of calculations through PLL as it is fed with a reference frequency sourced by a low frequency crystal oscillator, in which in LPC1768 has a frequency of 12MHz. The overall PLL system is shown in Fig. 6.

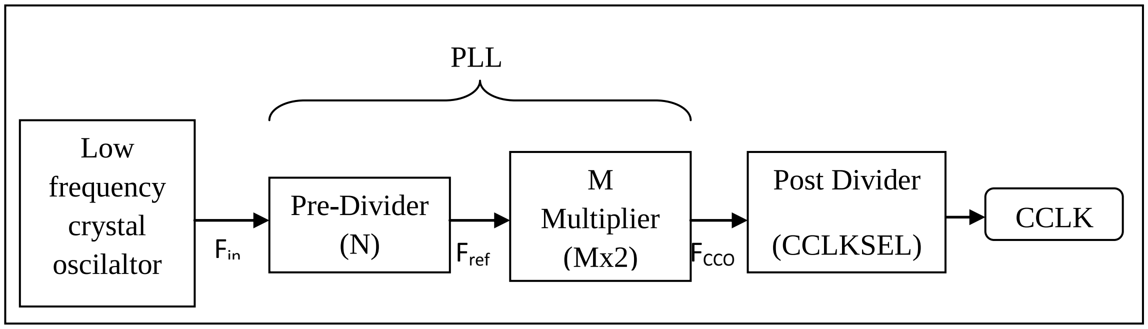


Figure 6: Phase Locked Loop overall system

The author decided to use the maximum 100MHz for processing clock (CCLK) for fast computation. To obtain this the author had to determine the current controlled oscillator frequency (F_{CCO}) which has to be in range between 275 to 550MHz. The author multiplied 100MHz with divide override value increasing from 1, until the value of F_{CCO} is in required range. For this project, the divide override is 4 as 100MHz multiplied by 4 gives 400MHz which satisfies the requirement.

Next, the author used the LPC17xx Main PLL Parameter Calculator from NXP to determine the value of N divider and M multiplier. Entering the value of Maximum CPU Frequency to 100MHz (this is the maximum frequency for LPC1768), target F_{CCO} to 400MHz, F_{IN} which is the crystal oscillator frequency to 12MHz, and divide override to 4, the author choose M value to 50 and N to 3 as these settings produce lowest frequency error.

To determine the CCLK resulting from parameters obtained, a calculation is done from Equation (1) which output 100MHz from parameters obtained:

$$CCLK = \frac{N \times (2 \times M)}{Divide\ Override} \qquad (1)$$

To apply all of the parameters into program, the value of N, M and divide override are subtracted with 1, giving values 2, 49, and 7 respectively. All of these values are then converted into hexadecimal. Inside main.c, the parameters are entered under hardware initialization where M and N are put into PLL0CFG, valued 0x00020031 (the first four digit in the right is the value of M-1 and next four digit is the value of N-1), and divide override is set in CCLKCFG, which is 0x00000003.

By determining the CCLK value, the author now can set the PWM and ADC peripheral clock (PCLK) input directly from CCLK. The author set the prescaling for the PCLK to 1MHz to obtain PWM with time base of 1µs (period the reciprocal of frequency). This is important in determining the MRx register value to obtain PWM of 20ms with varying frequency ranging from 0.5 to 2.5ms. Value of MRx is determined by dividing prescaled PCLK with desired output frequency.

For the PWM part, the author had done the following:

1. Enabling power to PWM (PCONP register)
2. Enabling peripheral clock to PWM peripheral
3. Selecting and configuring output pins for the PWM signal. There are two channel outputs that have been selected. Pull-up or pull-down resistors are disabled.
4. Setting the rising and falling edge of the signal. This function however is controlled by ADC input but with limitations such that to control pulse not to exceed the minimum and maximum duty cycle that can be handled by servo motor.

For the ADC part, the author had done the following:

1. Setting up interrupt in which it reads the value from the potentiometer and interrupt the program cycle with reading that has been obtained from the end of A/D conversion cycle.
2. Configuring the input part for the ADC. Selecting the potentiometer included on the board as input, which is of port 1 and numbered under pin number 31. No pull-up, pull-down and open drain mode are enabled.
3. Configuring the ADC channel 5 and conversion rate to 200kHz

For the Ethernet part, the author used a preconfigured µIP Ethernet library targeted for LPC1768 development board. The author had done the following [17]:

1. Include all the related .h files and libraries in main c file.
2. Altered the IP address located in FreeRTOSconfig.h file by changing the values of configIP_ADDRx. Note that the µIP is incorporated with FreeRTOS.
3. Modify the HTML code of the demo website located in webserver>httpd-fs folder. The author use HTML5 to produce slider GUI as most web browser nowadays supports HTML5. The author had to make sure that the whole website does not exceed the limitation of the library to 512Kbyte only.
4. HTML file is set to .shtml file to enable access to hardware. After modification, makefsdata.pl Perl script had to be executed for the HTML code modification to take place.
5. Add PWM function to the httpd-cgi to allow communication between input from web brower to hardware to manipulate the PWM duty cycle execution code.
6. Configured target computer network configuration. The steps are:

   1. Go to: Start > Control Panel > Network and Internet > Network Adapters

   2. Right click LAN network adapter > Properties

   3. Select Internet Protocol Version 4 (TCP/IPv4) > Properties > Select “Use the following IP-address”

   4. Insert parameters as the configured value in FreeRTOSconfig.h

The author had done configuring FreeRTOS in which all of the previously configured tasks are combined and set up to work in RTOS environment. The overall task structure flowchart is attached in **Appendix B**. Overall program would start by looking at the input interface, in our case is the TCP/IP, before processing the updated input parameters set by operator through web browser. The kernel is then set to wait until next control cycle, set in milliseconds resolution cycle, to enable periodic task execution by using FreeRTOS Application Programming Interface (API) named xTaskDelayUntil.

Another important API is xQueueRecieve used in blocking specified task execution in order to give way for other task to be executed. In this project, we can said that if the TCP/IP is not sending any updated value, then other task such as PWM can still be executed. When an input is sent, an interrupt would unblock the task. The task then would wait until all the parameters are updated. The kernel would then execute the outer loop model and update the MR1 and/or MR2 register of the PWM responsible for setting the PWM duty cycle in the interrupt service routine (ISR), before returning to the starting point.

