



UNIVERSITAS INDONESIA

INTERNET INTERFACE FOR MICROCONTROLLER

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik

MOHAMAD BAYU INDRA NUGRAHA

0405830067

DEPARTEMEN TEKNIK ELEKTRO PROGRAM INTERNASIONAL
FAKULTAS TEKNIK UNIVERSITAS INDONESIA

DEPOK

JANUARI 2010

Halaman Pernyataan Orisinalitas

**Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber yang dikutip maupun dirujuk
telah saya nyatakan benar**

Nama : Mohamad Bayu Indra Nugraha
NPM : 0405830067
Tanda Tangan :

Tanggal : 28 Januari 2010

Halaman Pengesahan

Skripsi ini diajukan oleh

Nama : Mohamad Bayu Indra Nugraha
NPM : 0405830067
Program Studi : Teknik Elektro Internasional
Judul Skripsi : Internet Interface for Microcontroller

Telah berhasil dipertahankan dihadapan dewan penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada program studi Teknik Elektro Internasional, Fakultas Teknik Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : John Edward

Penguji : Dr. Abdul Muis ST, M.Eng (.....)

Penguji : Muhammad Salman ST, MIT (.....)

Penguji : Prof. Dr. Ir. Nji Raden Poespawati M.T. (.....)

Statement of Authorship

The work contained in this project report has not been previously submitted for a degree or diploma at any other tertiary educational institution. To the best of my knowledge and belief, the project report contains no material previously published or written by another person except where due reference is made.

Signed

Date

Acknowledgments

I would like to extend my appreciation to everybody who has assisted me with this final year project throughout the course of the year. I am deeply indebted to my supervisor, Dr. John Edward for his continual guidance, stimulating suggestion, encouragement and motivation throughout the entire research period.

Special thanks also go to my family and friends for invaluable support to the completion of this project as well as Khawm Hung who looked closely at the final version of the project report for English style and grammar, correcting both and offering suggestion of improvement.

Especially, I would like to give my special thanks to my very best friend, Melly Indriasari, whose patient love enabled me to complete this work.

Abstract

At present, network is becoming the hot point for the investigation of embedded system. Considering the growth of data communication, connection between embedded system platforms and the internet interfaces has been an important development direction and indispensable functions for the embedded system in the future and it becomes an important role if the embedded platforms can be accessible and monitored whenever and wherever we need.

By implementing TCP/IP uIP-stack open source properties and correlative system interfaces architecture, some internet protocol application such as web server, ICMP and telnet server, can be integrated into the embedded systems. This paper describes how the combination between Real Time Operating System and Embedded Web Server Application can be established in ATMEL AT91SAM7X platform by sending multiple packet data and processed stably in I/O hardware architectures. At last, some real world simulations are applied in order to test system design performances and reliability.

Table of Contents

Statement of Authorship	1
Acknowledgments.....	2
Table of Figures	6
List of Abbreviation.....	8
Supplementary Material.....	9
Chapter 1- Introduction.....	10
1.1 Project Background.....	10
1.2 Project Aim, Objectives and Plan of Development	12
Chapter 2 - Hardware Characteristic.....	13
2.1 Atmel's AT91SAM7X256.....	13
2.2 Ethernet MAC.....	16
2.3 DM9161A – 10/100 Mbps Fast Ethernet Physical Layer Single Chip Transceiver	17
2.4 Parallel Input/Output Controller	24
Chapter 3 - Communication Protocol	27
3.1 Protocol Hierarchies.....	27
3.2 Service Primitive.....	28
3.3 TCP/IP Reference Model.....	29
3.4 TCP Protocol.....	31
Chapter 4 – Real Time Operating System	33
4.1 RTOS Concept.....	33
4.2 The differences between Task and Co-routines.....	36
4.3 FreeRTOS Open-Source Application Demonstration.....	39
Chapter 5 – TCP/IP Stack.....	42
5.1 Main Control Loop	43
5.2 Architecture Specific Functions.....	43
5.3 The uIP raw API	45
5.4 uIP Simple Application.....	46
Chapter 6 – Algorithm and Design Implementation.....	49

6.1	General Program Loop.....	50
6.2	PIO Algorithm	54
Chapter 7 – Software Demonstration.....		59
7.1	Possible Network Diagrams.....	59
7.2	ICMP and ARP	61
7.3	Web Server.....	62
Summaries and Conclusion.....		65
Bibliography		67
Appendix		68

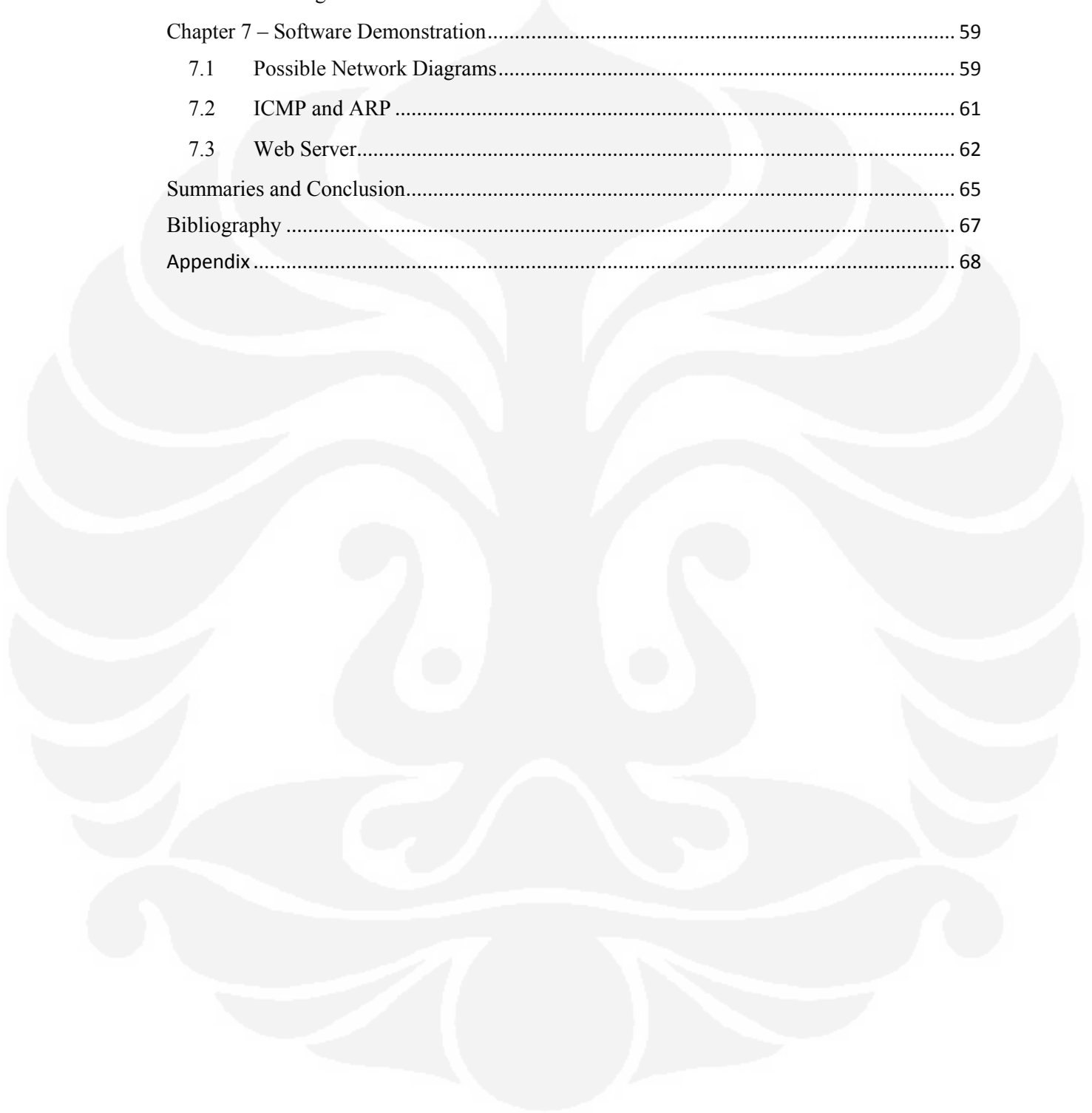


Table of Figures

Figure 1 - AT91SAM7X-EK Hardware.....	13
Figure 2 - AT91SAM7X256 Block Diagram	15
Figure 3 - EMAC Block Diagram.....	16
Figure 4 - DM9161A Chip General Functional Description	18
Figure 5 - DM9161A Schematic.....	18
Figure 6 - Specific Functional Description	19
Figure 7 - NRZ to NRZI Encoding Example.....	21
Figure 8 - MLT-3 Converter	21
Figure 9 - Manchester Encoding Example.....	23
Figure 10 - PIO Schematic.....	25
Figure 11 - Example Information Flow Supporting Virtual Communication.....	27
Figure 12 - Packet sent in a simple client-server interaction on a network	28
Figure 13 - the difference between OSI and TCP/IP Reference Model.....	29
Figure 14 - Protocol and Networks in the TCP/IP model.....	30
Figure 15 - Conventional VS Multitasking.....	34
Figure 16 - Scheduling Example.....	34
Figure 17 - Task States.....	36
Figure 18 - Co-Routine States.....	38
Figure 19 - uIP basic ICMP Demo.....	46
Figure 20 - uIP Basic Telnet Server Demo	47
Figure 21 - uIP Basic Web Server Demo.....	48
Figure 22 - General Program Loop.....	50
Figure 23 - Reading Joystick Algorithm.....	54
Figure 24 - Algorithm to Set LED Status	55
Figure 25 - Algorithm to toggle the LED	57
Figure 26 - Web Address	58
Figure 27 - Host to Host Network.....	59
Figure 28 - Local Area Network	60
Figure 29 - Wide Area Network	60
Figure 30 - ICMP and ARP Result	61
Figure 31 - Web Server - RTOS Page Stats.....	62
Figure 32 - Web Server - TCP Stats	63
Figure 33 - Web Server - Connection Page	64
Figure 34 - Web Server - IO Page.....	64

List of Table

Table 1 - Busses Function in MII Interface	20
Table 2 - PIO Register Mapping	26
Table 3 - FreeRTOS Function.....	41
Table 4 - uIP Interface Function	46

List of Abbreviation

API	-	Application Programming Interface
ARP	-	Address Resolution Protocol
BSD	-	Berkerley Software Distribution
DMA	-	Direct Memory Access
FTP	-	File Transfer Protocol
ICMP	-	Internet Control Message Protocol
IP	-	Internet Protocol
LED	-	Light Emitting Diode
GUI	-	Good User Interface
MAC Layer	-	Media Access Control Layer
MDIO	-	Management Data Input Output
MII	-	Media Independent Interface
PHY Layer	-	Physical Layer
PIO	-	Parallel Input Output
RMII	-	Reduced Media Independent Interface
TCP	-	Transmission Control Protocol
UDP	-	User Datagram Protocol
UTP	-	Unshielded Twisted Pair

Supplementary Material

1. Abstract
2. Poster
3. Presentation Slide
4. Final Year Project Code Files
5. Several necessary E-books and Reference Manuals
6. Technical Documentation

Chapter 1- Introduction

1.1 Project Background

As the World Wide Web (or Web) continues to evolve, it is clear that its underlying technologies are useful for much more than just browsing the Web. Web browsers have become the standard user interface for a variety of applications because Web browsers can provide a GUI interface to various client/server applications without having to implement a separate client.

General Web server, which were developed for general purpose computers such as NT servers or Unix workstations, typically require megabytes of memory, a fast processor, a pre-emptive multitasking operating system, and other resources. A web server can be embedded in a device to provide remote access to the device from a Web browser if the resource requirements of the Web server are reduced. The result typically a portable set of code that can run on embedded system with limited resources. [1]

Embedded Web Server are used to convey the state information of embedded systems, such as a systems working statistic, operation result and transfer user commands from a Web browser to an embedded system. The state information is extracted from an embedded system application and the control command is implemented through the embedded system application.

Atmel's AT91SAM7X256 ARM7 Based is the hardware that student used to implement embedded Web Server because it contains a large set of peripherals, including an 802.3 Ethernet MAC. So, by combining the ARM processor with on-chip Flash and SRAM, and a wide range of peripheral function, including USART, SPI, CAN Controller on it, it become cost-effective solution to many embedded control application in real world requiring communication over internet, for example, CAN wired and Zigbee wireless network. [2]

In order to establish communication between the hardware and each other across a network, we need a protocol suitable between those as a convention or standard that enables the connection. TCP/IP protocol suite has become a global standard protocol communication used for web page transfers, e-mail transmission, file transfer, and peer-to-peer networking over Internet. Traditional TCP/IP implementations have required too much resources of code size and memory usage for 8 or 16-bit systems. To solve this problem, student used open-source uIP implementation that is designed to have only absolute minimal set of features needed for a full TCP/IP stack. uIP implementation can only handle a single network interface and contain IP, ICMP, UDP and TCP Protocol. [4]

Real Time Operating System management also will be discussed in this report. The purpose is to allow user to do multiple tasks to a single processor attached at the hardware simultaneously without the system becoming unresponsive. The scheduling algorithm is used to finish and complete real-time function within a given time without any failure in the system. As a result, the combination of establishing data communication via Ethernet and RTOS implementation in Atmel's AT91SAM7X256 ARM7 board will be discussed in this report.

C languages will be used in GCC platform by the student. Even though, there is a software development tools for embedded systems called IAR systems that should be easier to be managed, it has some limitations that will be explained later. GCC is the leading free (open source) compiler environment, widely used in the industry. Though, it's really hard to be implemented and waste of time, there are no issues of confidentiality and limitations for sharing information in the joint research. FreeRTOS and uIP TCP/IP Stack open sources will be combined by the student as a mini Real Time Kernel routine and TCP/IP protocol stack respectively.

1.2 Project Aim, Objectives and Plan of Development

The fundamental aim of this project is to design and program AT91SAM7X ARM Based for networking purposes. As the result, the embedded platform can be accessed through a web server that can be used to control, transmit, and receive data to the Input Output peripherals remotely.

The project has four primary goals:

1. Analyzing the hardware characteristic and functional descriptions of AT91SAM7X256 as embedded platform chip and DM9161A as power transceiver for Ethernet.
2. Observing Real Time Operating System Algorithm that can be implemented into a single processor.
3. Investigating uIP TCP/IP stack to build several TCP applications in small resources. Transmitting and Receiving data will be tested.
4. Controlling Input Output Peripherals in the hardware. User LED and Joystick will be involved.

The remainder of this report is organized as follows. Chapter 1 is the introduction of the report. Chapter 2 briefly describes the hardware characteristic of Atmel AT91SAM7X256 as an embedded platform and its functional description. Chapter 3 will explain about the TCP/IP Protocol fundamental. Chapter 4 and 5 will discuss about Real Time Operating System that will be set up by using FreeRTOS open source and TCP/IP Stack respectively. The Embedded Server implementation and algorithm design explanation will be in Chapter 6 and the Software demonstration will be describe in Chapter 7.

Chapter 2 - Hardware Characteristic

2.1 Atmel's AT91SAM7X256

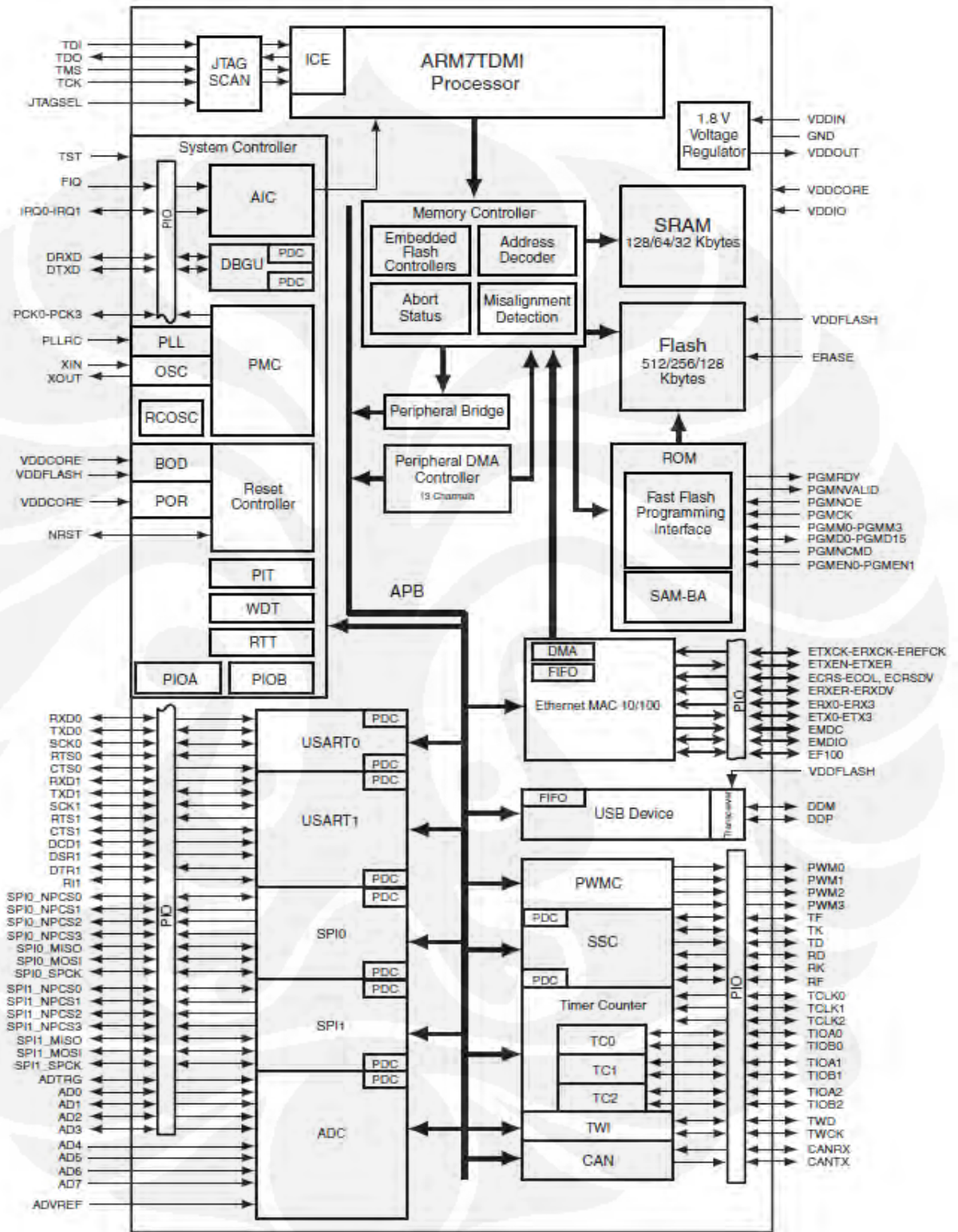
Atmel's AT91SAM7X256 is a member of a series of highly integrated flash microcontrollers based on the 32-bit ARM RISC processor. It features 256 Kbyte high-speed Flash and 64 Kbyte SRAM, a large set of peripherals, including an 802.3 Ethernet MAC and CAN controller, USART, SPI. [2]

The embedded Flash memory can be programmed by downloading the source code via JTAG-ICE interface or via parallel interface on a production programmer prior to mounting. Built-in lock bits and a security bit protect the firmware from accidental overwrite and preserve its confidentiality.