A flowchart is designed to guide the author in the coding and the flow of the program. The overall work flowchart is attached in **Appendix C** and **Appendix D** is the flowchart for PWM configuration and test using potentiometer (ADC) input. Note that the ADC part is only temporary, only to simulate on how the HTTP input should work as an input to the PWM port.

3.2.4 LPC1768 Debugging

The LandTiger LPC1768 development board has onboard support for JTAG debugging and program download through the debugging port (CN4) for the access to the on-board J-LINK emulator (U3). For this project, the author used J-LINK which is a JTAG emulator for ARM microprocessors that has built in 20-pin JTAG connector (CN1) to communicate with PC via USB (CN4) channel.

To use the JTAG debugging and JLINK emulator, the host PC has to be installed with J-LINK for ARM Version 4.54c software to provide IAR development program with JTAG/SWD interface and JLINK emulator. To download and debug program using JLINK emulator, the author had to configure the debugger options in project program. The steps are:

1. Right click on project folder > Select Options.
2. Select Debugger > Setup tab > Driver > Select J-Link / J-Trace
3. Select Download tab > tick Verify download and Use flash loader(s)
4. Select J-Link / J-Trace under Category window > Tick Auto under SWO clock.

During debugging and compiling process, there are some error encountered such as Error[Li005]: no definition which is solved by linking latest and related library files to the project. This is usually occurs when a library function is called in header files but the library files are not linked to the project. Other error encountered are Warning[Pe177] mainly due to unused function that has been declared in the program. These errors might be nothing, but would cause the program failed to be compiled and run.

3.3 Key Milestones

To achieve the objectives of the project, there are several key milestones had been underlined to meet the project requirements accordingly. Listed below are the key milestones:

3.3.1 Background Study and Analysis

Identifying the objective of the project and recognizing the purpose of FYP2 as continuity to the work done in FYP1 beside as requirement for a degree collectively. The author had gathered related information regarding project from journals, forums, and online resources with the help of a postgraduate student assigned by the supervisor to guide the author on keeping track with relevant topic in FYP1. For FYP2, the author would spend most of part in studying the composition and configuration of RTOS and embedded web server stacks through developers' manuals.

3.3.2 Project Design

In addition to outlined procedures, hardware, software, and tools required by the project in FYP1, the author had added several more into the project design as an alternative to previously underlined design or to solve problems encountered during project work. In FYP1, the author had configured two ouput pins on LPC1768 board to produce PWM signals with 50Hz frequency, together with configured on-board potentiometer (analog to digital (ADC) input) to control the duty cycle of the PWM to control servo arm position before testing with servo during debugging, in which the author had encountered several problems and had to introduce new tools, hardware and software, to solve the problem.

Later, the ADC was replaced with a slider input from a website hosted on configured µIP network stack to work on LPC1768. To manage both of the µIP task and PWM task, the author had to configure FreeRTOS to work in preemptive mode to assign task priorities to each tasks together with interrupt calls so that each tasks would run whenever an interrupt is made as decided by the configured FreeRTOS kernel.

3.3.3 Project Implementation

Toolchains such as Keil µVision and Rowley CrossWorks for ARM are used to modify preconfigured network stack and RTOS kernel before porting into IAR Embedded Workbench. Ethernet cable is connected from LPC1768 board to a computer. Loaded code will configure the LPC1768 over the network to work as embedded web server, but the user had to manually configure the IP address, subnet mask, and default gateway of the local area connection in target device to enable communication with the embedded web server.

Connection between two devices are then confirmed by using command prompt (cmd) where LPC1768 will be pinged through the cmd to test it for network connection. Using web browser, the IP configured in network stack is entered to request for server connection to the LPC1768 [5], where a graphical user interface, configured using HTML5, would be loaded on the web browser as the input to control servo motor on the PWM output part at the LPC1768 output pins.

Any progress and problem on the work done shall be reported to project assistant to keep track on job done beside underline the possible solutions to problems encountered.

3.3.4 Documentation and Reporting

Every related manuals and references are compiled in a folder. Detailed documentation is updated weekly following project progress. Problem encountered during progress should be underlined for future references. Findings and project work shall be analyzed and discussed thoroughly to nurture ideas on how to improve the project implementation. Lesson learned should be highlighted for future notations. All of the project work such as the source code are saved into a CD-ROM and submitted to the supervisor.

3.4 Gantt Chart

No	Detail / Week	1	2	3	4	5	6	7	Mid-Semester Break	8	9	10	11	12	13	14	15
1	Project Work Continues	🟦	🟦	🟦	🟦	🟦	🟦	🟦									
2	Submission of Progress Report							●									
3	Project Work Continues									🟦	🟦	🟦	🟦	🟦			
4	Pre-SEDEX											●					
5	Submission of Draft Report												●				
6	Submission of Dissertation (soft bound)													●			
7	Submission of Technical Paper													●			
8	Oral Presentation														●		
9	Submission of Project Dissertation (hard bound)																●

Table 2: Project Gantt chart

● Suggested milestone 🟦 Process

3.5 Tools

3.5.1 Software

No.	Name	Description
1.	Microsoft Office Professional Plus 2007	• Word for documentation, e.g; writing report • Excel for Gantt chart
2.	ActivePerl 5.16.3	To execute script when changes are made to HTML code
3.	Strawberry Perl 5.16.3	
4.	IAR Embedded Workbench	• Proprietary tools for compiling, programming and porting program into ARM board [18] • As tools to learn on embedded system programming based on example projects • To monitor and debug program
5.	KEIL µVision v4.03q RealView MDK-ARM	
6.	Rowley CrossWorks for ARM v2.3.2	

Table 3: List of software for the project

3.5.2 Hardware

No.	Name	Description
1.	LPC1768 ARM Board	Receive input from input unit, executes the stored user program, and send out appropriate output command to control devices
2.	Point-to-pint crossover Ethernet cable	To connect ARM board to the network
3.	Power supplies	• 240VAC adapter to supply 5VDC power to the board • 6VDC power supply for servo motor (4 AA batteries)
4.	74HC14N Hex inverting Schmitt trigger IC	To reproduce PWM signal to a stable 6V square wave signal for servo input
5.	Servo motor	Final element in this control project
6.	J-TAG Debugger with USB cable	To load and debug hardware. Also as an alternative power source to the board

Table 4: List of hardware for the project

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Pulse Width Modulation Output Signal

The PWM signals, named PWM1.1 and PWM1.2 are configured to be channeled to output pin P2.0 and P2.1 respectively. For the PWM testing part, the ARM processor would process A/D input, named AD0.5 from potentiometer onboard LPC1768 development board labeled as P1.23 to control the duty cycle of the PWM. The PWM is set for single edge application where one point of the match register, used to set the duty cycle of the signal, is fixed while the other register is set to trigger the signal.

The frequency of the PWM is determined at the MR0 match register where it is set to 50Hz for servo application. On the other hand match register MR1 is controlled by potentiometer to produce 0.528ms (for servo position at 0˚) to 2.5ms (for servo position at 180˚) of duty cycle PWM which is the operating range of the servo (0˚ to 180˚).

The PWM signal is verified working by channeling the PWM to two LED output onboard that is by manipulating the brightness of the LED using PWM signal. By using oscilloscope with w vertical setting is set to 2V/div and 5ms/div for horizontal setting, we could see PWM signal changing as we turn the on board potentiometer as shown in Fig. 7.

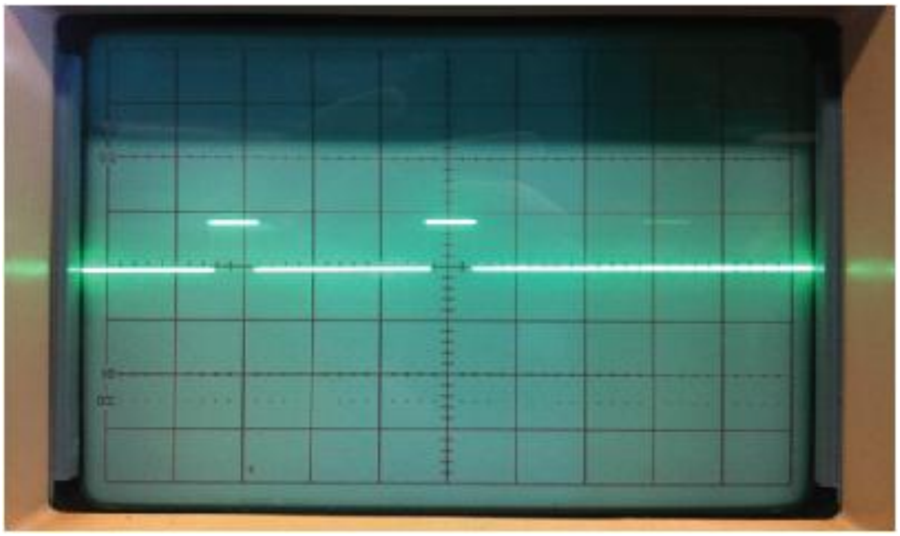


Figure 7: PWM output displayed by the oscilloscope

The ADC input was replaced with web interface slider input to provide the input to the PWM section after initial testing had been done. The author used the “measure” function from the oscilloscope to measure the frequency. In initialization state, the PWM is set to produce pulse width of 1.5ms with PWM period of 20ms, as instructed in servo manual. The 1.5ms pulse width is translated to 90° position, which is a neutral state as the range for servo rotation is 0°-180°, while the period 20ms is translated to 50Hz frequency of the signal. The result obtained was as shown in Fig. 8 below where the pulse width and signal frequency are confirmed to work in correct order:

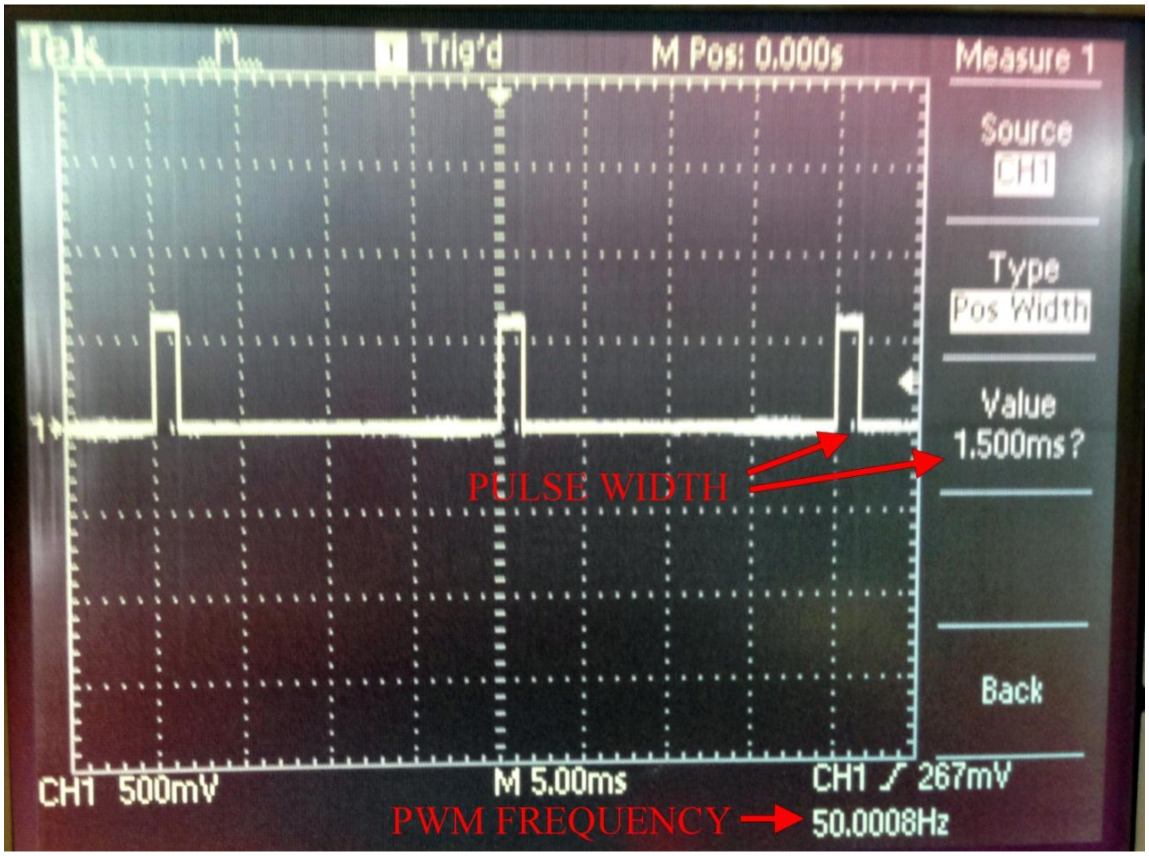


Figure 8: PWM signal after initialization with web interface as input

The pulse width and PWM period was the main concern since over-rating of pulse width from specified requirement in servo manual would cause damages to servo motor on top of undesired rotation and servo jittering as the result of wrong setting.