As we can see, to fulfill the requirement of this project, student needs to evaluate the hardware characteristic of EMAC, Davicom DM9161A chip, and PIO Controller to be synchronized with real time operating systems to ARM processor.



Figure 1 - AT91SAM7X-EK Hardware



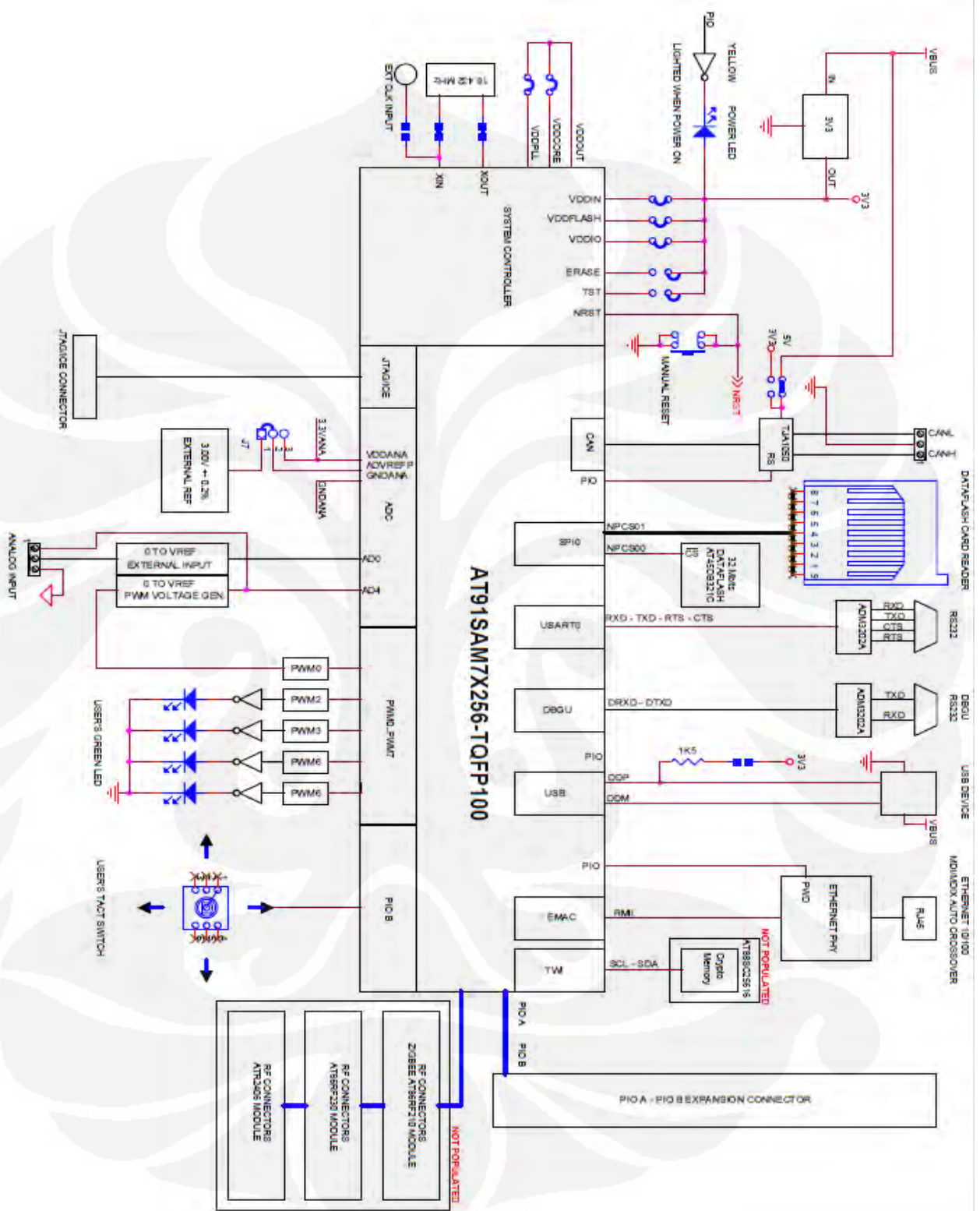


Figure 2 - AT91SAM7X256 Block Diagram

2.2 Ethernet MAC

The EMAC module implements a 10/100 Ethernet MAC compatible with the IEEE 802.3 standard using an address checker, statistic and control register, receive and transmit blocks, and a DMA interface.

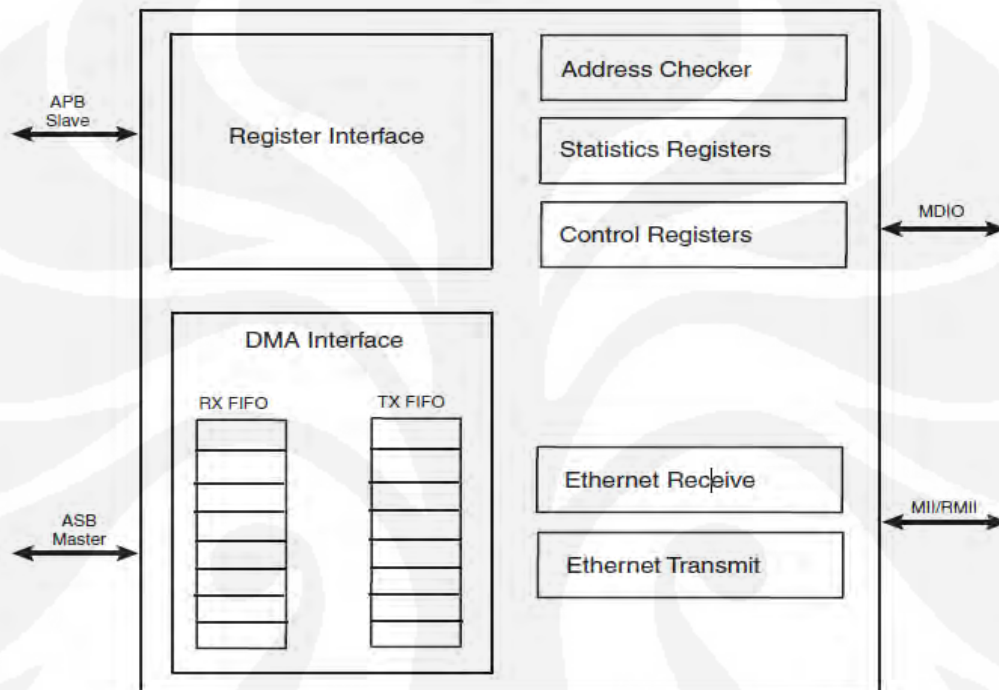


Figure 3 - EMAC Block Diagram

As we can see from figure 2, the explanation the functional description should be like student mention below:

- **Address Checker.** It will recognize four specific 48-bit addresses and contains a 64-bit hash register to match between multicast and unicast addresses, and then copy all frames to the memory and act on an external address match signal.
- **Statistic Register.** It contains registers for counting various types of event associated with transmit and receive operations, for example network management statistics.

- **Control Register.** It setups up DMA activity, start frame transmission and select modes of operation such as full or half duplex.
- **Receive Block.** It checks for a valid preamble, FCS, alignment and length, and presents received frames to the address checking block and DMA interface.
- **Transmit Block.** It takes data from the DMA interface, adds preamble end, pad and FCS and transmits data according to the CSMA/CD (Carrier Sense Multiple Access with Collision Detection). The start of transmission is deferred if CRS (Carrier Sense) is active. If the system is in full duplex mode, the Carrier Sense and Collision have no effect because full duplex because transmitting and receiving path are split into 2 channels.
- **DMA Interface.** The DMA block connects to external memory through its ASB bus interface and contains of Transmit and Receive FIFOs for buffering frame data. It loads the transmit FIFO and empties the receive FIFO. Receive data will not be sent to the memory until the address checker has determined that the frame should be copied. The length of Receive buffers is 128 bytes and Transmit buffers range in length between 0 and 2047 bytes. As summaries, DMA block manages transmit and receive frame buffer queues and can hold multiple frames.

2.3 DM9161A – 10/100 Mbps Fast Ethernet Physical Layer Single Chip Transceiver

The DM9161A Fast Ethernet single chip transceiver, providing the functionality as specified in IEEE 802.3u, integrates a complete 100 Base-TX module with Unshielded Twisted Pair Category 5 Cable (UTP5) and a complete 10 Base-T Module with UTP5/UTP3. Through the Media Independent Interface (MII), it can be connected to the Medium Access Control (MAC) layer. Figure 4 shows the major functional blocks implemented in the DM9161A chip and figure 5 describes the complete tasks how each block works.

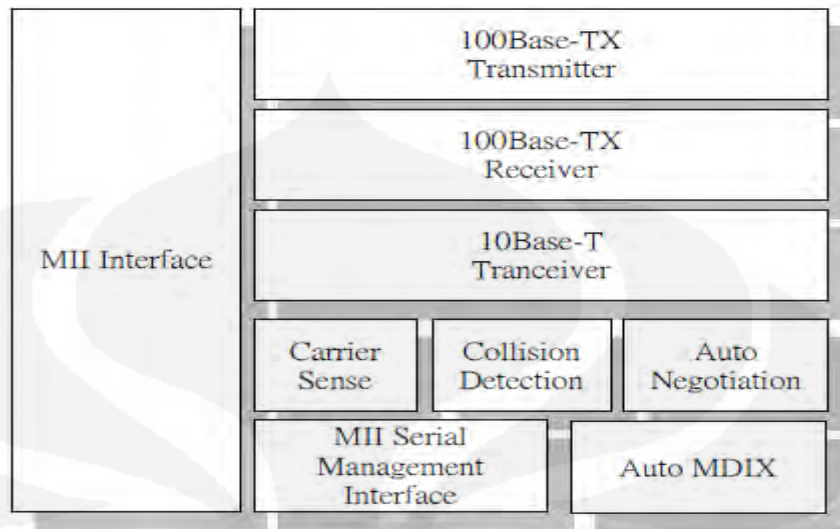


Figure 4 - DM9161A Chip General Functional Description

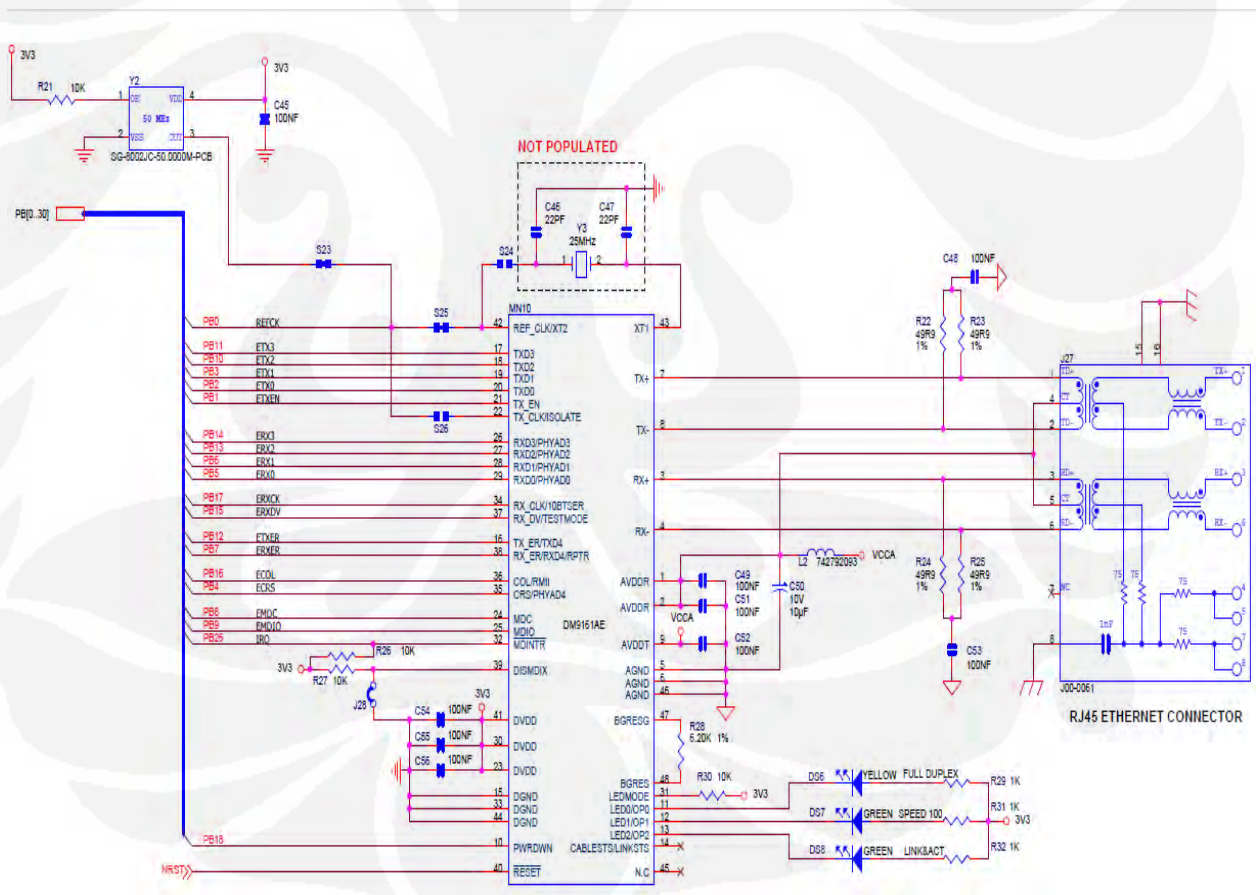


Figure 5 - DM9161A Schematic

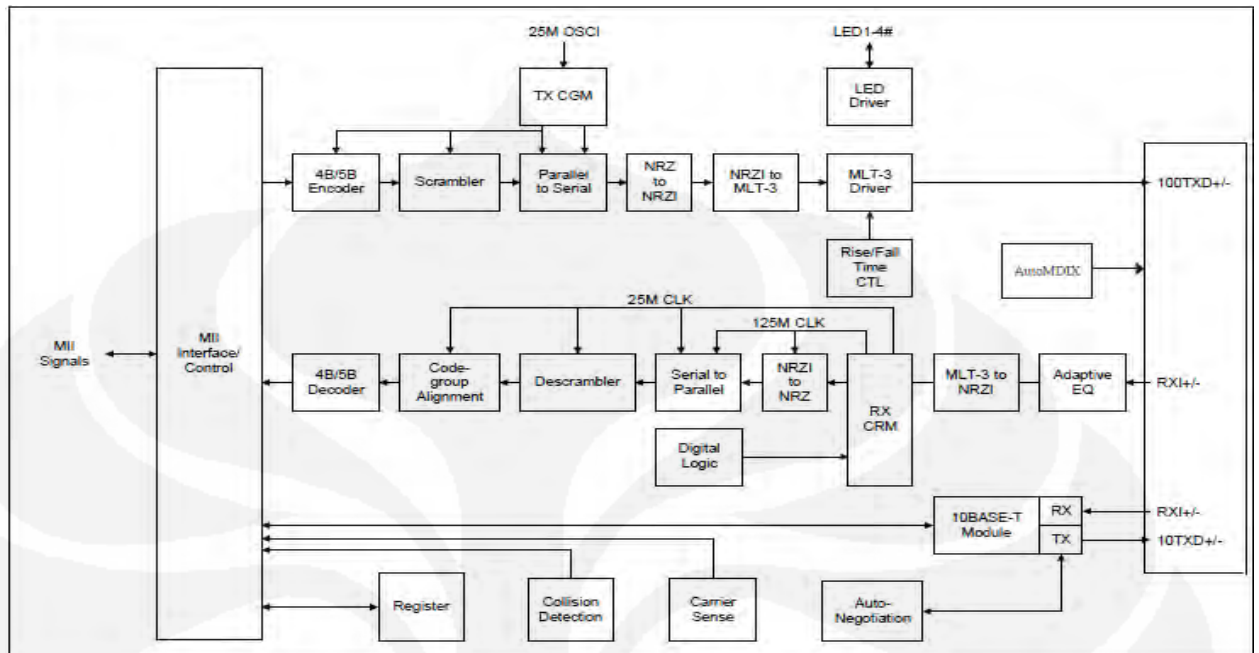


Figure 6 - Specific Functional Description

- MII Interface.** The purpose of Media Independent Interface (MII) is to provide a simple, easy to implement connection between the MAC reconciliation layer and the PHY. The MII is designed to make the differences between various media transparent to the MAC sub-layer.

MII Interface consists of a nibble wide receive data bus, a nibble wide transmit bus, and control signals for data transfer between PHY and the reconciliation layer. Table 1 shows the important bus in MII interface.

BUS	Function
TXD (transmit data)	A nibble of data that synchronous with respect of TXCLK.
TXCLK (transmit clock)	Continuous clock that provides the timing reference for the transmission transfer.
TXEN (transmit enable)	A nibble of data being presented on the MII for transmission.
TXER (transmit error)	Error detected somewhere in the frame being

error)	transmitted.
RXD (receive data)	A nibble of data that synchronous with respect of RXCLK.
RXCLK (receive clock)	Continuous clock that provides the timing reference for receiving transfer.
RXDV (receive data valid)	PHY is presenting decoded nibbles to the MAC reconciliation sub-layer.
RXER (receive error)	Error detected somewhere in the frame transmission for the PHY to the reconciliation layer.
CRS (carrier sense)	CRS is asserted when either transmit or receive medium is being processed.

Table 1 - Busses Function in MII Interface

- **100Base-TX Transmitter.**

As shown in figure 5, 100Base-TX Transmitter consists of the functional block that converts a nibble synchronous data provided by the MII to a scrambled MLT-3 125, a million symbols per second data stream. It contains the following functional diagram:

1. **4B5B Encoder**

It converts 4-bit nibble data from MAC reconciliation layer into 5-bit code group for transmission. This conversion is required for control and packet data to be combined in code groups. To convert them, see appendix table 4B5B as a reference.

2. **Scrambler**

The scrambler is required to control the radiated emission so that the total energy presented to the cable is distributed over a wide frequency range. The result is a data stream with sufficient randomization to decrease radiated emission at critical frequencies.

3. **Parallel to Serial Converter**

It receives 5-bit scrambled data. In order to be able operated by NRZ to NRZI Encoder, the parallel data stream should be serialized.

4. NRZ to NRZI Encoder

Data stream should be converted from serialized NRZ that receive from parallel to serial converter, to NRZI for TP-PDM standard compatibility over Category -5 unshielded twisted pair cable.



Figure 7 - NRZ to NRZI Encoding Example

5. NRZI to MLT-3

Then MLT-3 conversion is accomplished by converting the NRZI data stream into two binary data stream with alternately phased logic one events.

6. MLT-3 Driver

The two binary data streams are fed to the twisted pair output driver that will be compatible of the transmit transformer's primary winding, resulting in a minimal current MLT-3 Signal.



Figure 8 - MLT-3 Converter

- **100Base-TX Receiver**

Data stream received by the chip should have converted to synchronous 4-bit nibble data, so that can be processed by the MII. It contains the following functional diagram:

- 1. Signal Detect**

It should have met the specifications mandated by ANSI XT12 TP-PMD 100Base-TX Standards for both voltage threshold and timing parameters.

- 2. Adaptive Equalizer**

When receiving data from copper twisted at high speed, attenuation based on frequency can affect the randomness of the scrambled data stream. This variation in signal attenuation caused by frequency variations must be compensated for to ensure the integrity of the received data. Moreover, it should have to be adaptive to ensure proper condition of received signal.