4.2 Servo Motor Response

Initial sign that shows whether PWM signal is working is through LED LD11 and LD10 on the board since both LEDs shares the output with the PWM output pin. The author had tested the PWM to the servo motors and the servos responded as expected to the input signal as the author changed the slider in web interface to desired degree of rotation.

According to the servo manual, 0°-180° of rotation is translated from pulse width valued between 0.582ms to 2.5ms. Therefore, the author tested the servo with three point pulse width test, 0.582ms, 1.5ms, and 2.5ms to determine the correctness of servo rotation and had results agreed with the theory from the user manual as shown in Fig. 9 below:

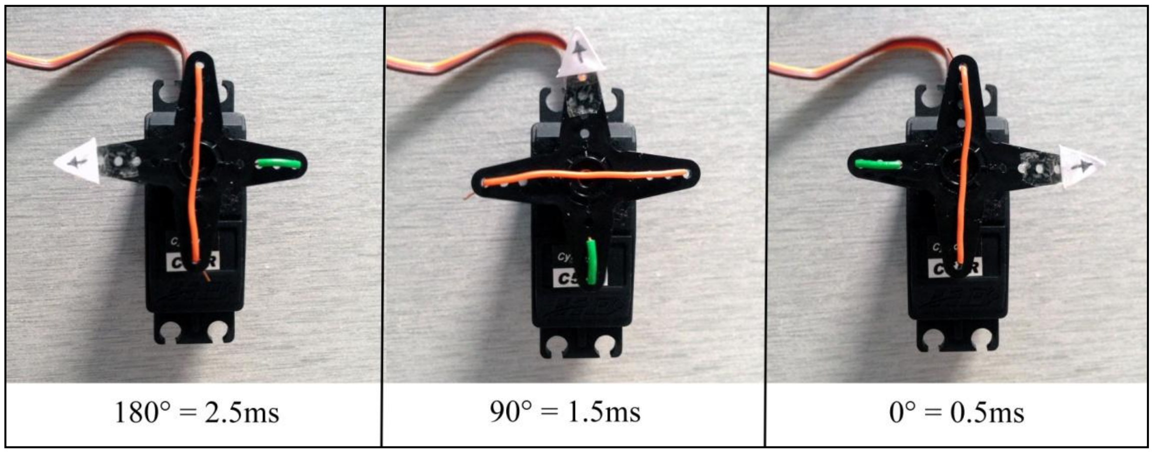
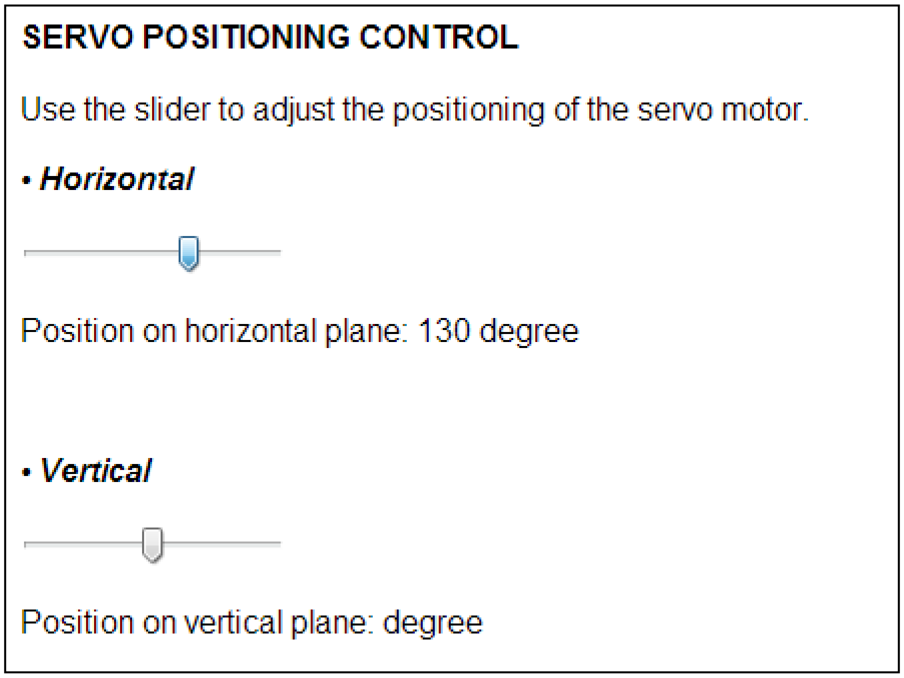


Figure 9: Degree of rotation of servo in three different signals with different pulse width

4.3 Web User Interface

User interface can be accessed by entering IP address: http://192.168.0.201/pwm.shtml. The IP address set for this project corresponds to the defined IP address set in FreeRTOSConfig.h file under definition "configIP_ADDR0 - configIP_ADDR3".


SERVO POSITIONING CONTROL

Use the slider to adjust the positioning of the servo motor.

- *Horizontal*

Position on horizontal plane: 130 degree

- *Vertical*

Position on vertical plane: degree

Figure 10: Graphical web interface for servo control displayed on web browser

The web interface shown in Fig. 10 would not be able to be accessed if the IP address entered in browsers does not agree with the address set in program code. Other than that, user has to ensure that parameters of the IPv4 for host network adapters are correct. For this project, the parameters used are as below (note that IP-address is +1 from address to be used in browser):

a. IP-address = 192.168.0.202
b. Subnet mask = 255.255.255.0
c. Default gateway = 192.168.0.254

4.4 Constraints and Problems Encountered

Designing and developing a system from scratch is very challenging especially when working individually. The only experience that the author had related with his project was developing a blinking LED in structured programming course using C and developing a basketball scoreboard in Microprocessor course, which is almost different with the ARM environment. The author had to learn about ARM programming, the ARM board, RTOS, and embedded web server from the beginning. However, the author had managed to learn by referring to example projects and reading related literature and online forums as references.

The first problem is to find suitable software for developing and debugging the program. There are several readily available software for ARM platform such as Kiel MDK-ARM. However for this project, the author needs IAR Embedded Workbench which has different environment to Kiel especially in its Library and Filing system. The author also had to upgrade his IAR software from 6.30 to version 6.40 to be able to load example projects which use upgraded library files.

However, not all example projects could be run properly where some of them require the author to modify the code. This is because some of the example codes are programmed for board build by specific manufacturer, for instance the IAR LPC1768-SK development board by IAR and Keil MCB1768 Eval Board 72R6098 by NXP. Although these board shared the same LPC1768 system, the pin and on-board configurations are somewhat differs to each other as chip manufacturers decide on the pin configuration for their product board as shown in **Appendix A**. The author had to refer to LPC17xx manual to modify the code, in which by doing that, the author had not only able to understand the example project, but also to learn on how to configure the board through programming.

Another problem faced by the author is in understanding numbered error messages during compiling and debugging process. With the assistance of a postgraduate student, the author managed to understand the proper initialization and setting to his projects as discussed in Chapter 4.

There are also some hardware issues during project execution. One of them is that all output pins on LandTiger board giving out sinusoidal noise even when there is no program are running as shown in Fig. 11. This had resulted in PWM signal disturbed by noise signal thus the servos are unable to run. Initially the author had changed the CMSIS library to mbed library and using example code from mbed LPC1768 to check if the coding was the culprit for the noise.

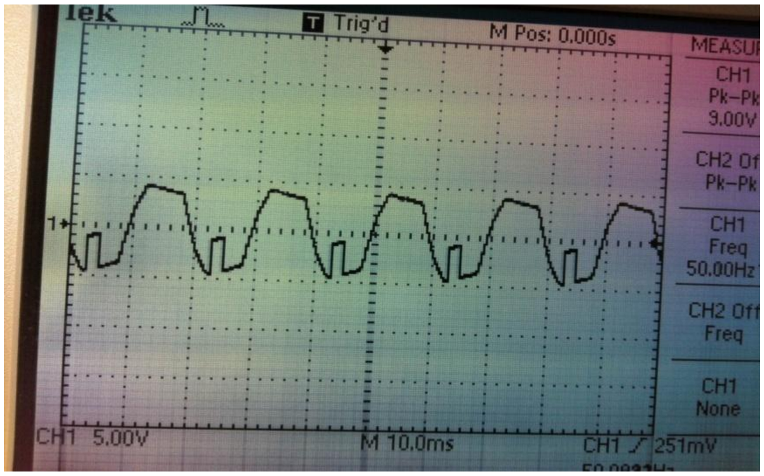


Figure 11: PWM signal (square wave pulse) and noise (sinusoidal) signal

Later it was discovered that the 240VAC power adapter is giving out more than 5VDC output. It was stated in LandTiger manual that the input voltage must not exceed 5V ±5% deviation [19]. The problem is solved by using USB power supply to the board (input source can be changed by adjusting the position of power supply jumper JP3 on the board). Another benefit of this solution is that the author had also encountered faulty J-Link debugger during debugging. The same USB supply can also be used to load program into the board by removing JTAG_SEL (JP4) jumper on the board to enable onboard J-Link emulator.

The next problem encountered by the author was the servos are not moving even though correct PWM signal had been obtained. From the oscilloscope, the voltage amplitude for the PWM was only at 3.3V. The author then uses 74HC14 Hex inverting Schmitt trigger to reproduce the PWM signal to perfect square wave with 6V amplitude as shown in Fig. 12. The 6VDC is the result of using 6V battery input shared with servos as power source. With this, the servos are able to run.

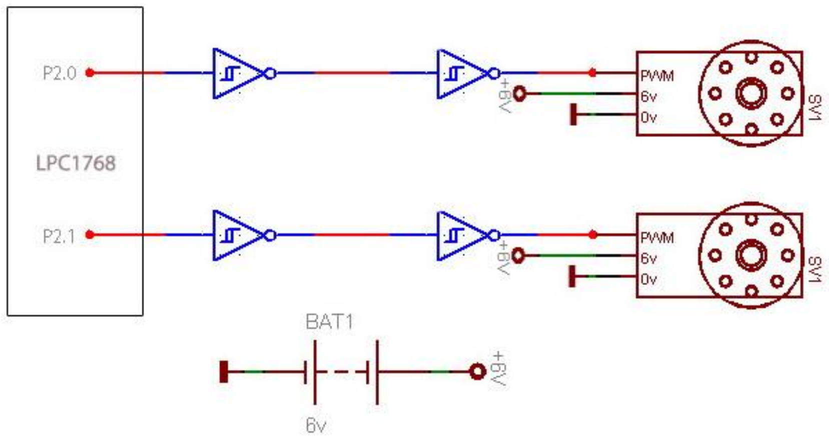


Figure 12: Connections from the PWM pin to servos through 74HC14

On top of that, due to complexity of coding a TCP/IP and RTOS stack for a microcontroller, the author resorted on using a pre-configured open source stack programmed specifically for embedded application namely the µIP and FreeRTOS respectively. By doing this, the author would be more focused on porting both stacks to work in microcontroller used through IAR toolchain.

The µIP network stack requires a driver for any Ethernet interface being used. The well documented µIP provides better solution for the author to concentrate on application level coding specifically on controlling the PWM duty cycle through web server for control of the servo motors. With examples of applications such as on how application level protocol like HTTP is coded, the author are able to learn on how to produce similar results based on established examples.

CHAPTER 5: RECOMMENDATION AND CONCLUSION

5.1 Recommendations

For future improvement of the project, the author would like to propose changes in web interface in terms of user friendliness with improved adaptations regardless of web browser used to access the interface. Current interface is coded using HTML5 which may be unsupported to old browsers. This can be solved by using elements or styles combining HTML, CSS, and JavaScript to produce interface that is supported by wide range of browser.

On top of that, the author suggests on direct submission of input from slider to the servo whenever the slider is moved. In this project, the author uses ‘submit’ option in HTML form, where data from slider input are only submitted to the processor after mouse click is released from the slider (as the author use JavaScript’s ‘onmouseup’ and ‘ontouchup’ event). An upgrade in this section would provide a more smooth operation of the program.