- 3. MLT-3 to NRZI Decoder**

Then MLT-3 to NRZI applied as shown in Figure 7.

- 4. Clock Recovery Module**

Clock Recovery Module will lock onto the current data stream and extract reference clock, so that the data stream can be processed at NRZI to NRZ Decoder.

- 5. NRZI to NRZ Decoder**

Data stream is required to be decoded to NRZ signal to be presented to the Serial to parallel conversion block.

- 6. Serial to Parallel**

The NRZ data stream then will be converted to parallel to be presented to the descrambler

- 7. Descrambler**

Because the data stream is scrambled in order to minimize radiated emission for transmission data, it should have to be descrambler and present to the Code Group Alignment group before it can be processes in MII interface.

8. Code Group Alignment

Un-aligned 5B data from descrambler will be converted to 5B code group data in order to make aligned subsequent data on a fixed boundary.

9. 4B5B Decoder

Finally, Conversion from 5-bit data to 4-bit nibble data is accomplished by 4B5B decoder to be ready presented to the reconciliation layer by MII interface.

- **10 Base-T Operation**

When 10 Base-T Mode is operated, for transmission, a nibble data format will be converted to a serial bit stream then encoded by Manchester encoder. When receiving, the data stream will be decoded and converted to nibble format to be presented to the MII interface.

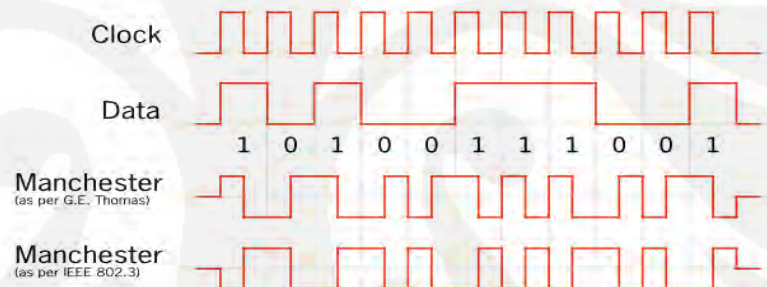


Figure 9 - Manchester Encoding Example

- **Carrier Sense**

CRS is used in half-duplex operation during transmission or reception of data in order to avoid collision in traffic.

- **Collision Detection**

Collision Detection also works in half-duplex operation when transmit and receive channels are active at the same time. It will be reported by COL signal on the MII interface.

- **Auto Negotiation**

The function of Auto-negotiation is to provide a means to exchange information between segment linked devices and to automatically configure both devices to take maximum advantages of their abilities.

- **MII Serial Management**

MII serial management interface consists of a data interface, basic register set, and a serial management interface to configure multiple PHY devices, get status and error information, and also determine the type and capabilities of the attached PHY device. The serial control interface uses a simple two wired serial interface to obtain and control physical layer. It consists of MDC (Management Data Clock), and MDI/O (Management Data Input/Output) signals which pin is bi-directional and shared up to 32 devices.

- **Auto MDIX**

Common Ethernet network cables are straight and crossover cable. This Ethernet network cable is made of 4 pair high performance cable that consist twisted pair conductor that used for data transmission. Auto MDIX is used to detect cable connection type, so that those cables still can be worked into Ethernet interface.

2.4 Parallel Input/Output Controller

PIO controller provides multiplexing up to two peripheral functions on a single pin. Peripheral A and peripheral B represent Joystick Input and User LED respectively. PIO Controller manages up to 32 fully programmable input/output lines. Each I/O line dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral.

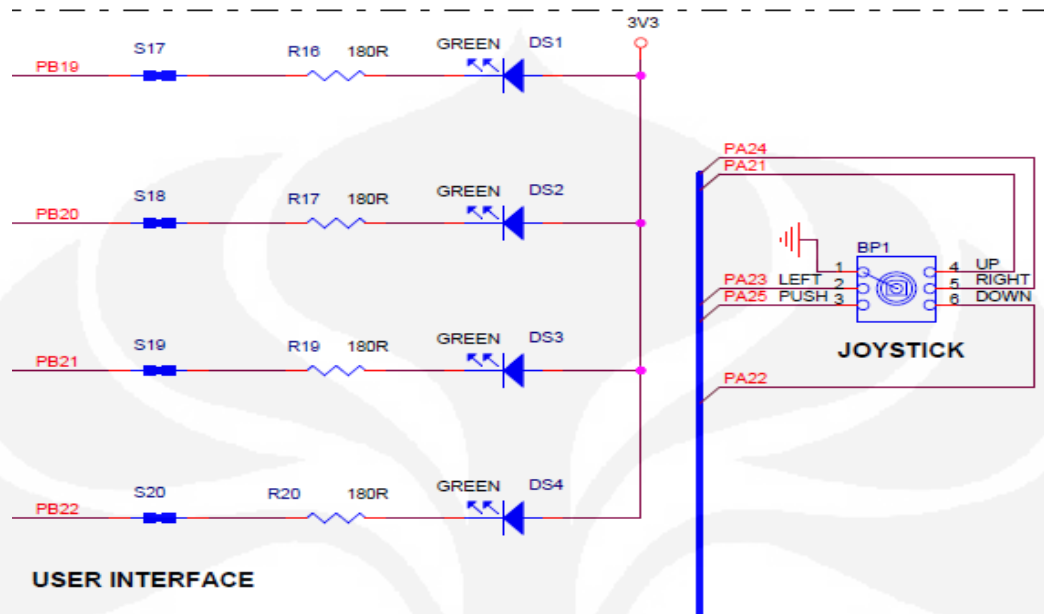


Figure 10 - PIO Schematic

Each pin is configurable according to product definition so programmer must carefully determine the configuration of the PIO controller required by their application. There are some aspects that should be consider controlling the PIO such as Pull-up resistor control, Peripheral Function Selection, Output Control, Synchronous Data output, Input data, Input Change Interrupt, etc. Table 2 describes PIO controller that associated with a bit in each of the PIO Controller User Interface Register.

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or ⁽²⁾ Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register ⁽⁴⁾	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Table 2 - PIO Register Mapping

Chapter 3 - Communication Protocol

To establish connection between the embedded platform and other devices, we need to consider how they actually communicate each other. In this section student will explain about the protocol hierarchies, service primitive, TCP/IP Reference Model.

3.1 Protocol Hierarchies

Basically Protocol is an agreement between the communicating parties on how communication is to proceed. To reduce the design complexity, most networks are organized as a stack of layers or levels, each one built upon the one below it. The purpose of each later is to offer certain services to the higher layers, shielding those layers from the detail of how the offered services are actually implemented. Each layer is a kind of virtual machine, offering certain services to the layer above it.

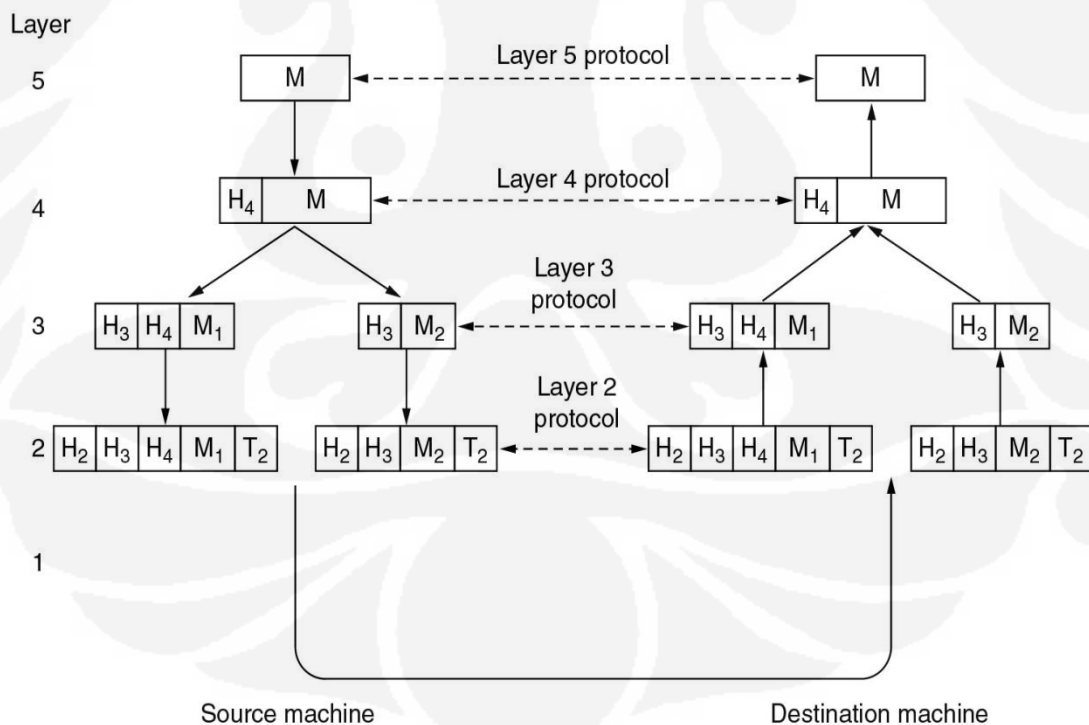


Figure 11 - Example Information Flow Supporting Virtual Communication

As illustrated in Figure 11, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data to the layer immediately below it, until the data reach to the lowest layer. Layer 1 is the physical medium which actual communication occurs.

The entities comprising the corresponding layers on different machines are called peers. The peers may be processes or hardware devices. In other words, it is the peers that communicate by using the protocol. Between each pair of layers is an interface. The interface defines which primitive operations and services the lower layer makes available to the upper one.

3.2 Service Primitive

A service is formally specified by a set of primitives (operations) available to a user process to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packet. [5]

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

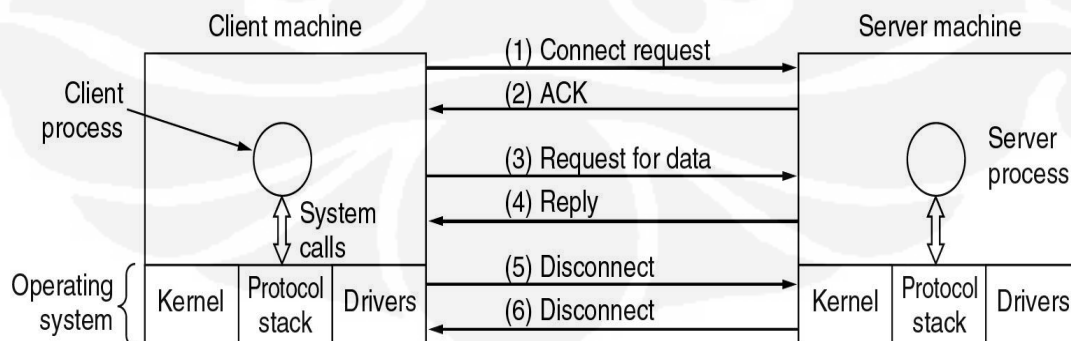


Figure 12 - Packet sent in a simple client-server interaction on a network

These primitives might be used as follows. First, the server executes LISTEN to indicate that it is prepared to accept incoming connections. Next, the client process executes CONNECT to establish a connection with the server (1). The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. When the packet arrives at the server, it is processed by server's operating system to see if there is a listener and send back the acknowledgment (2).

The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before acknowledgement can get back to the client. Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply. After the arrival of the request packet at the server machine will process the request by uses SEND to return the answer to client (4). If it is done, it can use DISCONNECT to terminate the connection. It used a handshake scheme to end communication between server and client.

3.3 TCP/IP Reference Model

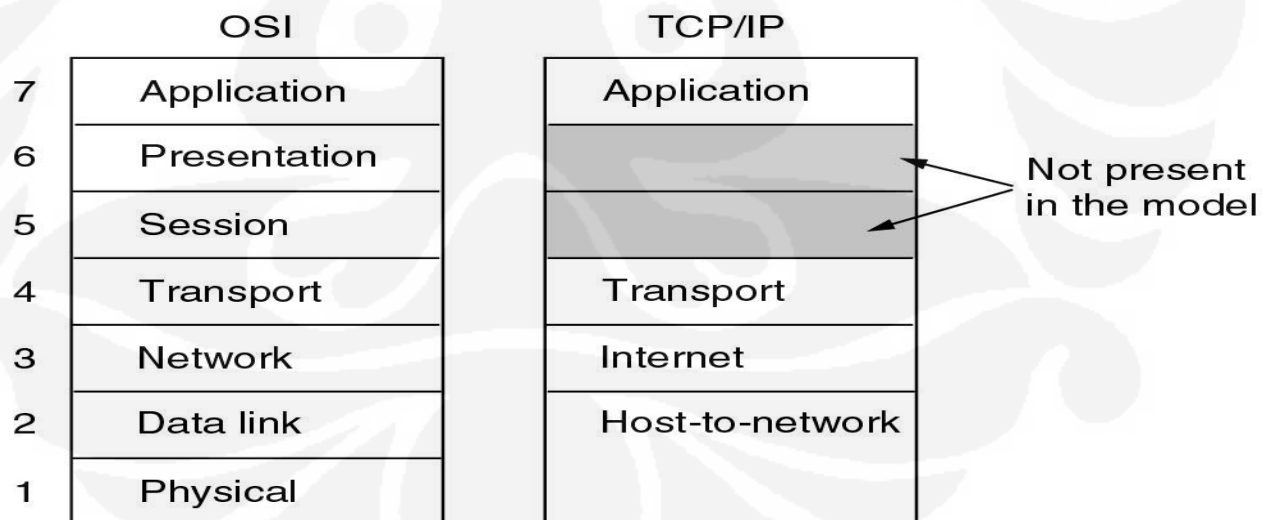


Figure 13 - the difference between OSI and TCP/IP Reference Model

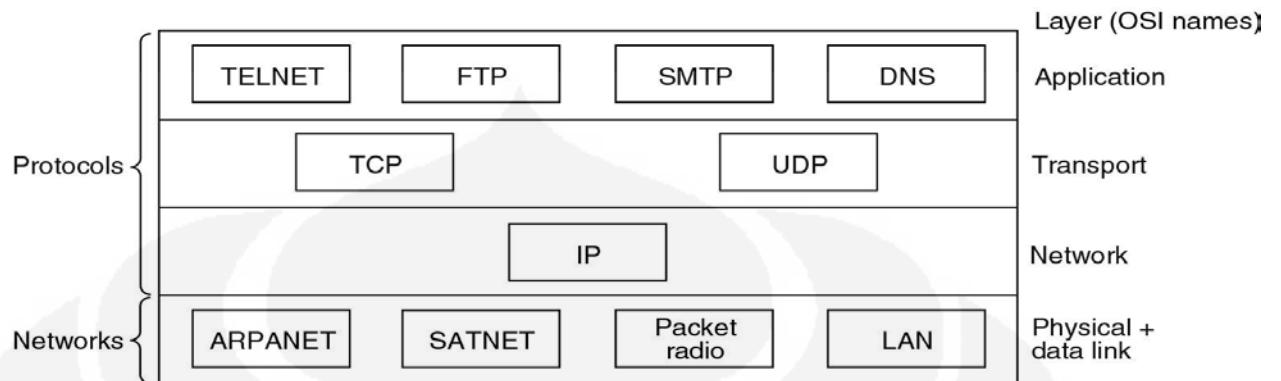


Figure 14 - Protocol and Networks in the TCP/IP model

Application Layer

The TCP/IP model does not have session and presentation layer like OSI model. Therefore, it is not necessary for both layers to be perceived, so they were not included. Application layer contains all the higher-level protocol, for example virtual terminal (Telnet), File Transfer Protocol (FTP), electronic mail (SMTP), protocol for fetching pages (HTTP) etc.

Transport Layer

Transport Layer is designed to allow peer entities on the source and destination hosts to carry on a conversation. As shown in Figure 13, there are 2 protocols that have been defined; TCP and UDP. TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error. UDP (User Datagram Protocol) is an unreliable, connectionless protocol application that does not want TCP's sequencing or flow control and wish to provide their own.

Internet Layer

The internet layer defines an official packet format and protocol called IP (Internet Protocol). The job of the internet layer is to deliver IP packets where they are supposed to go and select the best path through the network for packet to travel. ICMP, ARP also operated at this layer.

Host-to-network layer

TCP/IP reference model does not really say much about what happen in this layer, except that the host has to connect to the network using some protocol so it can send IP packets to it. This protocol is not define and varies from host to host and network to network.

3.4 TCP Protocol

Transmission Control Protocol that referred at transport layer in TCP/IP reference model is used in this implementation. It provides reliable end-to-end delivery service including data transmission and flow control. Using the reliable service, there must not any data loss and the frame has to be reassembled in the right places and makes up for Internet Protocol's (IP) deficiencies. TCP adds a great deal of functionality to the IP service compare to UDP is layered over:

- **Reliable Delivery and Round Trip Estimation.** Sequence numbers are used to coordinate which data has been transmitted and received. TCP will arrange for retransmission if it determines that data has been lost in expected time period.
- **Network Adaptation.** TCP will dynamically learn the delay characteristics of a network and adjust its operation to maximize throughput without overloading the network
- **Flow Control.** TCP manages data buffers, and coordinates traffic so its buffer will never overflow. Fast senders will be stopped periodically to keep up with slower receivers.

TCP is provided through three mechanisms:

1. Acknowledgment.

When a receiver gets a message from a transmitter, the receiver has to acknowledge the message by sending acknowledgment to the transmitter.

2. Sliding Windows

The receiver stop the transmitter from sending messages if a message was dropped. Then the receiver tells the transmitter the number of message was expected to be continued.

3. Sequence Number

TCP uses a 32-bit sequence number that counts bytes in the data stream. Each TCP packet contains the starting sequence number of the data in that packet, and the sequence number of the last byte received from the remote peer. With this information, a sliding window is implemented and each TCP peer must track both its own sequence numbering and the numbering being used by the remote peer. If the number received is not in sequence, the receiver will tell the transmitter that the number was wrong and gave it expected number to be retransmitted.

And also when the client want to terminate the established connection it uses a handshake scheme to end communication. Client will send message with FIN flag set to indicate that the client want to terminate the connection. When server receives the message, it will send the acknowledgment first and then terminate its connection.

Chapter 4 – Real Time Operating System

GCC environment is decided to use in this project, therefore open source called FreeRTOS is used to implement a scale-able real time kernel that designed specifically for small embedded systems. It means that routine and some modules in the program implementation are based on this open-source. There are some advantages using FreeRTOS open-source are [11]:

- Preemptive, cooperative and hybrid configuration options.
- Designed to be small, simple and easy to use.
- Very portable code structure predominantly written in C.
- Support both tasks and co-routines.
- Stack-overflow detection options.

4.1 RTOS Concept

In order to develop FreeRTOS to be satisfied to our needs, it is essential to know the fundamental and the background of RTOS concept. Student will write RTOS concept such as Multitasking, Scheduling, Context Switching, and how they will be processed in running task and co-routines.

Multitasking

A conventional processor can only execute a single task at a time, on the other hand by rapidly switching between tasks a multitasking operation system can make it appear as if each task is executing concurrently. Figure 14 explain the execution pattern of three tasks with respect to time. The upper diagram demonstrates the perceived concurrent execution pattern and the lower the actual multitasking execution pattern.