In terms of peripheral use, the author recommends upgrading the RC servo to servos that can rotates continuously or provide 360° rotation. Example of this type of servo is the one used in model yacht sail winches such as the G15 cube servo by Cytron. Small modification is needed in program code to increase pulse width of the output signal to enable rotation more than 180° compared to normal RC servo. This would provide the project with more capable output.

In addition, the author would like to purpose an expansion to this project that is to produce a working prototype of Ethernet controlled robot. The robot actions and movement would be controlled remotely through web interface or smartphone apps while maintaining the essence of the project that is controlling and monitoring through the means of embedded HTTP server on ARM board.

For use in mission critical application or where security is the main concern, suggestion of adding password into website by using HTML form is highly recommended. On top of that, security can be added by replacing FreeRTOS with SafeRTOS which is mainly use in industrial, medical, aerospace, and nuclear sector with added cost compared to FreeRTOS.

5.2 Conclusion

The author had successfully produce a working prototype showing peripheral control and monitoring through embedded web server sourced from ARM board, in which in this project, a web interface enables user to control servo motor by adjusting input sliders which also shows the degree of servo rotation. This project had also successfully simulate the usage of RTOS in managing different tasks by the microcontroller which can be confirmed by using FreeRTOS web example that shows running tasks besides detecting error in tasks execution. Overall, all of the objectives in this project had been successfully achieved with recommendations underlined by the author for future improvements.

REFERENCES

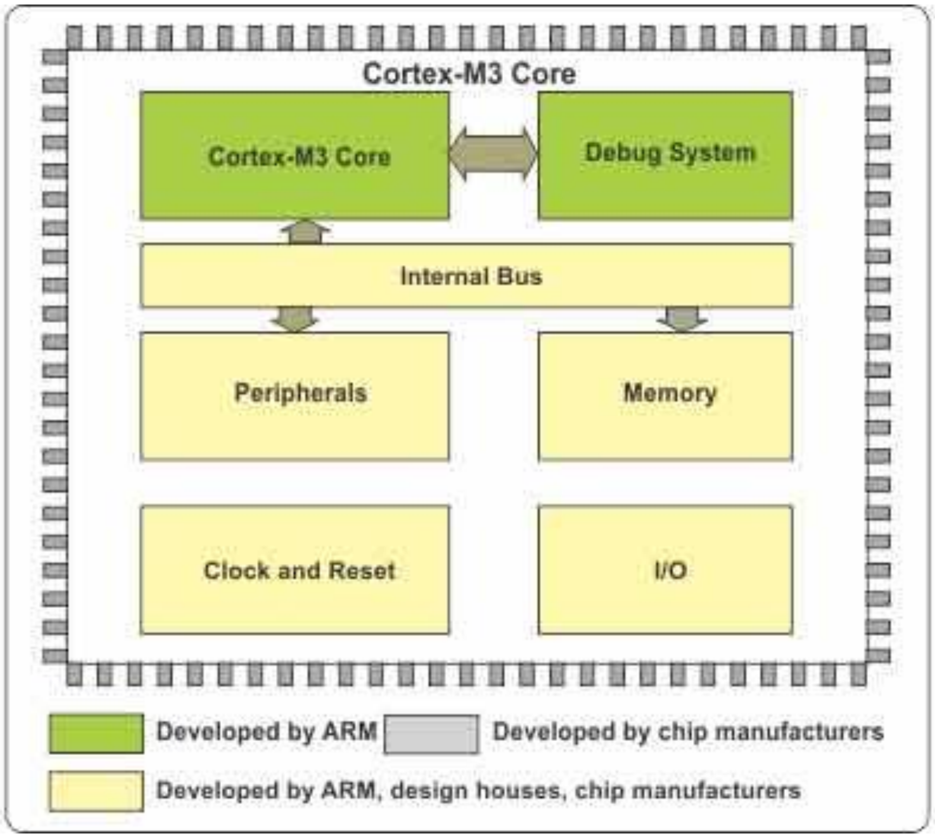
1. Booth, J.A, and Cozart, S., 2010. *Access Your Embedded Controller with Ease through a Web Server*. Texas Instruments Corporation, Dallas, Texas, USA.
2. Arora, H. (2012, February 6). *What is RTOS? – Real Time Operating Systems Basics.* [online]. Retrieved from: http://www.thegeekstuff.com/2012/02/rtos-basics/
3. Rakesh, (2012, April 12). *What is Real Time Operating System (RTOS)- How it works?*[online] Available: http://www.circuitstoday.com/what-is-real-time-operating-system-rtos
4. Prado, Sergio. (2012, May 27). *Mbed – Integrating FreeRTOS on a Cortex-M*, [online] Available: http://sergioprado.org/mbed-integrando-o-freertos-em-um-cortex-m3/
5. Roy, B.R., Dessai, S., and Shiva Prasad Yaday, S.G., 2009, "Design and Development of ARM Processor Based Web Server," *International Journal of Recent Trends in Engineering,*Vol. 1, No. 4, May 2009.
6. Dunkels, A. "The µIP Embedded TCP/IP Stack: The µIP 1.0 Reference Manual", Swedish Institue of Computer Science, June 2006.
7. ARM University Relations. *ARM Cortex-M3 Introduction* (2012, December). Retrieved from: http://www.arm.com/support/university/
8. NXP B.V. *UM10360: LPC17xx User Manual* (2010, August) Document No. UM10360, Rev. 2. Retrieved from: http://www.nxp.com/
9. Wikipedia contributors. (2013, August). Joint Test Action Group. [Online]. Available: http://en.wikipedia.org/wiki/Joint_Test_Action_Group
10. Pittroff, L. "Tutorial: The Role of JTAG in system debug & test throughout the embedded system development lifecycle," [online] 22 October 2008, http://www.embedded.com/design/prototyping-and-development/4008137/2/Tutorial-The-Role-of-JTAG-in-system-debug--test-throughout-the-embedded-system-development-lifecycle (Accessed 16 August 2013).