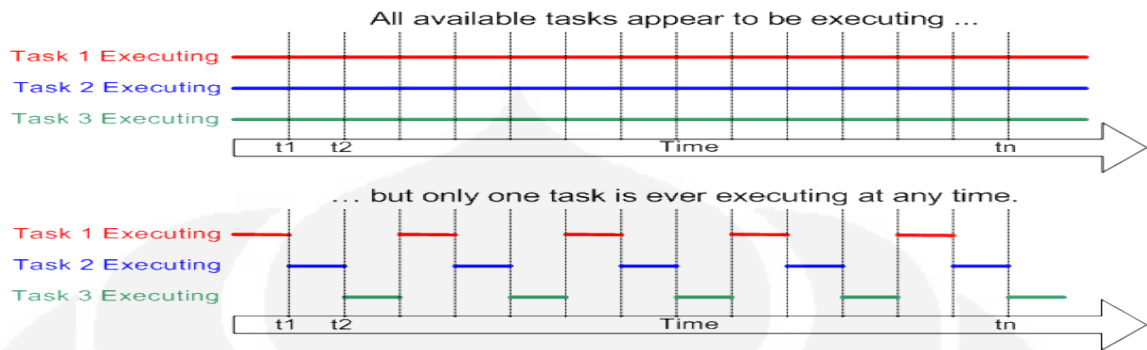


Figure 15 - Conventional VS Multitasking

Scheduling

The scheduler is part of the kernel that is responsible to decide which task should be executed at any particular time. The scheduling policy is the algorithm used by the scheduler to decide which task to execute at a specific time.

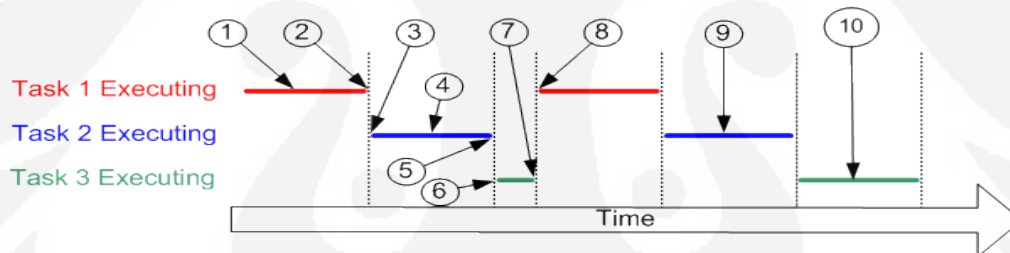


Figure 16 - Scheduling Example

Referring to the numbers in figure above:

- At (1) task 1 is executing.
- At (2) the kernel suspends task 1.
- At (3) resumes task 2.
- While task 2 is executing (4), it locks a processor peripheral for its own exclusive access.
- At (5) the kernel suspends task 2.
- At (6) resumes task 3.

- Task 3 tries to access the same processor peripheral, finding it locked task 3 cannot continue so suspends itself at (7).
- At (8) the kernel resumes task 1.
- The next time task 2 is executing (9) and it finishes.
- The next time task 3 is executing (10) and it finishes.

Context Switching

As a task executes, it utilizes the microcontroller registers and accesses RAM and ROM just as any other program. These resources such as processor register, stack, etc comprise the task execution context. While the task is suspended, other task will execute and may modify the processor register values. Upon resumption, the task will not know that the processor have been altered and result in an incorrect value.

The operating system kernel is responsible to ensure saving the context of a task as it is suspended. So, when the task is being resumed, its saved context is restored and task will continue in a correct value.

4.2 The differences between Task and Co-routines

There are several API references such as Task creation control utilities, Kernel Control, Queues, Semaphore/Mutexes and Co-routines those being used in FreeRTOS open-source as a default. However, student only added and modified some aspects that can fulfill the project requirements in Task management and Co-Routines. Therefore, student will explain the differences between those two.

TASK

In a real time application that uses an RTOS can be structured as a set of tasks and each task executes within its own context. Unfortunately, only one task within the application can be executed at any point in time and real time scheduler is responsible for deciding which task should be executed. Therefore, scheduler repeatedly starts and stops each task, and as a task has no knowledge of the scheduler activity, scheduler also responsible for context switching process when a task is being swapped out and swapped in. To achieve this one, each task is provided with its own stack, so the context will be saved to the stack of that task. As a result, it will use high RAM usage.

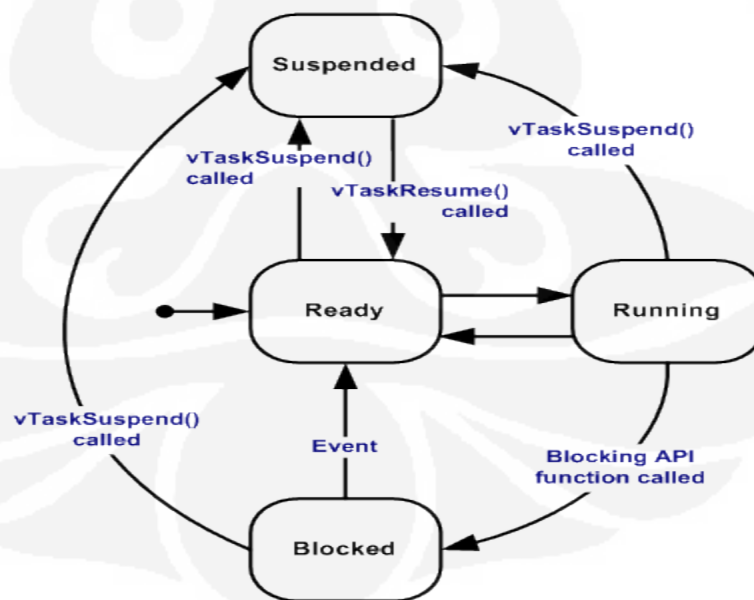


Figure 17 - Task States

A task can exist in one of the following states:

- **Running**

When a task is actually executing, it is said to be in the Running state. It is currently utilizing the processor.

- **Ready**

Ready tasks are those that are able to execute but are not executing because a different task of equal or higher priority is already in the running state. They are not blocked or suspended.

- **Blocked**

A task is said to be in the blocked state if it is currently waiting for either a temporal or external event and it will block until the delay period has expired. Blocked tasks are not available for scheduling.

- **Suspended**

Task in Suspended state are also not available for scheduling. Tasks will only enter or exit the Suspended state when explicitly commanded to do, so there are going to have two functions; Suspend and Resume.

Each task is assigned a priority. The scheduler will ensure that a task in the ready or running state will always be given processor time in preference to tasks of a lower priority that are also in the ready state. In other words, the task given processing time will always be the highest priority task that is able to run.

Co-Routine

All the co-routines within an application share a single stack. This reduces the amount of RAM usage compared with tasks. Co-routines use prioritized cooperative scheduling with respect to other co-routines and also its implementation is provided through a set of macros.

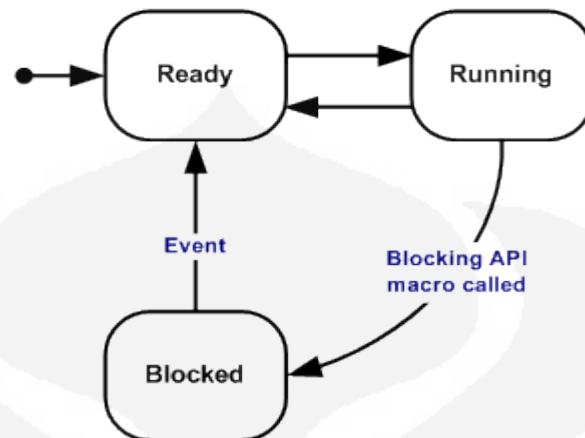


Figure 18 - Co-Routine States

A co-routine can exist in one of the following states:

- **Running**

When a co-routine is actually executing it is said to be in the running state. It is currently utilizing the processor.

- **Ready**

Ready co-routines are those that are able to execute (they are not blocked) but are not currently executing. A co-routine may be in the ready state because of 2 reasons:

1. Another co-routine of equal or higher priority is already in the running state.
2. If the application uses both tasks and co-routines, a co-routine might be in the ready state when the task is in the running state.

- **Blocked**

A co-routine is said to be in the Blocked state if it is currently waiting for either a temporal or extended event.

4.3 FreeRTOS Open-Source Application Demonstration

The table below lists the files that make up the demo projects along with a brief indication of the RTOS features demonstrated and describes each task and co-routine within the demo project.

File	Features Demonstrated
main.c	<ul style="list-style-type: none"> • Starting/Stopping the kernel • Using the trace visualisation utility • Allocation of priorities
dynamic.c	<ul style="list-style-type: none"> • Passing parameters into a task • Dynamically changing priorities • Suspending tasks • Suspending the scheduler
BlockQ.c	<ul style="list-style-type: none"> • Inter-task communications • Blocking on queue reads • Blocking on queue writes • Passing parameters into a task • Pre-emption • Creating tasks
ComTest.c	<ul style="list-style-type: none"> • Serial communications • Using queues from an ISR • Using semaphores from an ISR • Context switching from an ISR

	<ul style="list-style-type: none"> • Creating tasks
CRFlash.c	<ul style="list-style-type: none"> • Creating co-routines • Using the index of a co-routine • Blocking on a queue from a co-routine • Communication between co-routines
CRHook.c	<ul style="list-style-type: none"> • Creating co-routines • Passing data from an ISR to a co-routine • Tick hook function • Co-routines blocking on queues
Death.c	<ul style="list-style-type: none"> • Dynamic creation of tasks (at run time) • Deleting tasks • Passing parameters to tasks
Flash.c	<ul style="list-style-type: none"> • Delaying • Passing parameters to tasks • Creating tasks
Flop.c	<ul style="list-style-type: none"> • Floating point math • Time slicing • Creating tasks
Integer.c	<ul style="list-style-type: none"> • Time slicing • Creating tasks

PollQ.c	<ul style="list-style-type: none">• Inter-task communications• Manually yielding processor time• Polling a queue for space to write• Polling a queue for space to read• Pre-emption• Creating tasks
Print.c	<ul style="list-style-type: none">• Queue usage
Semtest.c	<ul style="list-style-type: none">• Binary semaphores• Mutual exclusion• Creating tasks

Table 3 - FreeRTOS Function

Chapter 5 – TCP/IP Stack

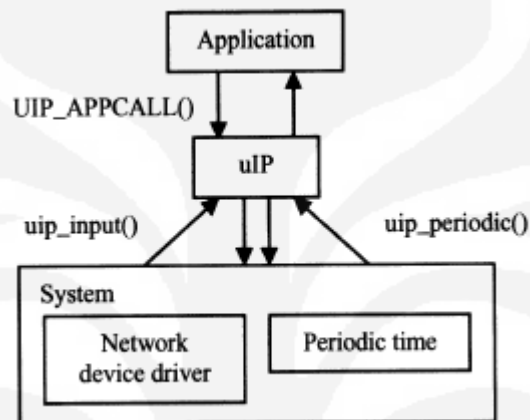
The uIP stack that student used for TCP/IP stack is an open source that is intended to make it possible to communicate using TCP/IP protocol suite even on small 8-bit micro-controllers. The size of the code is only up to a few kilobytes and RAM usage can be configured to be as low as a few hundred bytes.

The uIP TCP/IP stack becomes one of alternatives free open-source to substitute the business version that has a very expensive price in correspondence. Although the effect of business version is perfect, programmers choose some free TCP/IP stacks and improve them to satisfy their needs. The uIP stack can be run either as a task in a multitasking system, or as the main program in a single tasking system.

This uIP TCP/IP Stack has the following features: [4]

- Well documented and well commented source code
- Very small code size
- Very low RAM usage, configurable at compile time
- ARP, SLIP, IP, UDP, ICMP, and TCP protocols.
- Includes a set of example applications: web server, web client, SMTP client, Telnet server, DNS hostname resolver.
- Any number of concurrently active TCP connections.
- Any number of passively listening (server) TCP.
- Free for both commercial and non-commercial use.

5.1 Main Control Loop



The main control loop in uIP stack does two things repeatedly:

- Check if a packet has arrived from the network. (Using function `uip_input()`).
- Check if a periodic timeout has occurred. (Using function `uip_periodic()`).

5.2 Architecture Specific Functions

uIP requires a few functions to be implemented specifically for the architecture to run. C language implementations are given as part of the uIP distribution. Below is the basic function of uIP stack:

1. Checksum Calculation

The TCP and IP protocols implement a checksum that covers the data and header portions of the TCP and IP packets. Since the calculation of this checksum is made over all bytes in every packet being sent and received it is important that the function that calculates the checksum is efficient.

While uIP includes a generic checksum function, it also leaves it open for an architecture specific implementation of the two functions **uip_ipchksum()** and **uip_tcpchksum()**.

2. 32-bit Arithmetic

The TCP protocol uses 32-bit sequence numbers, and a TCP implementation will have to do a number of 32-bit additions as part of the normal protocol processing. Since 32-bit arithmetic is not natively available on many of the platforms for which uIP is intended, uIP leaves the 32-bit additions to be implemented by the architecture specific module and does not make use of any 32-bit arithmetic in the main code base.

While uIP implements a generic 32-bit addition, there is support for having an architecture specific implementation of the **uip_add32()** function.

3. Memory Management

The uIP stack does not use explicit dynamic memory allocation. Instead, it uses a single global buffer for holding packets and has a fixed table for holding connection state. The global packet buffer is large enough to contain one packet of maximum size. When a packet arrives from the network, the device driver places it in the global buffer and calls the TCP/IP stack. If the packet contains data, the TCP/IP stack will notify the corresponding application. Because the data in the buffer will be overwritten by the next incoming packet, the application will either have to act immediately on the data or copy the data into a secondary buffer for later processing.

The total amount of memory usage for uIP depends heavily on the applications of the particular device in which the implementations are to be run. The memory configuration determines both the amount of traffic the system should be able to handle and the maximum amount of simultaneous connections.

4. Application Program Interface (API)

The Application Program Interface (API) defines the way the application program interacts with the TCP/IP stack. The most commonly used API for TCP/IP is the BSD socket API which is used in most Unix systems and has heavily influenced the Microsoft Windows WinSock API. Because the socket API uses stop-and-wait semantics, it requires support from an underlying multitasking operating system. Since the overhead of task management, context switching and allocation of stack space for the tasks might be too high in the intended uIP target architectures, the BSD socket interface is not suitable for our purposes.

5.3 The uIP raw API

The "raw" uIP API uses an event driven interface where the application is invoked in response to certain events. An application running on top of uIP is implemented as a C function that is called by uIP in response to certain events. uIP calls the application when data is received, when data has been successfully delivered to the other end of the connection, when a new connection has been set up, or when data has to be retransmitted. The application is also periodically polled for new data. The application program provides only one callback function; it is up to the application to deal with mapping different network services to different ports and connections. Because the application is able to act on incoming data and connection requests as soon as the TCP/IP stack receives the packet, low response times can be achieved even in low-end systems.

Interface function	Application event
uip_listen()	Start listening on a port
uip_send()	Send data on the current connection
uip_acked()	Sent data has been acknowledged
uip_newdata()	Remote host has sent new data
uip_datalen()	The size of the incoming data

uip_connect()	Connect to a remote host
uip_connected()	The current connection has just been connected
uip_poll()	Application is being polled
uip_close()	Close the current connection
uip_abort()	Abort the current connection
uip_stop()	Stop the current connection

Table 4 - uIP Interface Function

5.4 uIP Simple Application

Hello World (ICMP)

Hello World uIP application is an example showing how to write applications with protosockets function. The protosocket library in uIP provides functions for sending data without having to deal with retransmissions and acknowledgements, as well as functions for reading data without having to deal with data being split across more than one TCP segment.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Bayu Indra Nugraha>ping 10.10.10.10

Pinging 10.10.10.10 with 32 bytes of data:

Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.10.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Bayu Indra Nugraha>arp -a

Interface: 10.10.10.11 --- 0x4
Internet Address      Physical Address      Type
10.10.10.10           00-45-56-78-9a-bc    dynamic

Interface: 131.181.102.172 --- 0x5
Internet Address      Physical Address      Type
131.181.100.1         00-0f-34-a7-1d-44    dynamic

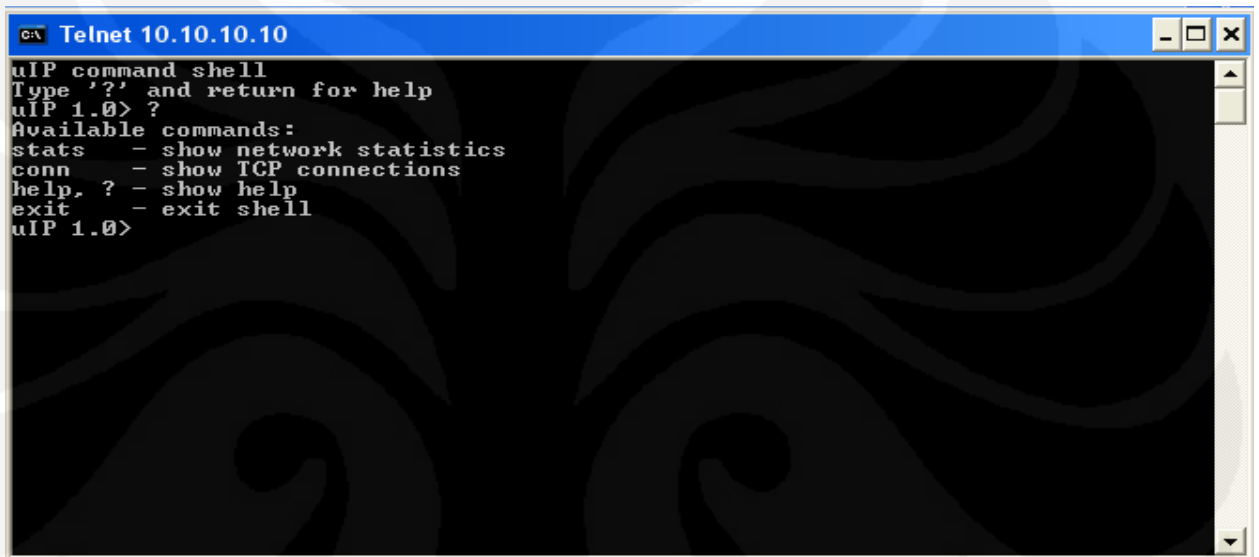
C:\Documents and Settings\Bayu Indra Nugraha>

```

Figure 19 - uIP basic ICMP Demo

Telnet

The purpose of this application is to provide a TCP bidirectional interactive communications facility in port 23. Typically, telnet provides access to a command-line interface on a remote host via a virtual terminal connection which consists of an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). User data is interspersed in-band with TELNET control information.

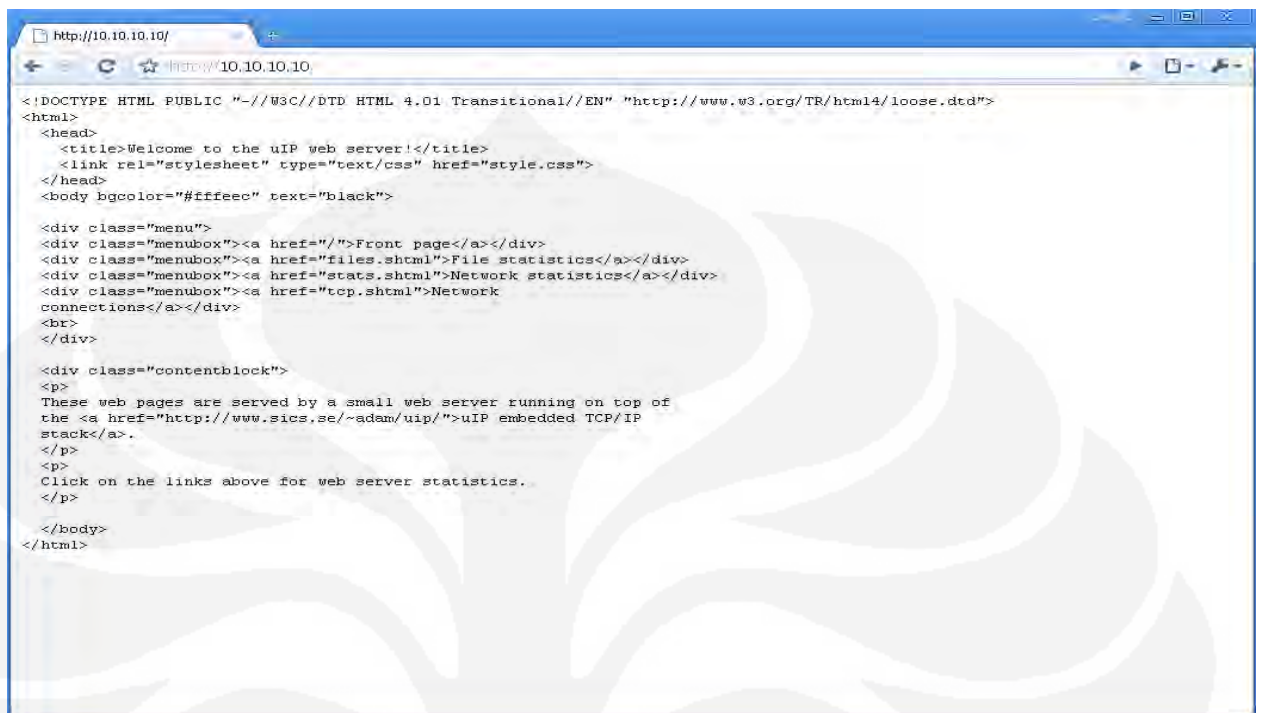


```
uIP command shell
Type '?' and return for help
uIP 1.0> ?
Available commands:
stats - show network statistics
conn - show TCP connections
help, ? - show help
exit - exit shell
uIP 1.0>
```

Figure 20 - uIP Basic Telnet Server Demo

Web Server

The application that responsible for accepting HTTP requests from *clients* (user agents such as web browsers), and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Welcome to the uIP web server!</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body bgcolor="#fffee" text="black">

  <div class="menu">
  <div class="menubox"><a href="/">Front page</a></div>
  <div class="menubox"><a href="files.shtml">File statistics</a></div>
  <div class="menubox"><a href="stats.shtml">Network statistics</a></div>
  <div class="menubox"><a href="tcp.shtml">Network
connections</a></div>
  <br>
  </div>

  <div class="contentblock">
  <p>
  These web pages are served by a small web server running on top of
  the <a href="http://www.sics.se/~adam/uip/">uIP embedded TCP/IP
  stack</a>.
  </p>
  <p>
  Click on the links above for web server statistics.
  </p>
  </div>
</body>
</html>
```

Figure 21 - uIP Basic Web Server Demo

Chapter 6 – Algorithm and Design Implementation

There are 4 development environment options that can be used in implementation;

1. YAGARTO
2. uC/OS-II Micrium
3. IAR
4. Rowley Crossworks for ARM

Except for option 3, these environments are all built around the free compiler tool-chain GCC. Furthermore, YAGARTO is being used in this project since it is free open-source software, has some technical documentation done by software developers and there is no memory limitation issue compare with other. Student chose GCC environment because there are more references on building various API applications even though it is more complicated than other development software. Since it is completely free to experiment and deploy pre-configured demo applications to ensure student to start with a known good and working project, student could develop them until they meet the objectives of the project.

Student use 2 GCC open-sources with C language as a basic demo applications. There are:

1. FreeRTOS, a mini Real Time Kernel
2. uIP stack, TCP/IP stack that provides TCP/IP connectivity

Both open-sources are licensed by GNU General Public that guarantees the freedom to share and change them. Student is responsible to obey GNU General Public restriction by still showing the copyright of the software and offer the license which give the student legal permission to copy, distribute and modify the software.

The new module is created from the combination of FreeRTOS and uIP stack. Several API applications that student have been used to achieve the requirement of the project is discussed in this section. Some ineffective API's that work during the implementation will not be removed since it will make the code structure become unbalance.

6.1 General Program Loop

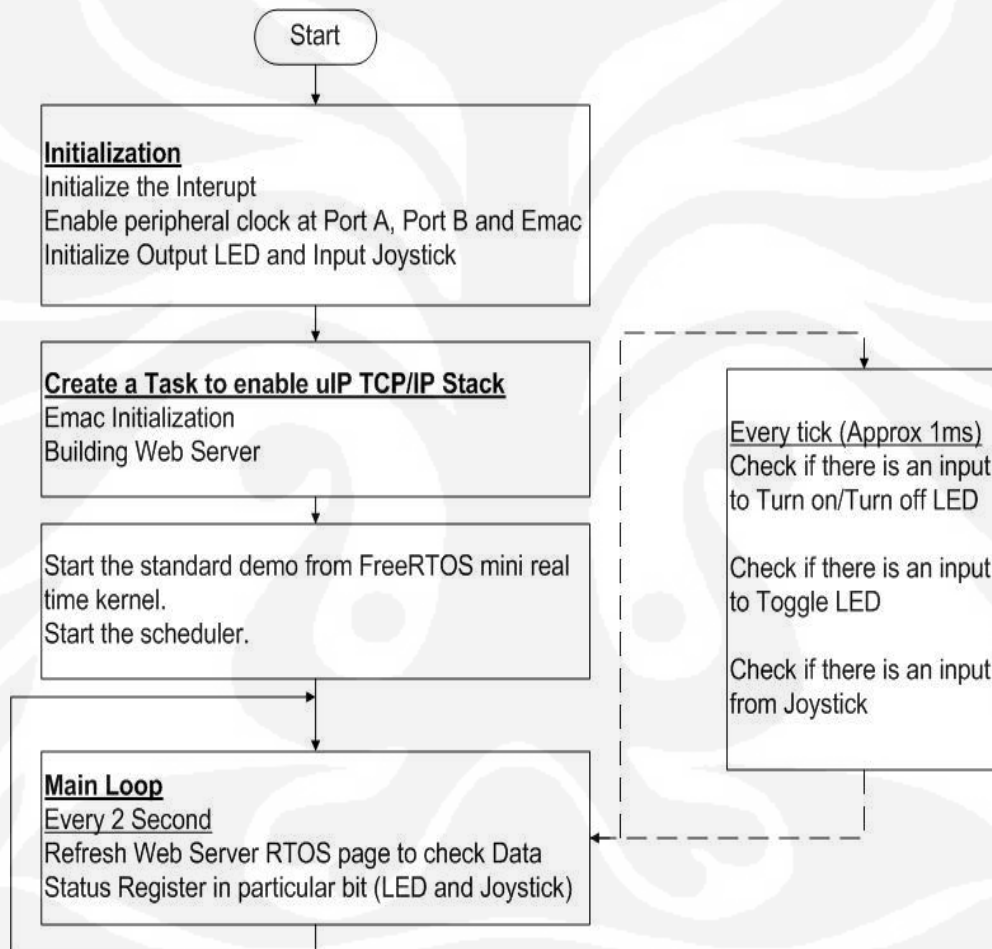


Figure 22 - General Program Loop

Figure 22 describes how the whole program works with infinity loop. The brief explanation of the operations are:

Initialization

Initialize the interrupt. When using JTAG debugger, the hardware is not always initialized to the correct default state. Make sure that issue does not make all interrupt to be masked at the start.

Enable the peripheral clock. It is essential to turn on the clock at peripherals Port A, Port B and the EMAC in order to be able to change bit state at clock transitions in specific time. By “Turn off” the clock mean we block the clock signal to that peripheral.

- **Initialize Output LED and Input Joystick.** To make I/O peripheral to be able work properly, some register in PIO control peripheral need to be enabled. This section will be discussed later.

Create a task to enable uIP TCP/IP Stack. This is how student combine the FreeRTOS and uIP stack by creating a task that provided by FreeRTOS that implement uIP TCP/IP Stack in API function.

EMAC Initialization. To be able to establish the connection and synchronization between EMAC to DM9161A chip, it is necessary to do some general steps for the EMAC initialization;

- Initialize both Tx and Rx descriptor used by the EMAC.
- Enable the Management Data Input/Output bit in MAC Control Register.
- Function to be able read and write value into a PHY register.
- Function to detect MAC and PHY
- EMAC initialization to initialize the Ethernet.
- EMAC initialization to receive packets.
- EMAC initialization to be able to send a packet through EMAC and PHY.
- ICMP, IP, TCP Checksum Calculation.
- Function to be able to send and receive frames.

Building Web Server. Web Server is build by developing the basic web server demo in uIP TCP/IP stack. There are some C project files those become fundamental of the web server in this project;

- **Httpd.c:** This file contains of the macro, or a rule that specifies how a certain input sequence used in HTTPD CGI function. Furthermore, Web server initialization is stored in this file by setting up TCP application in port 80.
- **httpd-fsdata.c:** This file stores HTML script to create web server design. The implementation in this file is using hexadecimal code so that can be read by the processor. As a result ASCII HTML script needs to be manually converted.
- **httpd-fs.c:** This file stores the network statistic that monitored in port 80.
- **httpd-cgi.c:** This file stores web server script interface that can be inserted to httpd-fsdata.c. Student uses functions here in order to create some input/output applications at the web-server.
- **uIP-Task.c:** This file stores the implementation functions that become a bridge from the web server to the embedded platform register.

Start the standard demo from FreeRTOS mini time real time kernel. Several task, co-routine, queue and semaphores are being looped and priorities will be implemented.

Main Loop

Every 2 second: The web server will check the Data Status Register in particular bit in order to check the User LED and Joystick Input status. The web server will be refreshed every 2 second to minimize the load of the network.

Every Tick in Approximately 1 ms

Check an Input user interface from the web server to turn on/turn off or toggle the LED: The bit status for I/O peripheral can be controlled from web server. Student uses 1ms tick to read input user interface status and then update the particular I/O peripheral status.

Check Joystick Input Data Register Status: It has the same concept with above operation. Student uses 1ms tick to read Input from joystick and then update the particular I/O peripheral status and the status at web-server.

6.2 PIO Algorithm

A. Reading Joystick Input

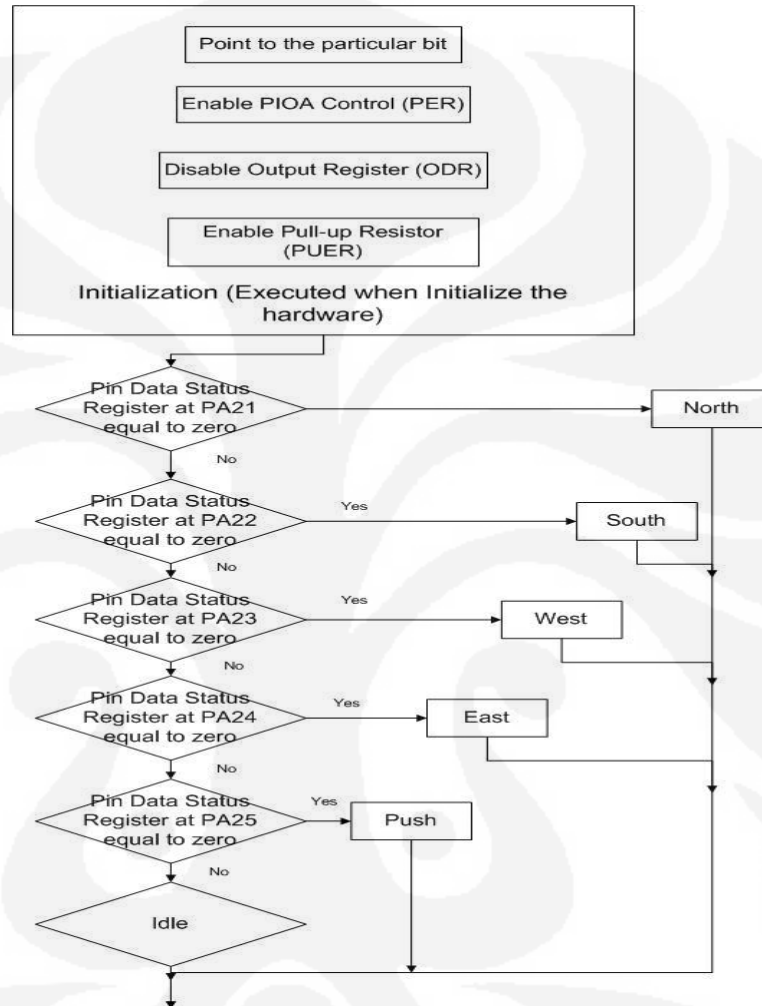


Figure 23 - Reading Joystick Algorithm

Initialization for the input algorithm is executed during hardware initialization at the very beginning of the program. The brief explanations of figure 22 are:

- Point to the particular bit:** There are 6 wires for Joystick Input those represent the 5 data directions and a ground. Student defines those bits those have bit address PIOA21, PIOA22, PIO23, PIO24, and PIO25 as North, South, West, East, and Push respectively.

- **Enable PIOA Control (PER):** The purpose is to enable the PIO controller and ready to be used in the implementation.
- **Disable Output Register (ODR):** The purpose of disabling the output register is to restrict the data status register so that it only can be updated by controlling the joystick manually.
- **Enable Pull-up Resistor (PUER):** When a joystick contact is closed, it will connect the related port bit to ground. Otherwise the port bit will be floating. The floating status can be avoided by enabling the internal pull-up resistor.
- **Check Data Status Register:** In every 1 ms tick, each Data Status Register will be monitored and also will be uploaded to the web-server.

B. Set the LED Status to turn on/turn off the LED

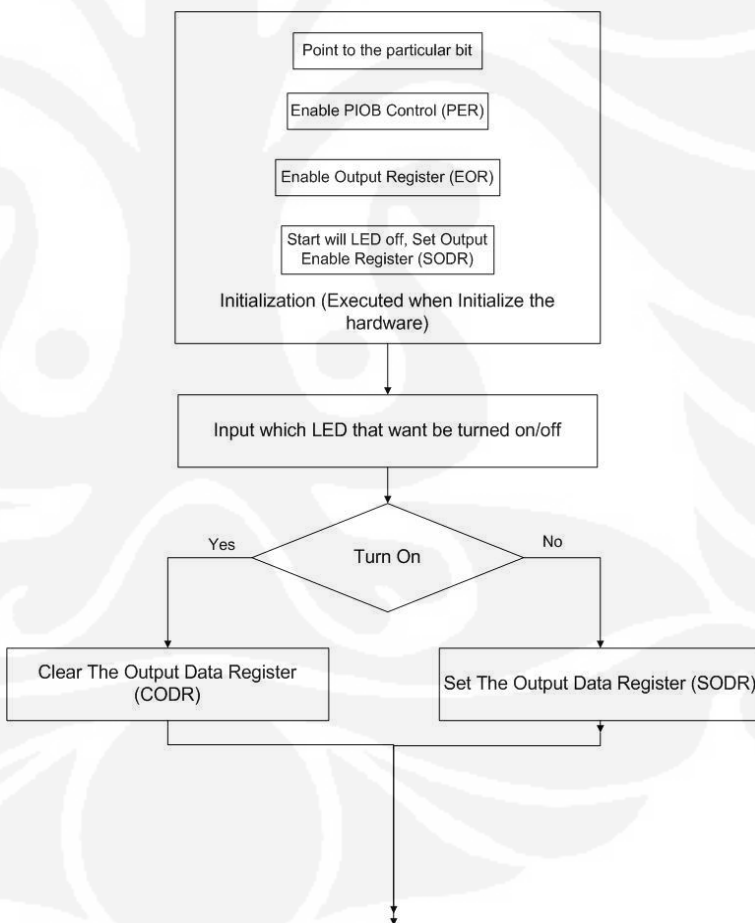


Figure 24 - Algorithm to Set LED Status

Initialization for the input algorithm is executed during hardware initialization at the very beginning of the program. The brief explanation should be like below:

- **Point to the particular bit.** There are 4 address bit that stand for the LEDs; PIOB 19, PIOB20, PIOB21, PIOB22. They are representing LED DS1, DS2, DS3 and DS4 respectively.
- **Enable PIOB Control (PER).** The purpose is to enable the PIO controller and ready to be used in the implementation.
- **Enable Output Register (EOR).** The purpose enabling this register is to be able to update the LED status as an output.
- **Set Output Enable Register (SODR).** The purpose is to make sure the LED is being turned off for the initialization.
- **Set the LED Status.** First, LED which wants to be turned on/off should be defined. Assigning Clear Output Data Register will turn on the LED and assigning Set Output Data Register will turn off the LED.

C. Toggle LED



Figure 25 - Algorithm to toggle the LED

- The delay will be done by controlling the tick port in particular time.
- When calling this toggle function, first it will check the LED Data Status Register and then do the opposite operation from it. The operation will still work until there is an input from Input User Interface to stop calling this toggle function.

D. Transmitting buffer data in Web Server User interface as an input

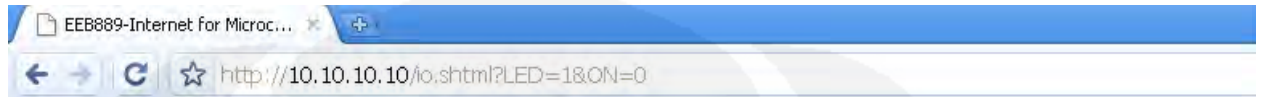


Figure 26 - Web Address

uIP TCP/IP stack is allow us to control the web address as shown in figure 25. As we can see the value of LED and ON are 1 and 0 respectively. By separating these numbers from the web address string and sending those into Peripheral I/O control function, then the LED peripheral can be controlled remotely from the web server. In the figure case, the web server will turn the LED DS1 on.

Chapter 7 – Software Demonstration

The following operations are going to be demonstrated:

- Possible Network Diagrams
- ICMP and ARP
- Web Server Page RTOS Stats
- Web Server Page TCP Stats
- Web Server Page Connection
- Web Server Page IO

7.1 Possible Network Diagrams

There are some real world cases that can be implemented so that the microcontroller can be accessed remotely. The figures below show network diagrams that is suitable for the network implementation.