11. New Product – SEGGER J-Link EDU – JTAG/SWD Debugger, adafruit industries blog, [online] 2013, http://www.adafruit.com/blog/2013/05/16/new-product-segger-j-link-edu-jtagswd-debugger/ (Accessed: 17 August 2013).
12. Xing-tao, S., Wen-rui, Z., 2009, “High Speed Data Acquisition and Processing System Design of Power Transformer” Sch. of Electr. & Autom. Eng., Tianjin Polytech. Univ., Tianjin, China. 978-1-4577-0860-2/11/$26.00, 2011.
13. Putra, B. P., Mutijarsa, K., & Adiprawita, W. (2011, July). Design and implementation of software architecture behavioral-based robot control system using Active Object Computing Model. In *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on* (pp. 1-6). IEEE.
14. Limpraptono, F. Y., Sudibyo, H., Ratna, A. A. P., & Arifin, A. S. (2011, November). The design of embedded web server for remote laboratories microcontroller system experiment. In *TENCON 2011-2011 IEEE Region 10 Conference* (pp. 1198-1202). IEEE.
15. N. U. Chipde; V. R. Raut, 2013, “Industrial Process Parameter Control using Ethernet,” Dept. Elec. Telecommunication, P.R.M.I.T, Badnera, Amravati, Maharashtra, India. *International Journal of Science and Research, Vol.2, No. 5,* May 2013.
16. Sagar. (2010, November 25). *Using the PLL on LPC17xx* [online] Available: http://gvworks.blogspot.com/2010/11/using-pll-on-lpc17xx.html
17. *Getting Started with the Simplecortex*, BRC-Electronics, March, 19. 2012.
18. EmbeddedCraft. *ARM Microprocessor Basics: Introduction to ARM Processor*(2012, December). Retrieved from: http://www.embeddedcraft.org/
19. *Landtiger V2.0 LPC17XX Development Board: User Manual*, version V1.1, PowerMCU, 2012.