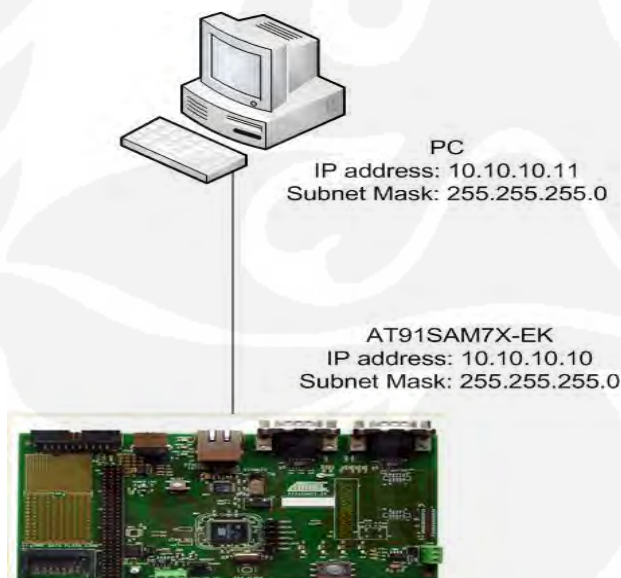
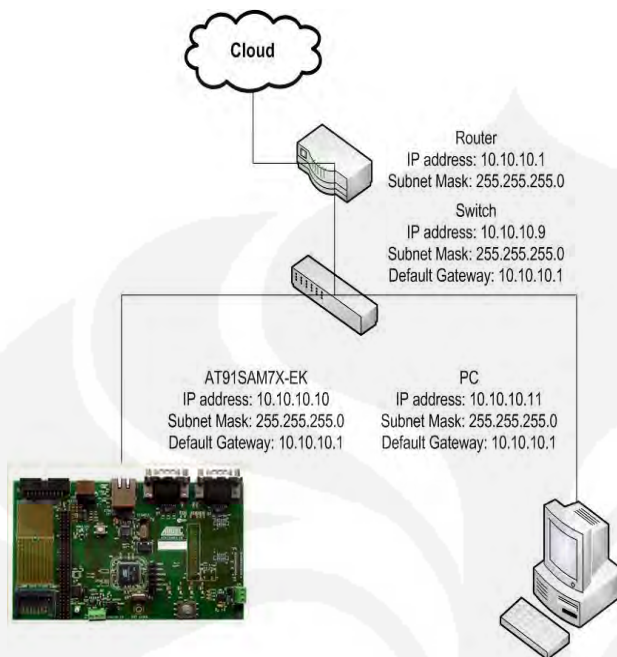


Figure 26 shows the implementation of host to host network. The Davicom D9161A supports Auto MDIX (Cable type detection). Therefore any type of UTP cables can be used to establish the connection. Cross-Over UTP cable is used in this project to avoid traditional issue.

Both IP address are assigned in the same subnet.

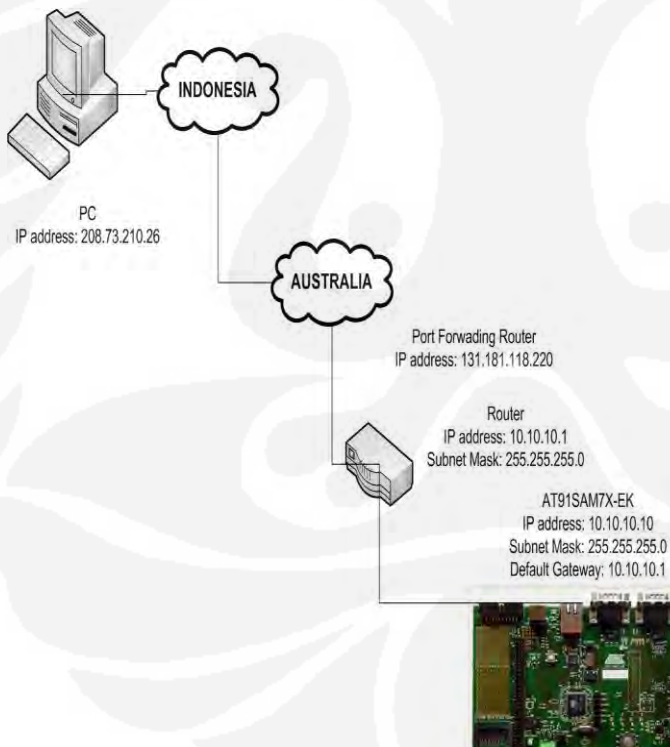
Figure 27 - Host to Host Network



By adding a switch into network diagram, it is still possible to access the embedded platform through Local Area Network.

All IP addresses must be in the same subnets so that all electronic devices can be communicated each other.

Figure 28 - Local Area Network



By adding a router as a gateway, student also could access the embedded platform from Wide Area Network.

The router should be configured to forward the embedded platform's port so that the microcontroller has a public/internet IP address.

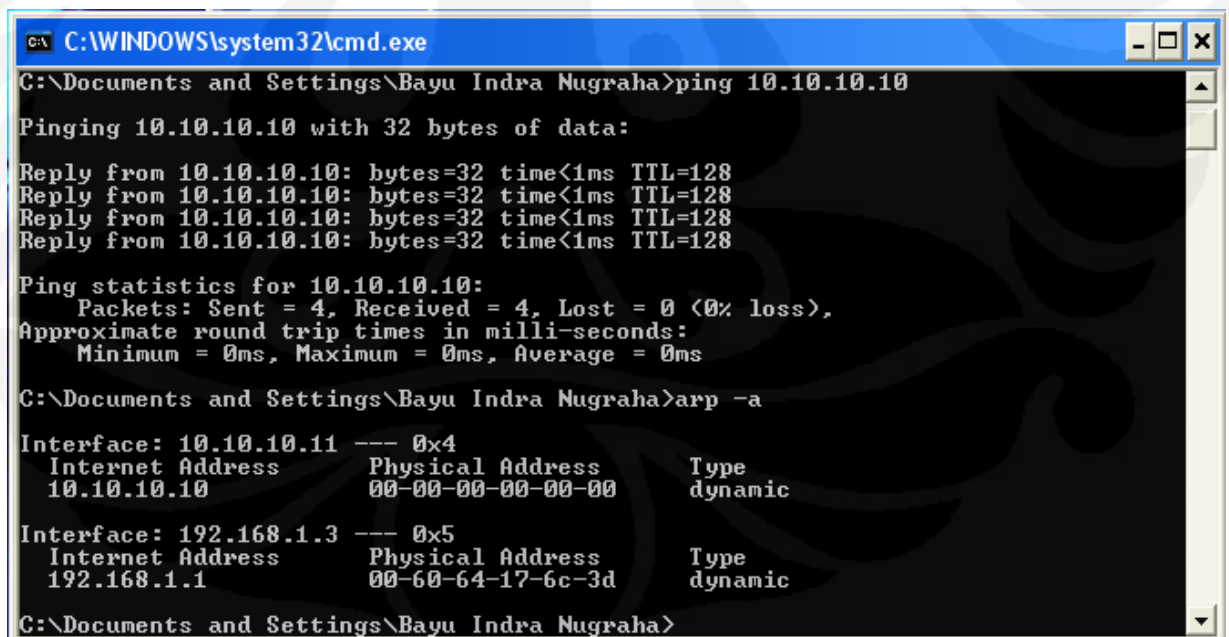
It means if the public IP address is accessed, basically it will point to the local embedded platform IP

Figure 29 - Wide Area Network

7.2 ICMP and ARP

Ping is a command that uses ICMP Protocol to check connectivity through network. It is important to check connectivity between a host and embedded platform before performing any other task. Note that if the ping command shows no connectivity, it simply means the packets cannot be delivered. The **Address Resolution Protocol (ARP)** is the method for finding a host's link layer (hardware) address when only its IP address or some other network layer address is known. So it simply means MAC address of the hardware can be detected.

In order to test the system, we can examine by using **ping** and **arp-a** command in Command Prompt Window. Figure 29 shows the connection between host and embedded platform was successful. However, the ARP shows that the MAC address of the embedded platform is 00:00:00:00:00:00 due to the combination error between FreeRTOS and uIPstack. As long student could just broadcast the packet, the packet will still delivered, but that will also clog our network with unnecessary broadcasts.



```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Bayu Indra Nugraha>ping 10.10.10.10

Pinging 10.10.10.10 with 32 bytes of data:

Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.10.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Bayu Indra Nugraha>arp -a

Interface: 10.10.10.11 --- 0x4
    Internet Address      Physical Address      Type
    10.10.10.10          00-00-00-00-00-00    dynamic

Interface: 192.168.1.3 --- 0x5
    Internet Address      Physical Address      Type
    192.168.1.1          00-60-64-17-6c-3d    dynamic

C:\Documents and Settings\Bayu Indra Nugraha>

```

Figure 30 - ICMP and ARP Result

7.3 Web Server

Web Server Page – RTOS Stats

In this web server page, student put 3 statistics that represent the LED Output Status, Joystick Input status and Task statistic. Those statistics will be updated every 2 seconds instead of 1 second to minimize the network load. LED Output Status and Joystick Input Status will read and receive the Data Status Register from particular I/O peripheral bit. And, Task statistic is read from FreeRTOS's function called `vTasklist()` that show the FreeRTOS tasks that executed in running state.

Final Year Project - Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha - n6420125

[RTOS Stats](#) | [TCP Stats](#) | [Connections](#) | [I/O](#)

LED Output Status

LED DS1, LED DS2, LED DS3, LED DS4,

Joystick Input Status

West, North, East, South, Push

Task statistics

Page will refresh every 2 seconds.

Task	State	Priority	Stack	#
uIP	R	2	299	0
QConsB6	R	0	84	6
QProdB5	R	0	84	5
GenQ	R	0	86	9
CNT_INC	R	0	102	17
MuLow	R	0	90	10
SUSP_RX	R	0	91	21
IDLE	R	0	103	22
QProdB2	R	0	93	2
QProdB3	R	0	93	3
PeekL	B	0	88	13
SUSP_TX	B	0	102	20
C_CTRL	B	0	99	19

Figure 31 - Web Server - RTOS Page Stats

Web Server – TCP stats

This page shows the uIP statistic about the network performances that defines in `uip_stats()` in default. It determines the IP, ICMP, and TCP packets status from Ethernet traffic network.

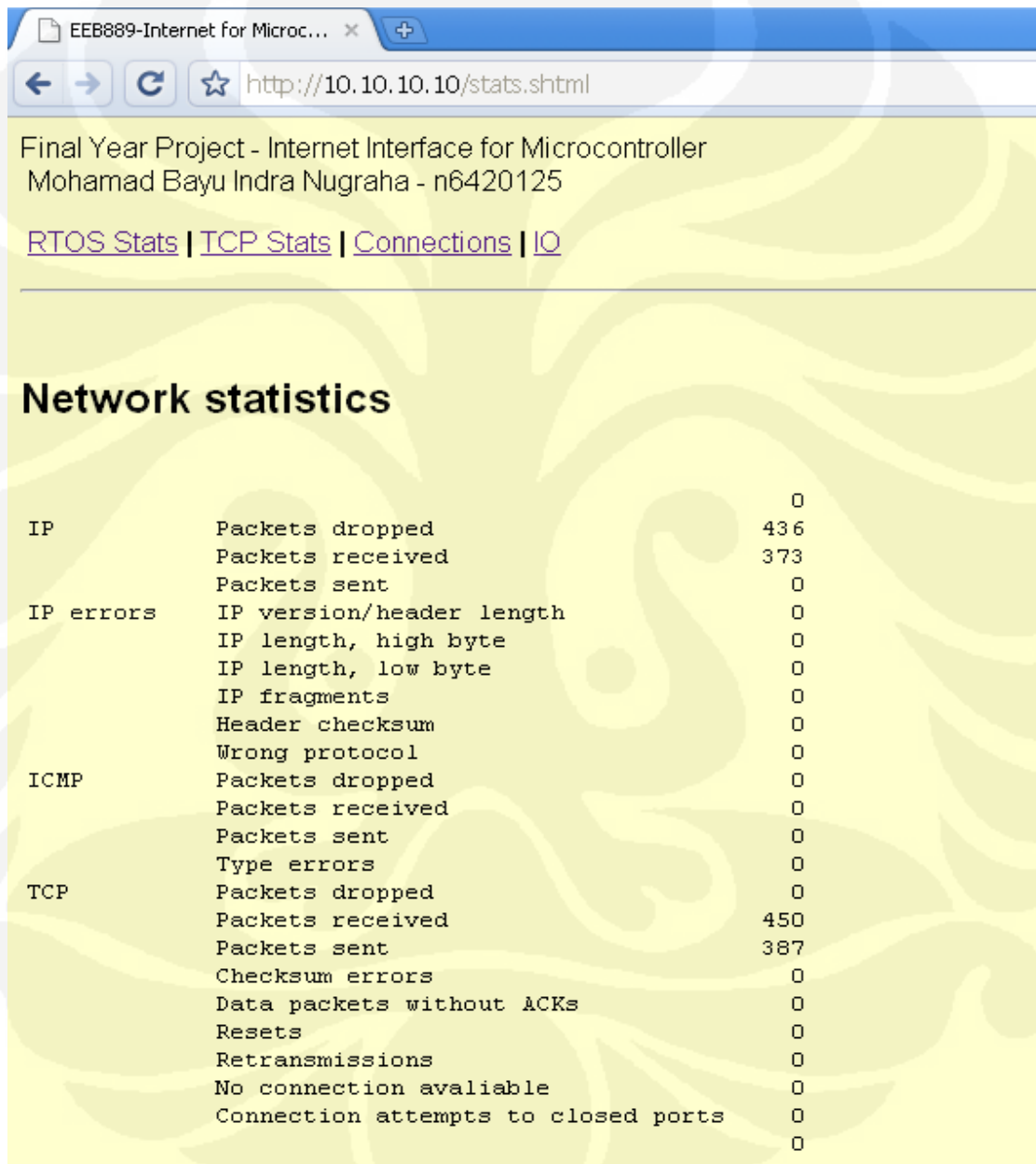


Figure 32 - Web Server - TCP Stats

Web Server – Connection Page

This page shows the embedded platform's network traffic and service discovery. It determines the statistic log of activity in Ethernet-based network.

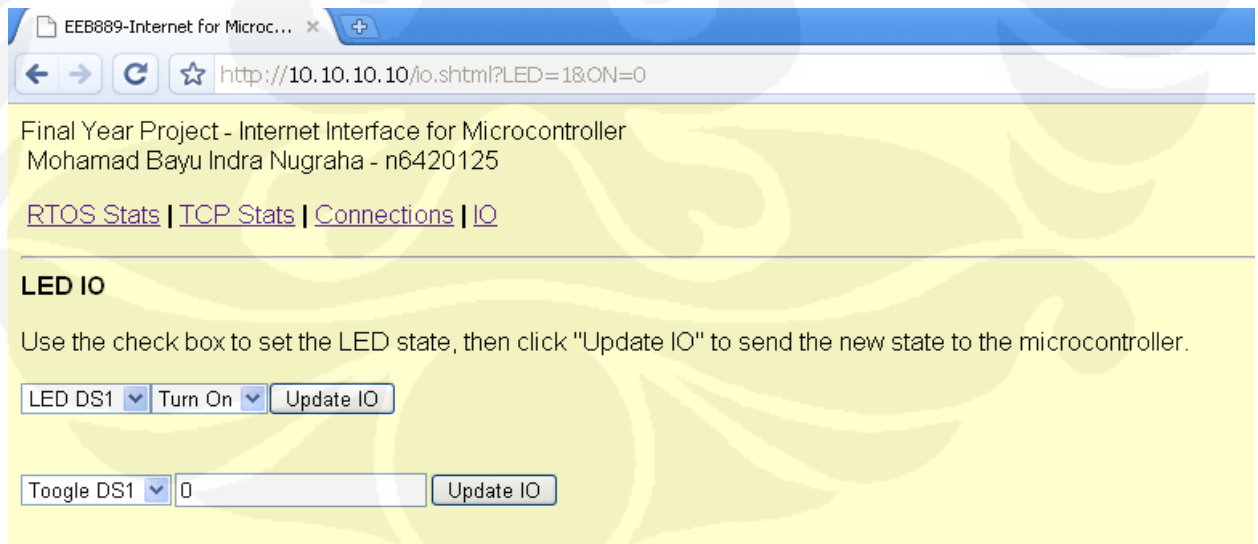


Local	Remote	State	Retransmissions	Timer Flags
80	10.10.10.11:1799	TIME-WAIT	0	9
80	10.10.10.11:1802	TIME-WAIT	0	1
80	10.10.10.11:1806	ESTABLISHED	0	4

Figure 33 - Web Server - Connection Page

Web Server – IO Page

This page shows the GUI Interface that able to send buffers into the embedded platform by clicking the „update IO’ options. Student built two GUI interfaces to send bytes to turn on/off and toggle LED. The LED status can be seen in RTOS stats page.



LED IO

Use the check box to set the LED state, then click "Update IO" to send the new state to the microcontroller.

LED DS1 ▼ Turn On ▼ Update IO

Toogle DS1 ▼ 0 Update IO

Figure 34 - Web Server - IO Page

Summaries and Conclusion

Atmel's AT91SAM7X256 is a member of a series of highly integrated flash microcontrollers based on the 32-bit ARM RISC processor including an 802.3 Ethernet MAC. Therefore we can design and program the platform to be accessed through a web server that can be used to control, transmit, and receive data to the Input Output peripherals at the board remotely. Evaluation of EMAC, Davicom DM9161A chip, and PIO Controller are needed to be synchronized with real time operating systems to ARM processor.

Transmission Control Protocol that referred at transport layer in TCP/IP reference model is used in this implementation. It provides reliable end-to-end delivery service including data transmission and flow control. Using the reliable service, there must not any data loss and the frame has to be reassembled in the right places and makes up for Internet Protocol's (IP) deficiencies.

GCC environment is decided to use in this project, therefore student use 2 GCC open-sources with C language as a basic demo application. There are:

1. FreeRTOS, a mini Real Time Kernel
2. uIP stack, TCP/IP stack that provides TCP/IP connectivity

FreeRTOS is used to implement a scale-able real time kernel that designed specifically for small embedded systems. It means that routine and some modules in the program implementation are based on this open-source. There are some advantages using FreeRTOS open-source are:

- Preemptive, cooperative and hybrid configuration options.
- Designed to be small, simple and easy to use.
- Very portable code structure predominantly written in C.
- Support both tasks and co-routines.
- Stack-overflow detection options.

The uIP stack that student used for TCP/IP stack is an open source that is intended to make it possible to communicate using TCP/IP protocol suite even on small 8-bit micro-controllers. uIP requires a few functions to be implemented specifically for the architecture to run based on Transmission Control Protocol. C language implementations are given as part of the uIP distribution.

As the result, Peripheral I/O can be controlled remotely from the web server by combining FreeRTOS and uIP stack. Furthermore, via testing, this web server worked stably. Data can be transmitted and received into embedded platform reliably and it is suitable for mini web LAN.

Bibliography

- [1] <http://ieeexplore.ieee.org.ezp02.library.qut.edu.au/stamp/stamp.jsp?tp=&arnumber=830384&isnumber=18005>
- [2] http://www.atmel.com/dyn/resources/prod_documents/6120s.pdf
- [3] http://www.atmel.com/dyn/resources/prod_documents/doc6120.pdf
- [4] http://www.msc-ge.com/download/atmel/pdf_arm9/SAM7XC-user_guide.pdf
- [5] <http://www.sics.se/~adam/download/uip-1.0-refman.pdf>
- [6] **Andrew S. Tanenbaum**, *Computer Networks*. Prentice Hall.2003
- [7] http://en.wikipedia.org/wiki/Manchester_code
- [8] <http://www.lincoln.edu/math/rmyrick/ComputerNetworks/InetReference/83.htm>
- [9] QUT Blackboard – ENB346 Digital Communication
- [10] QUT Blackboard – ENB241 Software System Design
- [11] QUT Blackboard – ENB244 Microprocessor and Digital System
- [12] <http://abstract.cs.washington.edu/~shwetak/classes/ee472/assignments/lab2/timers.pdf>
- [13] http://www.ethernut.de/nutwiki/AT91SAM7X-EK_Port_I/O
- [14] <http://www.mynetwatchman.com/pckidiot/arp.htm>

Appendix

HTML SCRIPT

Final Year Project Code\Demonstration\webserver\httpd-fsdata.c

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

```

404.html

```

<html>
  <body bgcolor="white">
    <center>
      <h1>404 - file not found</h1>
      <h3>Go <a href="/">here</a> instead.</h3>
    </center>
  </body>
</html>

```

index.html

```

<html>
  <head>
    <title>EEB889-Internet Interface for Microcontroller</title>
  </head>
  <BODY
onLoad="window.setTimeout(&quot;location.href='index.shtml'&quot;;,10
00)"bgcolor="#FFFFCC">
  <font face="arial">
Final Year Project <br>
Internet Interface for Microcontroller <br>
Mohamad Bayu Indra Nugraha - n6420125 <br><p>
Loading index.shtml. Click <a href="index.shtml">here</a> if not
automatically redirected.
  </font>
</body>
</html>

```

index.shtml

```

<html>
  <head>
    <title>EEB889-Internet Interface for Microcontroller</title>

```

```

</head>
<BODY
onLoad="window.setTimeout (&quot;location.href='index.shtml'&quot;;,10
00)"bgcolor="#FFFFCC">
<font face="arial">
Final Year Project <br>
Internet Interface for Microcontroller <br>
Mohamad Bayu Indra Nugraha - n6420125 <br><p>
<a href="index.shtml">RTOS Stats</a> <b>|</b> <a
href="stats.shtml">TCP Stats</a> <b>|</b> <a
href="tcp.shtml">Connections</a> <b>|</b> <a href="io.shtml">IO</a>
<br><p>
<hr>
<br><p>
<h2>LED Output Status</h2>
%! led-io
<h2>Joystick Input Status</h2>
*% joystick-status
<h2>Task statistics</h2>
Page will refresh evey 2 seconds.<p>
<font face="courier"><pre>Task          State Priority Stack
#<br>*****<br>
%! rtos-stats
</pre></font>
</font>
</body>
</html>

```

Stats.shtml

```

<html>
<head>
<title>EEB889-Internet Interface for Microcontroller</title>
</head>
<BODY bgcolor="#FFFFCC">
<font face="arial">
Final Year Project <br>
Internet Interface for Microcontroller <br>
Mohamad Bayu Indra Nugraha - n6420125 <br><p>
<a href="index.shtml">RTOS Stats</a> <b>|</b> <a
href="stats.shtml">TCP Stats</a> <b>|</b> <a
href="tcp.shtml">Connections</a> <b>|</b> <a href="io.shtml">IO</a>
<br><p>
<hr>
<br><p>
<h2>Network statistics</h2>
<table width="300" border="0">
<tr><td align="left"><font face="courier"><pre>
IP          Packets dropped
            Packets received
            Packets sent
IP errors   IP version/header length
            IP length, high byte

```



```

IP length, low byte
IP fragments
Header checksum
Wrong protocol
ICMP   Packets dropped
        Packets received
        Packets sent
        Type errors
TCP    Packets dropped
        Packets received
        Packets sent
        Checksum errors
        Data packets without ACKs
        Resets
        Retransmissions
        No connection available
        Connection attempts to closed ports

```

```
</pre></font></td><td><pre>%! net-stats
</pre></table>
</font>
</body>
</html>
```

Tcp.shtml

```

<html>
  <head>
    <title>EEB889-Internet Interface for Microcontroller</title>
  </head>
  <BODY bgcolor="#FFFFCC">
    <font face="arial">
      Final Year Project - Internet Interface for Microcontroller <br>
      Mohamad Bayu Indra Nugraha - n6420125 <br><p>
      <a href="index.shtml">RTOS Stats</a> <b>|</b> <a
      href="stats.shtml">TCP Stats</a> <b>|</b> <a
      href="tcp.shtml">Connections</a> <b>|</b> <a href="io.shtml">IO</a>
      <br><p>
      <hr>
      <br>
      <h2>Network connections</h2>
      <p>
      <table>
        <tr><th>Local</th><th>Remote</th><th>State</th><th>Retransmissions</
        th><th>Timer</th><th>Flags</th></tr>
        %! tcp-connections
      </pre></font>
    </body>
  </html>
```

Io.shtml

```
<html>
```

```
<head>
  <title>EEB889-Internet Interface for Microcontroller</title>
</head>
<BODY bgcolor="#FFFFCC">
<font face="arial">
Final Year Project <br>
Internet Interface for Microcontroller <br>
Mohamad Bayu Indra Nugraha - n6420125 <br><p>
<a href="index.shtml">RTOS Stats</a> <b>|</b> <a
href="stats.shtml">TCP Stats</a> <b>|</b> <a
href="tcp.shtml">Connections</a> <b>|</b> <a href="io.shtml">IO</a>
<br><p>
<hr>
<b>LED IO</b><br>
<p>
Use the check box to set the LED state, then click "Update IO" to
send the new state to the microcontroller.
<p>
<form name="aForm" action="/io.shtml" method="get">
%! led-io
<p>
<input type="submit" value="Update IO">
</form>
<br><p>
</font>
</body>
</html>
```

GUI Interface

Final Year Project Code\Demonstration\webserver\httpd-cgi.c

Network Stats

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

static unsigned short
generate_tcp_stats(void *arg)
{
    struct uip_conn *conn;
    struct httpd_state *s = (struct httpd_state *)arg;

    conn = &uip_conns[s->count];
    return snprintf((char *)uip_appdata, UIP_APPDATA_SIZE,

"<tr><td>%d</td><td>%u.%u.%u.%u:%u</td><td>%s</td><td>%u</td><td>%u<
/td><td>%c %c</td></tr>\r\n",
        htons(conn->lport),
        htons(conn->ripaddr[0]) >> 8,
        htons(conn->ripaddr[0]) & 0xff,
        htons(conn->ripaddr[1]) >> 8,
        htons(conn->ripaddr[1]) & 0xff,
        htons(conn->rport),
        states[conn->tcpstateflags & UIP_TS_MASK],
        conn->nrtx,
        conn->timer,
        (uip_outstanding(conn)) ? '*:*:* ',
        (uip_stopped(conn)) ? '!:*:* ');
}
/*-----*/
-----*/
static
PT_THREAD(tcp_stats(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);

    for(s->count = 0; s->count < UIP_CONNS; ++s->count) {
        if((uip_conns[s->count].tcpstateflags & UIP_TS_MASK) !=
UIP_CLOSED) {
            PSOCK_GENERATOR_SEND(&s->sout, generate_tcp_stats, s);
        }
    }

    PSOCK_END(&s->sout);
}

```

```
}
```

uIP Stats – IP TCP ICMP Checksum

```
char *pcStatus4,*pcStatus3,*pcStatus2,*pcStatus1;
unsigned long ulString;

/*-----*/
-----*/
static unsigned short
generate_net_stats(void *arg)
{
    struct httpd_state *s = (struct httpd_state *)arg;
    return sprintf((char *)uip_appdata, UIP_APPDATA_SIZE,
                  "%5u\n", ((uip_stats_t *)&uip_stat)[s->count]);
}

static
PT_THREAD(net_stats(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);

    #if UIP_STATISTICS

        for(s->count = 0; s->count < sizeof(uip_stat) /
        sizeof(uip_stats_t);
            ++s->count) {
            PSOCK_GENERATOR_SEND(&s->sout, generate_net_stats, s);
        }

    #endif /* UIP_STATISTICS */

    PSOCK_END(&s->sout);
}
/*-----*/
-----*/
```

FreeRTOS Stats

```
extern void vTaskList( signed char *pcWriteBuffer );
static char cCountBuf[ 32 ];
long lRefreshCount = 0;
static unsigned short
generate_rtos_stats(void *arg)
```

```

{
    lRefreshCount++;
    sprintf( cCountBuf, "<p><br>Refresh count = %ld",
lRefreshCount );
    vTaskList( uip_appdata );
    strcat( uip_appdata, cCountBuf );

    return strlen( uip_appdata );
}
/*-----*/
-----*/

static
PT_THREAD(rtos_stats(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_rtos_stats, NULL);
    PSOCK_END(&s->sout);
}
/*-----*/
-----*/

```

Input User Interface – LED Turn on/Off

```

char *Input1ON, *Input1OFF;
generate_input(void *arg)
{
    sprintf( uip_appdata,
        "<select name=\"LED\"><option value=\"0\">LED
DS1</option><option value=\"1\">LED DS2</option><option
value=\"2\">LED DS3</option><option value=\"3\">LED
DS4</option></select><select name=\"ON\"><option value=\"0\">Turn
On</option><option value=\"1\">Turn Off</option></select>", Input1ON,
Input1OFF);

    return strlen( uip_appdata );
}

static
PT_THREAD(led_input(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_input, NULL);
    PSOCK_END(&s->sout);
}

```

Input User Interface – Toggle LED

```

generate_toggle(void *arg)
{
    sprintf( uip_apdata,
            "<select name=\"Tog\"><option value=\"0\">Toggle
DS1</option><option value=\"1\">Toggle DS2</option><option
value=\"2\">Toggle DS3</option><option value=\"3\">Toggle
DS4</option></select><input type=\"textarea\" name=\"De\"
value=\"0\" %s>", Input1ON, Input1OFF);

    return strlen( uip_apdata );
}

static
PT_THREAD(led_toggle(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_toggle, NULL);
    PSOCK_END(&s->sout);
}

```

LED Stats

```

char *pcStatus4,*pcStatus3,*pcStatus2,*pcStatus1;
//extern unsigned long uxParTextGetLED( unsigned long uxLED );

static unsigned short generate_io_state( void *arg )
{
    if( GetLED(4) ){
        pcStatus4 = "checked";
    }
    else{
        pcStatus4 = "";
    }
    if( GetLED(3) ){
        pcStatus3 = "checked";
    }
    else{
        pcStatus3 = "";
    }
    if( GetLED(2) ){
        pcStatus2 = "checked";
    }
    else{

```

```

        pcStatus2 = "";
    }
    if(GetLED(1) ){
        pcStatus1 = "checked";
    }
    else{
        pcStatus1 = "";
    }

    sprintf( uip_appdata,
        "<input type=\"checkbox\" name=\"LED1\" value=\"1\"
%s>LED DS1, <input type=\"checkbox\" name=\"LED2\" value=\"1\"
%s>LED DS2,<input type=\"checkbox\" name=\"LED3\" value=\"1\" %s>LED
DS3,<input type=\"checkbox\" name=\"LED0\" value=\"1\" %s>LED DS4,
"\"
        "<p>",
        pcStatus1, pcStatus2, pcStatus3, pcStatus4 );

    return strlen( uip_appdata );
}

static PT_THREAD(led_io(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_io_state, NULL);
    PSOCK_END(&s->sout);
}

```

Joystick Stats

```

char *North ,*South,*East,*West, *Push , *InputStatus;
static unsigned short generate_joystick( void *arg )
{
    if( GetJoystick(5) ){
        Push = "checked";
    }
    else{
        Push = "";
    }
    if( GetJoystick(4) ){
        East = "checked";
    }
    else{
        East = "";
    }
    if( GetJoystick(3) ){
        West = "checked";
    }
    else{
        West = "";
    }
}

```

```

    }
    if( GetJoystick(2) ){
        South = "checked";
    }
    else{
        South = "";
    }
    if(GetJoystick(1) ){
        North= "checked";
    }
    else{
        North = "";
    }

    sprintf( uip_appdata,
        "<input type=\"checkbox\" name=\"West\" value=\"1\"
%s>West, <input type=\"checkbox\" name=\"North\" value=\"1\"
%s>North,<input type=\"checkbox\" name=\"East\" value=\"1\"
%s>East,<input type=\"checkbox\" name=\"South\" value=\"1\"
%s>South, <input type=\"checkbox\" name=\"Push\" value=\"1\"
%s>Push\"
        "<p>",
        West, North, East, South, Push );

    return strlen( uip_appdata );
}

static PT_THREAD(joystick_status(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_joystick, NULL);
    PSOCK_END(&s->sout);
}

```


Reading Web Address and Separate value we want

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

int nLEDNum;
int nLEDOn;

void vProcessInput( char *pcInput )
{
char *c,*LED, *Temp, *ON, *To, *De;
//ToggleInput LED;
/* Turn the LED on or off depending on the checkbox status. */

/*Check the address website*/
c = strstr( pcInput, "?" );
LED= strstr( pcInput, "LED=");
ON = strstr( pcInput, "ON=");
To = strstr( pcInput, "Tog=");
De = strstr( pcInput, "De=");
if( c ){
    if(LED && ON){
        nLEDNum = atoi(LED+4);
        nLEDOn = atoi(ON+3);
        SetLED(nLEDNum,nLEDOn);
    }
    if(To && De){
        nLEDNumber = atoi(To+4);
        nTickRate = atoi(De+3);
    }
}
}

```

Delay to Toggle LEDs

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

void vApplicationTickHook(void)
{
    if(nTickRate != 0){
static unsigned portLONG Count = 0, ErrorFound = pdFALSE;

/* The rate at which LEDs will toggle if an error has been found in
one or
more of the standard demo tasks. */

const unsigned portLONG ErrorFlashRate = 1000*nTickRate /
portTICK_RATE_MS;

/* The rate at which LEDs will toggle if no errors have been found
in any
of the standard demo tasks. */

const unsigned portLONG NoError = nTickRate*1000 /
portTICK_RATE_MS;
Count++;
if( ErrorFound != pdFALSE )
{
    /* We have already found an error, so flash the LED with
the appropriate
frequency. */

    if( Count > ErrorFlashRate ){
        Count = 0;
        ToggleLED( nLEDNumber);
    }
}
else
{
    if( Count > NoError )
    {
        Count = 0;
        ToggleLED( nLEDNumber );
    }
}
}
}

```

Main Program

```

/* Standard includes. */
#include <stdlib.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"

/* Demo application includes. */
#include "parallelIO.h"
#include "uip_task.h"
#include "BlockQ.h"
#include "blocktim.h"
#include "flash.h"
#include "QPeek.h"
#include "dynamic.h"
#include "httpd.h"
#include "uIP_Task.h"
#include "httpd-cgi.h"

/* Priorities for the demo application tasks. */
#define UIP_PRIORITY ( tskIDLE_PRIORITY +
2 )
#define mainUSB_PRIORITY (
tskIDLE_PRIORITY + 2 )
#define mainBLOCK_Q_PRIORITY ( tskIDLE_PRIORITY +
1 )
#define mainFLASH_PRIORITY ( tskIDLE_PRIORITY + 2 )
#define mainGEN_QUEUE_TASK_PRIORITY ( tskIDLE_PRIORITY )

/* The task allocated to the uIP task is large to account for its
use of the
sprintf() library function. Use of a cut down printf() library
would allow
the stack usage to be greatly reduced. */
#define UIP_STACK ( configMINIMAL_STACK_SIZE * 6 )

/*-----*/

/*
 * Configure the processor for use with the Atmel demo board. Setup
is minimal
 * as the low level init function (called from the startup asm file)
takes care
 * of most things.
 */
static void HardwareInit( void );

/*-----*/

```

```

/*
 * Starts all the other tasks, then starts the scheduler.
 */
int main( void )
{
    /* Setup any hardware that has not already been configured by
the low
    level init routines. */
    HardwareInit();

    /* Start the task that handles the TCP/IP and WEB server
functionality. */
    xTaskCreate( vuIP_Task, "uIP", UIP_STACK, NULL, UIP_PRIORITY,
NULL );

    /* Start the standard demo tasks. */
    vStartBlockingQueueTasks( mainBLOCK_Q_PRIORITY );
    vCreateBlockTimeTasks();
    vStartLEDFlashTasks( mainFLASH_PRIORITY );
    vStartGenericQueueTasks( mainGEN_QUEUE_TASK_PRIORITY );
    vStartQueuePeekTasks();
    vStartDynamicPriorityTasks();

    /*Start the scheduler*/
    vTaskStartScheduler();
    /* We should never get here as control is now taken by the
scheduler. */
    return 0;
}
/*-----*/

static void HardwareInit( void )
{
    portDISABLE_INTERRUPTS();

    /* When using the JTAG debugger the hardware is not always
initialised to
    the correct default state. This line just ensures that this
does not
    cause all interrupts to be masked at the start. */
    AT91C_BASE_AIC->AIC_EOICR = 0;

    /* Most setup is performed by the low level init function
called from the
    startup asm file. */

    /* Enable the peripheral clock. */
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOA;
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOB;
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_EMAC;

    /* Initialise the LED outputs for use by application tasks.*/

```

```

    LEDInitialise();
    JoystickInitialise();
}

```

Parallel IO Controller

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

/* Scheduler includes. */
#include "FreeRTOS.h"

/* Demo application includes. */
#include "parallelIO.h"

/*-----
 * Simple parallel port IO routines for the LED's. LED's can be
 * set, cleared
 * or toggled.
 *-----*/

/* Joystick inputs used*/
#define North ( 1 << 21 ) /* PA21 */
#define South ( 1 << 22 ) /* PA22 */
#define East ( 1 << 23 ) /* PA23 */
#define West ( 1 << 24 ) /* PA24 */
#define Push ( 1 << 25 ) /* PA25 */
/*LED Outputs used*/
#define DS1 ( 1 << 19 ) /* PB19 */
#define DS2 ( 1 << 20 ) /* PB20 */
#define DS3 ( 1 << 21 ) /* PB21 */
#define DS4 ( 1 << 22 ) /* PB22 */

/*Put The I/O Parts into an Array to make me writing program
easier*/
#define LEDs ( nLED_Mask[ 0 ] | nLED_Mask[ 1 ] |
nLED_Mask[ 2 ] | nLED_Mask[ 3 ] )
#define Joystick ( nJoy_Mask[ 0 ] | nJoy_Mask[ 1 ] | nJoy_Mask[ 2 ]
| nJoy_Mask[ 3 ] | nJoy_Mask[ 4 ] )

/*Pointing each array to the particular bit*/
const unsigned portLONG nLED_Mask[ 4 ]= { DS1, DS2, DS3, DS4
};

```

```

const unsigned portLONG      nJoy_Mask[ 5 ] = { North, South, East,
West, Push };

/* LED Initialization */
void LEDInitialise( void )
{
    /* Configure the PIO Lines corresponding to LED1 to LED4 to be
outputs. */
    AT91C_BASE_PIOB->PIO_PER = LEDs; // Enable The PIOB Control
    AT91C_BASE_PIOB->PIO_OER = LEDs; // Output Enable Register
    /* Start with all LED's off. */
    AT91C_BASE_PIOB->PIO_SODR = LEDs; // Set Output Enable Register
}

/* Joystick Initialization */
void JoystickInitialise( void )
{
    /*Configure the PIO Lines corresponding to Joystick to be
inputs*/
    AT91C_BASE_PIOA->PIO_PER = Joystick; // Enable The PIOA
Control
    AT91C_BASE_PIOA->PIO_ODR = Joystick; // Output Disable
Register to closed the Joystick contact
    AT91C_BASE_PIOA->PIO_PPUER = Joystick; // Enabling Intenal
pull-up resistor
}

void SetLED( unsigned portBASE_TYPE nLED, signed portBASE_TYPE Value
)
{
    /*Check if LED number is not bigger than 4*/
    if( nLED < ( portBASE_TYPE ) 4 )
    {
        /*if Value is not equal to zero, it will turn off the
LED*/
        /*else it will turn on the LED*/
        if( Value ){
            AT91C_BASE_PIOB->PIO_SODR = nLED_Mask[ nLED ];
        }
        else{
            AT91C_BASE_PIOB->PIO_CODR = nLED_Mask[ nLED ];
        }
    }
}
/*-----*/

void ToggleLED( unsigned portBASE_TYPE nLED )
{
    /*Check if LED number is not bigger than 4*/
    if( nLED < ( portBASE_TYPE ) 4 ){
        if( AT91C_BASE_PIOB->PIO_PDSR & nLED_Mask[ nLED ] ){
            AT91C_BASE_PIOB->PIO_CODR = nLED_Mask[ nLED ];
        }
    }
}

```