APPENDICES

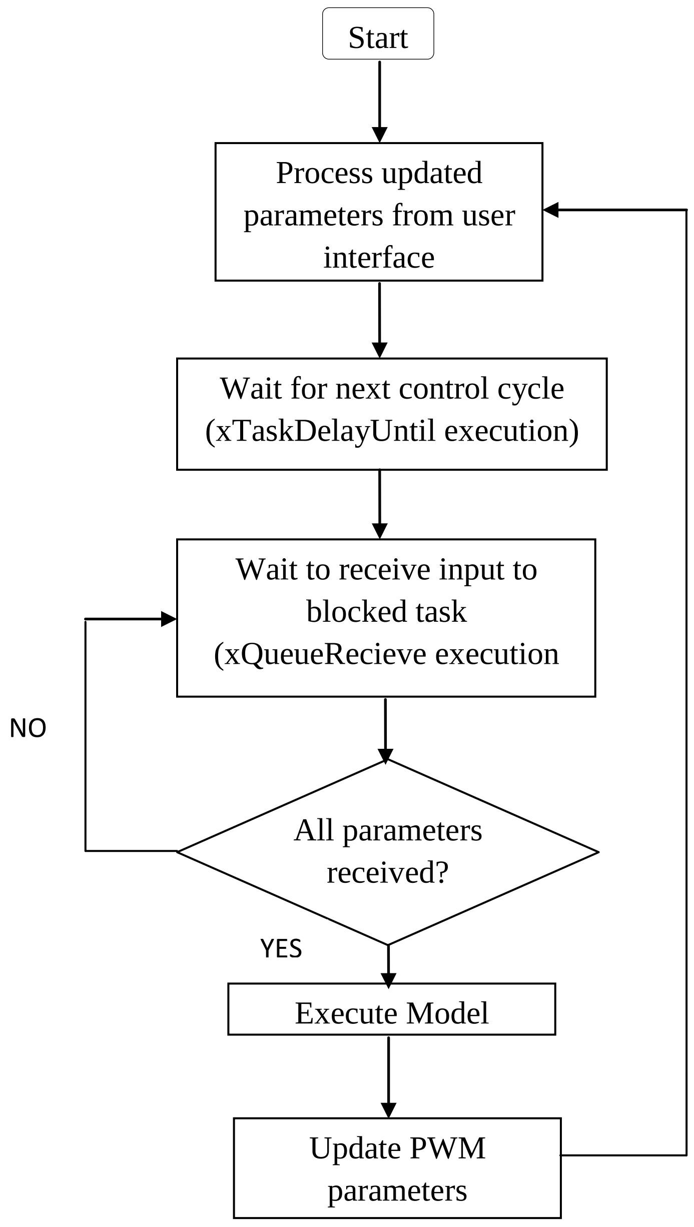
APPENDIX A

OVERALL STRUCTURE OF ARM CHIP



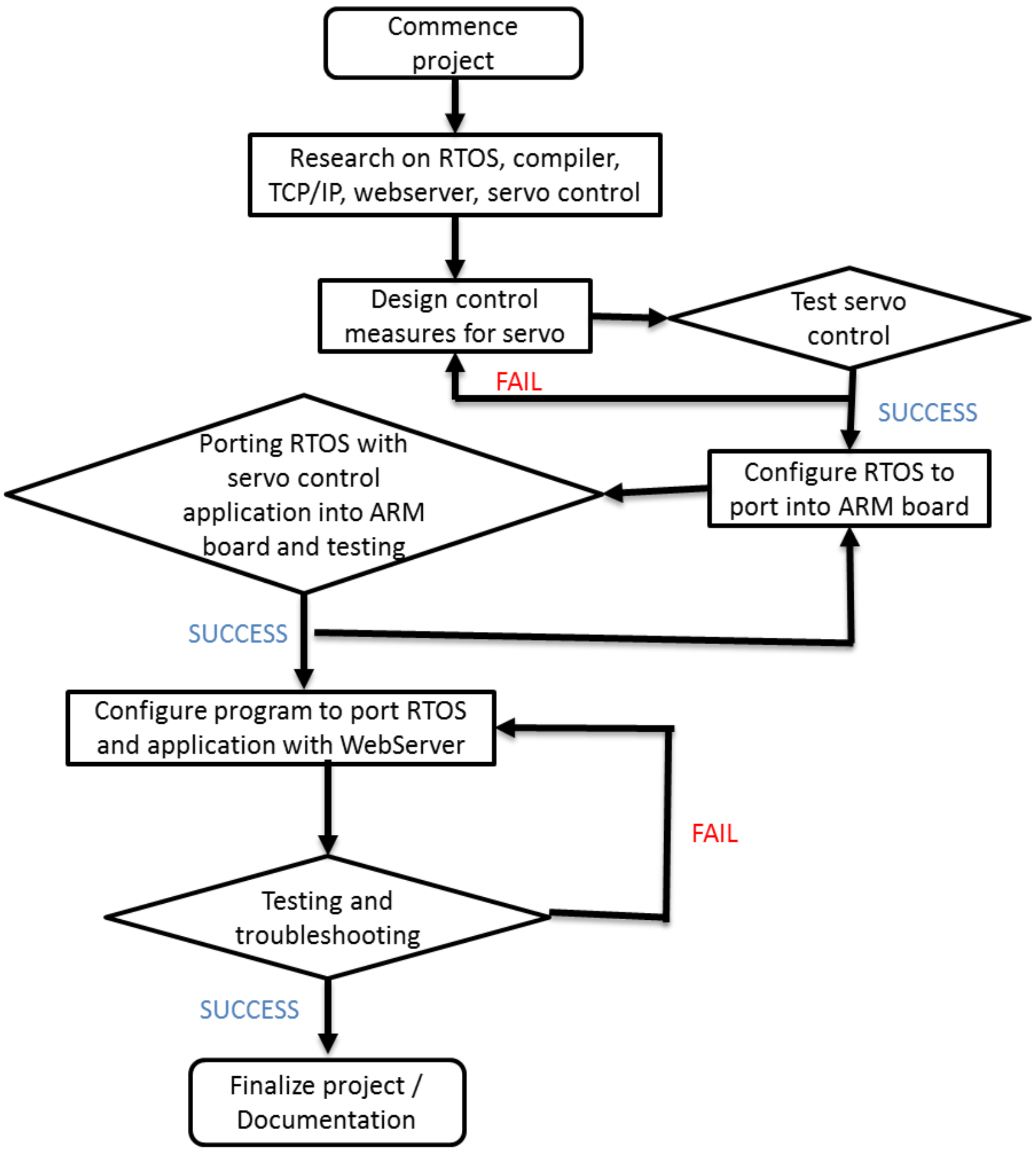
APPENDIX B

OVERALL PROGRAM EXECUTION IN FREERTOS


Start
Process updated parameters from user interface
Wait for next control cycle (xTaskDelayUntil execution)
Wait to receive input to blocked task (xQueueRecieve execution
NO
All parameters received?
YES
Execute Model
Update PWM parameters

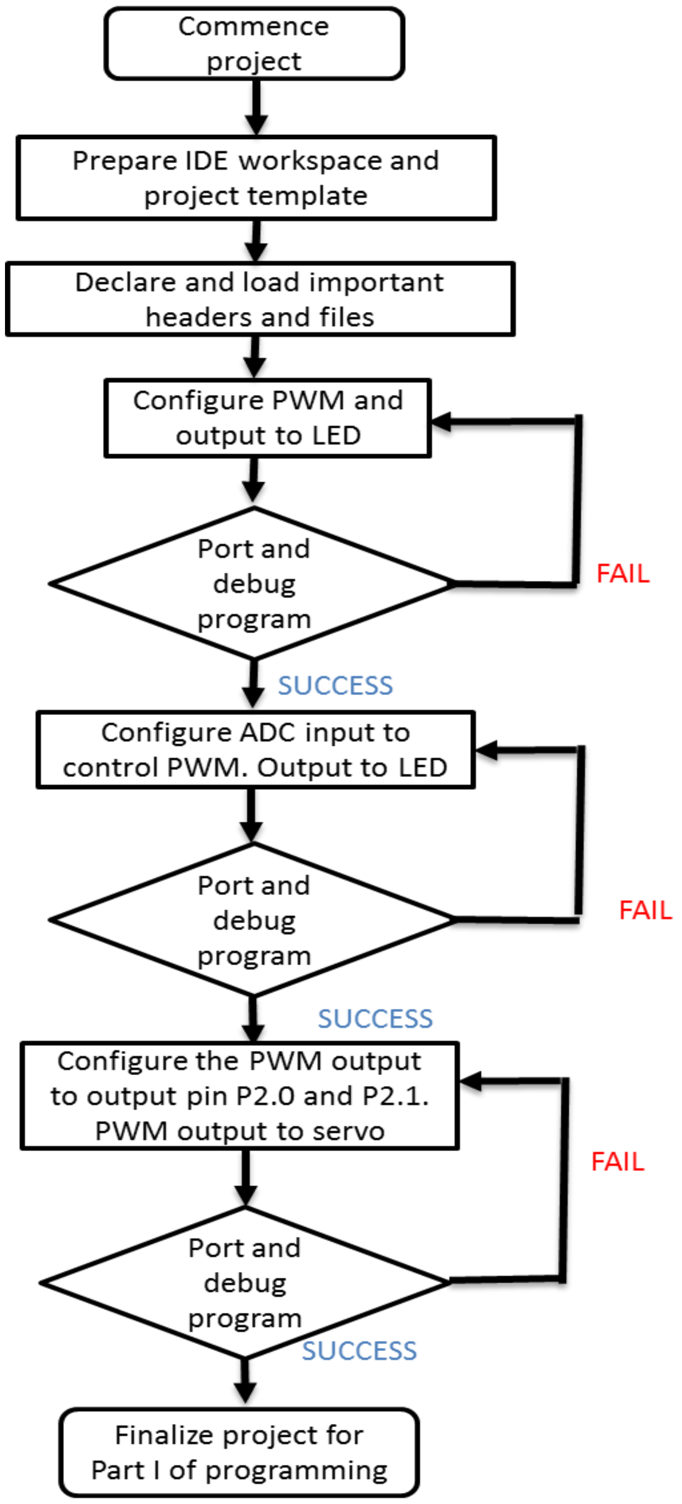
APPENDIX C

OVERALL PROJECT FLOWCHART


Commence project
Research on RTOS, compiler, TCP/IP, webserver, servo control
Design control measures for servo
Test servo control
FAIL
SUCCESS
Configure RTOS to port into ARM board
Porting RTOS with servo control application into ARM board and testing
SUCCESS
Configure program to port RTOS and application with WebServer
FAIL
Testing and troubleshooting
SUCCESS
Finalize project / Documentation

APPENDIX D

PWM PROGRAMMING AND TEST FLOWCHART


Commence project
Prepare IDE workspace and project template
Declare and load important headers and files
Configure PWM and output to LED
Port and debug program
FAIL
SUCCESS
Configure ADC input to control PWM. Output to LED
Port and debug program
FAIL
SUCCESS
Configure the PWM output to output pin P2.0 and P2.1. PWM output to servo
FAIL
Port and debug program
SUCCESS
Finalize project for Part I of programming