```

        else{
            AT91C_BASE_PIOB->PIO_SODR = nLED_Mask[ nLED ];
        }
    }
}
/*-----*/

/*This Function is to check the LED Status*/
unsigned portBASE_TYPE GetLED(int nLED)
{
    return !( AT91C_BASE_PIOB->PIO_PDSR & nLED_Mask[ nLED -
1 ] );
}
/*This Function is to check the Joystick Input Status*/
unsigned portBASE_TYPE GetJoystick(int nJoystick )
{
    return !(AT91C_BASE_PIOA->PIO_PDSR & nJoy_Mask [
nJoystick - 1]);
}

void ControlledfromJoystick (void)
{
    /*if North is pressed, then LED DS2 will turn on*/
    if(GetJoystick(1)){
        SetLED(1,0);
    }
    /*if South is pressed, then LED DS4 will turn on*/
    else if(GetJoystick(2)){
        SetLED(3,0);
    }
    /*if West is pressed, then LED DS1 will turn on*/
    else if(GetJoystick(3)){
        SetLED(0,0);
    }
    /*if East is pressed, then LED DS3 will turn on*/
    else if(GetJoystick(4)){
        SetLED(2,0);
    }
    /*if Push is pressed, then All LEDs will turn off*/
    else if(GetJoystick(5)){
        SetLED(0,1);
        SetLED(1,1);
        SetLED(2,1);
        SetLED(3,1);
    }
}
}

```

Abstract

At present, network is becoming the hot point for the investigation of embedded system. Considering the growth of data communication, connection between embedded system platforms and the internet interfaces has been an important development direction and indispensable functions for the embedded system in the future and it becomes an important role if the embedded platforms can be accessible and monitored whenever and wherever we need.

By implementing TCP/IP uIP-stack open source properties and correlative system interfaces architecture, some internet protocol application such as web server, ICMP and telnet server, can be integrated into the embedded systems. This paper describes how the combination between Real Time Operating System and Embedded Web Server Application can be established in ATMEL AT91SAM7X platform by sending multiple packet data and processed stably in I/O hardware architectures. At last, some real world simulations are applied in order to test system design performances and reliability.

Chapter 1- Introduction

1.1 Project Background

As the World Wide Web (or Web) continues to evolve, it is clear that its underlying technologies are useful for much more than just browsing the Web. Web browsers have become the standard user interface for a variety of applications because Web browsers can provide a GUI interface to various client/server applications without having to implement a separate client.

General Web server, which were developed for general purpose computers such as NT servers or Unix workstations, typically require megabytes of memory, a fast processor, a pre-emptive multitasking operating system, and other resources. A web server can be embedded in a device to provide remote access to the device from a Web browser if the resource requirements of the Web server are reduced. The result typically a portable set of code that can run on embedded system with limited resources. [1]

Embedded Web Server are used to convey the state information of embedded systems, such as a systems working statistic, operation result and transfer user commands from a Web browser to an embedded system. The state information is extracted from an embedded system application and the control command is implemented through the embedded system application.

Atmel's AT91SAM7X256 ARM7 Based is the hardware that student used to implement embedded Web Server because it contains a large set of peripherals, including an 802.3 Ethernet MAC. So, by combining the ARM processor with on-chip Flash and SRAM, and a wide range of peripheral function, including USART, SPI, CAN Controller on it, it become cost-effective solution to many embedded control application in real world requiring communication over internet, for example, CAN wired and Zigbee wireless network. [2]

In order to establish communication between the hardware and each other across a network, we need a protocol suitable between those as a convention or standard that enables the connection. TCP/IP protocol suite has become a global standard protocol communication used for web page transfers, e-mail transmission, file transfer, and peer-to-peer networking over Internet. Traditional TCP/IP implementations have required too much resources of code size and memory usage for 8 or 16-bit systems. To solve this problem, student used open-source uIP implementation that is designed to have only absolute minimal set of features needed for a full TCP/IP stack. uIP implementation can only handle a single network interface and contain IP, ICMP, UDP and TCP Protocol. [4]

Real Time Operating System management also will be discussed in this report. The purpose is to allow user to do multiple tasks to a single processor attached at the hardware simultaneously without the system becoming unresponsive. The scheduling algorithm is used to finish and complete real-time function within a given time without any failure in the system. As a result, the combination of establishing data communication via Ethernet and RTOS implementation in Atmel's AT91SAM7X256 ARM7 board will be discussed in this report.

C languages will be used in GCC platform by the student. Even though, there is a software development tools for embedded systems called IAR systems that should be easier to be managed, it has some limitations that will be explained later. GCC is the leading free (open source) compiler environment, widely used in the industry. Though, it's really hard to be implemented and waste of time, there are no issues of confidentiality and limitations for sharing information in the joint research. FreeRTOS and uIP TCP/IP Stack open sources will be combined by the student as a mini Real Time Kernel routine and TCP/IP protocol stack respectively.

1.2 Project Aim, Objectives and Plan of Development

The fundamental aim of this project is to design and program AT91SAM7X ARM Based for networking purposes. As the result, the embedded platform can be accessed

through a web server that can be used to control, transmit, and receive data to the Input Output peripherals remotely.

The project has four primary goals:

1. Analyzing the hardware characteristic and functional descriptions of AT91SAM7X256 as embedded platform chip and DM9161A as power transceiver for Ethernet.
2. Observing Real Time Operating System Algorithm that can be implemented into a single processor.
3. Investigating uIP TCP/IP stack to build several TCP applications in small resources. Transmitting and Receiving data will be tested.
4. Controlling Input Output Peripherals in the hardware. User LED and Joystick will be involved.

The remainder of this report is organized as follows. Chapter 1 is the introduction of the report. Chapter 2 briefly describes the hardware characteristic of Atmel AT91SAM7X256 as an embedded platform and its functional description. Chapter 3 will explain about the TCP/IP Protocol fundamental. Chapter 4 and 5 will discuss about Real Time Operating System that will be set up by using FreeRTOS open source and TCP/IP Stack respectively. The Embedded Server implementation and algorithm design explanation will be in Chapter 6 and the Software demonstration will be describe in Chapter 7.

Chapter 2 - Hardware Characteristic

2.1 Atmel's AT91SAM7X256

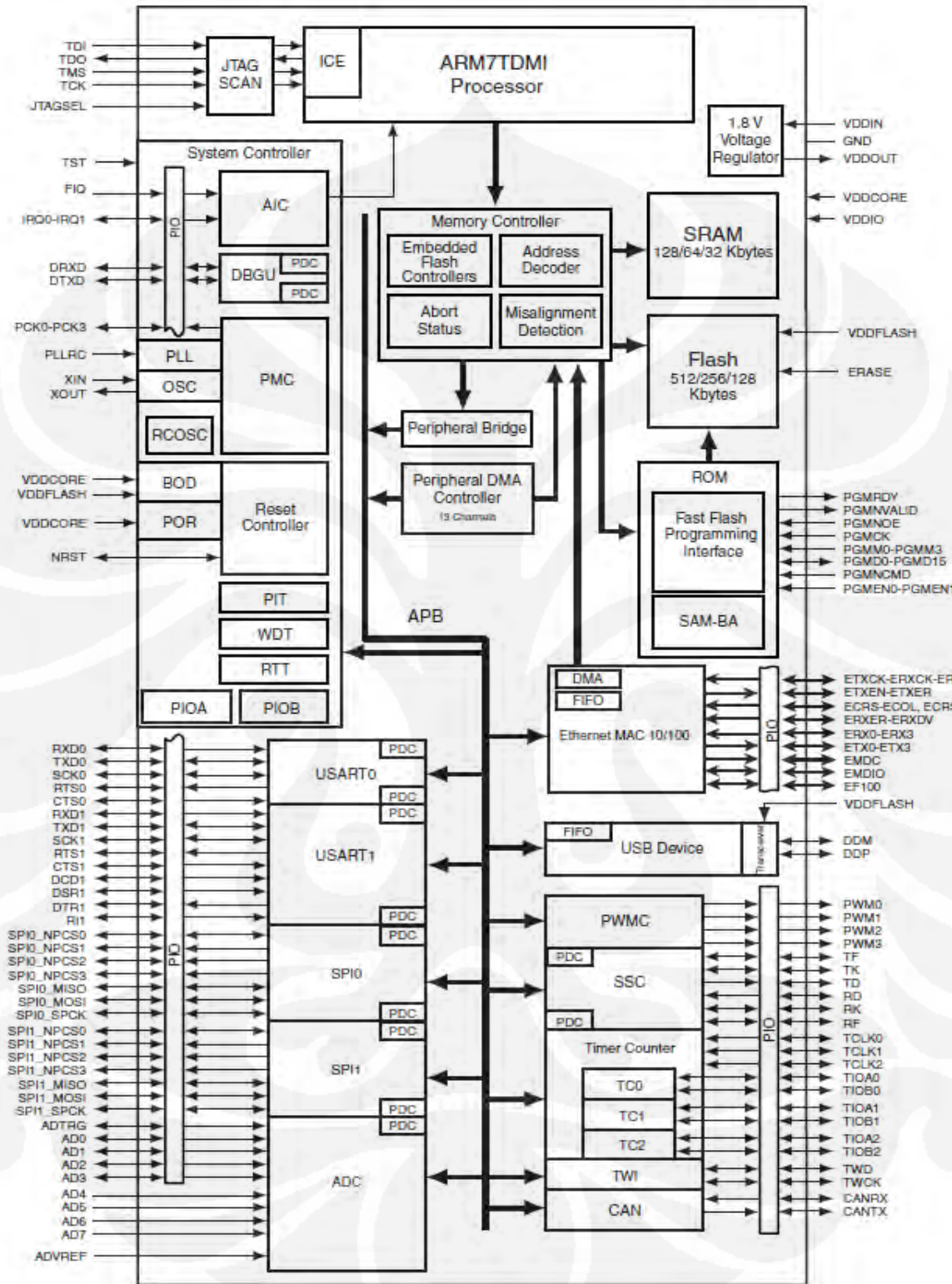
Atmel's AT91SAM7X256 is a member of a series of highly integrated flash microcontrollers based on the 32-bit ARM RISC processor. It features 256 Kbyte high-speed Flash and 64 Kbyte SRAM, a large set of peripherals, including an 802.3 Ethernet MAC and CAN controller, USART, SPI. [2]

The embedded Flash memory can be programmed by downloading the source code via JTAG-ICE interface or via parallel interface on a production programmer prior to mounting. Built-in lock bits and a security bit protect the firmware from accidental overwrite and preserve its confidentiality.

As we can see, to fulfill the requirement of this project, student needs to evaluate the hardware characteristic of EMAC, Davicom DM9161A chip, and PIO Controller to be synchronized with real time operating systems to ARM processor.



Figure 1 - AT91SAM7X-EK Hardware



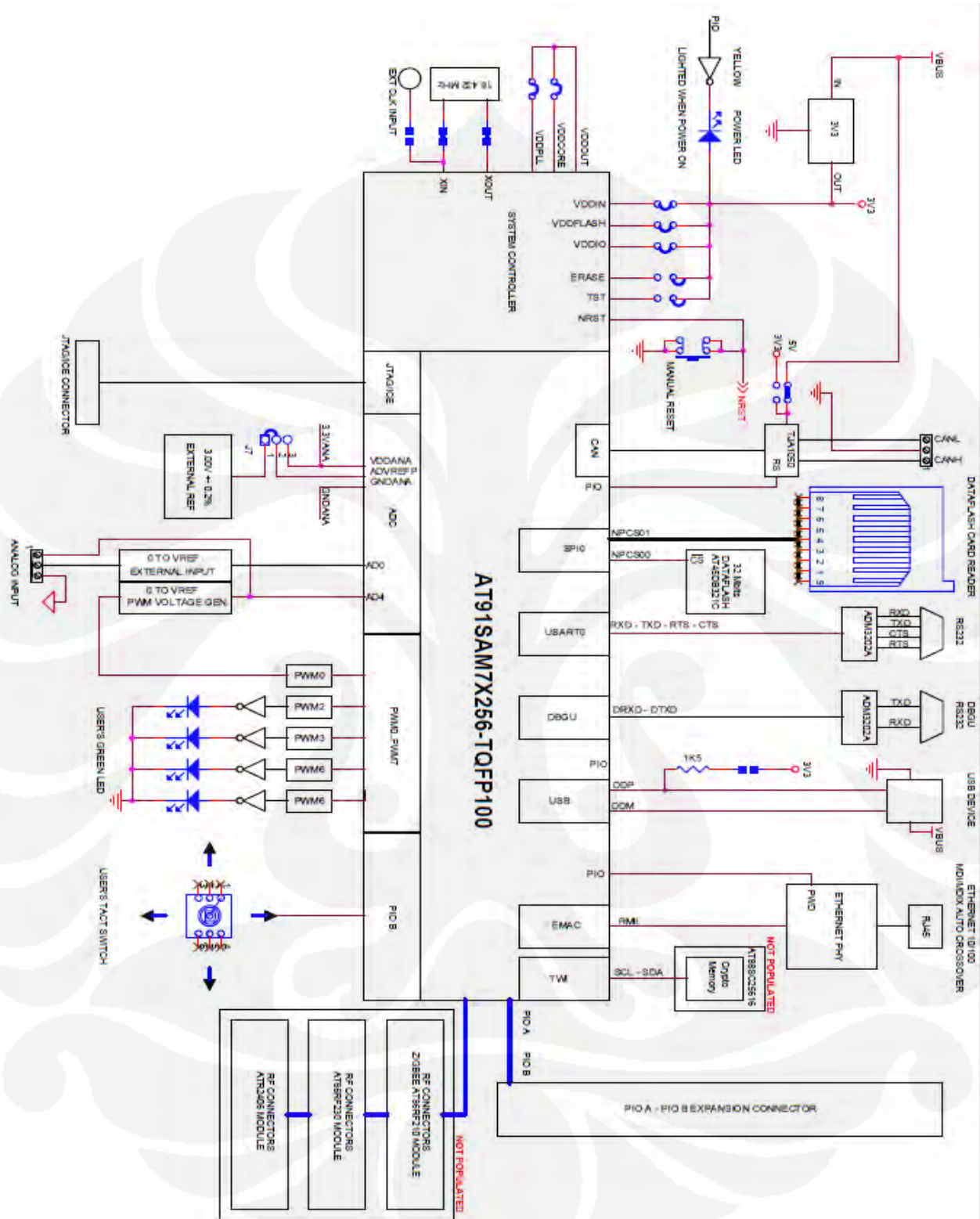


Figure 2 - AT91SAM7X256 Block Diagram

2.2 Ethernet MAC

The EMAC module implements a 10/100 Ethernet MAC compatible with the IEEE 802.3 standard using an address checker, statistic and control register, receive and transmit blocks, and a DMA interface.

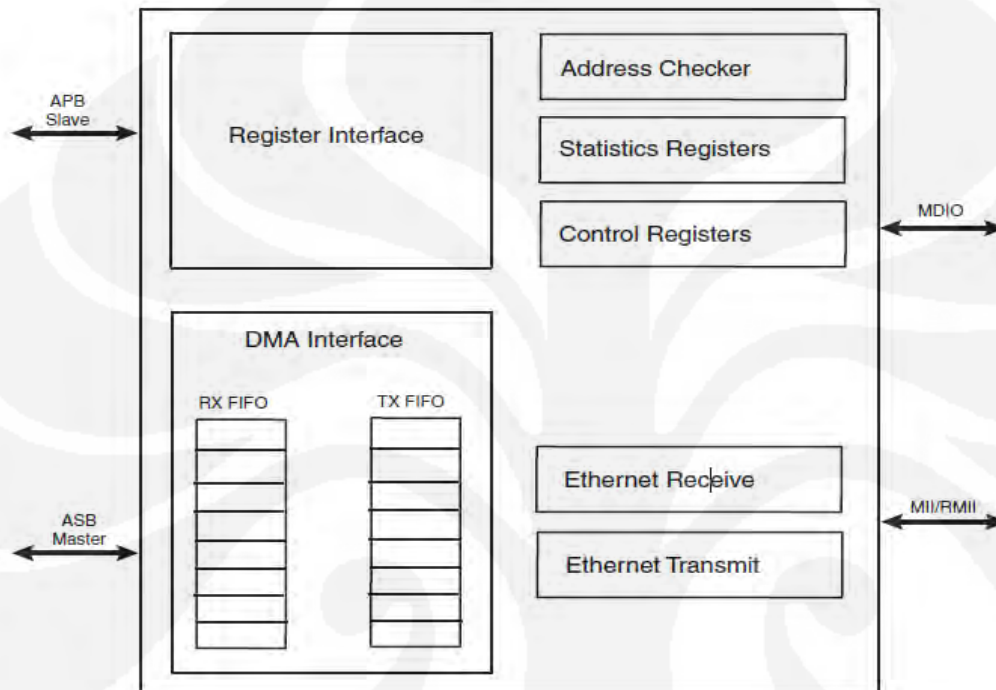


Figure 3 - EMAC Block Diagram

As we can see from figure 2, the explanation the functional description should be like student mention below:

- **Address Checker.** It will recognize four specific 48-bit addresses and contains a 64-bit hash register to match between multicast and unicast addresses, and then copy all frames to the memory and act on an external address match signal.
- **Statistic Register.** It contains registers for counting various types of event associated with transmit and receive operations, for example network management statistics.
- **Control Register.** It setups up DMA activity, start frame transmission and select modes of operation such as full or half duplex.
- **Receive Block.** It checks for a valid preamble, FCS, alignment and length, and presents received frames to the address checking block and DMA interface.

- **Transmit Block.** It takes data from the DMA interface, adds preamble end, pad and FCS and transmits data according to the CSMA/CD (Carrier Sense Multiple Access with Collision Detection). The start of transmission is deferred if CRS (Carrier Sense) is active. If the system is in full duplex mode, the Carrier Sense and Collision have no effect because full duplex because transmitting and receiving path are split into 2 channels.
- **DMA Interface.** The DMA block connects to external memory through its ASB bus interface and contains of Transmit and Receive FIFOs for buffering frame data. It loads the transmit FIFO and empties the receive FIFO. Receive data will not be sent to the memory until the address checker has determined that the frame should be copied. The length of Receive buffers is 128 bytes and Transmit buffers range in length between 0 and 2047 bytes. As summaries, DMA block manages transmit and receive frame buffer queues and can hold multiple frames.

2.3 DM9161A – 10/100 Mbps Fast Ethernet Physical Layer Single Chip Transceiver

The DM9161A Fast Ethernet single chip transceiver, providing the functionality as specified in IEEE 802.3u, integrates a complete 100 Base-TX module with Unshielded Twisted Pair Category 5 Cable (UTP5) and a complete 10 Base-T Module with UTP5/UTP3. Through the Media Independent Interface (MII), it can be connected to the Medium Access Control (MAC) layer. Figure 4 shows the major functional blocks implemented in the DM9161A chip and figure 5 describes the complete tasks how each block works.

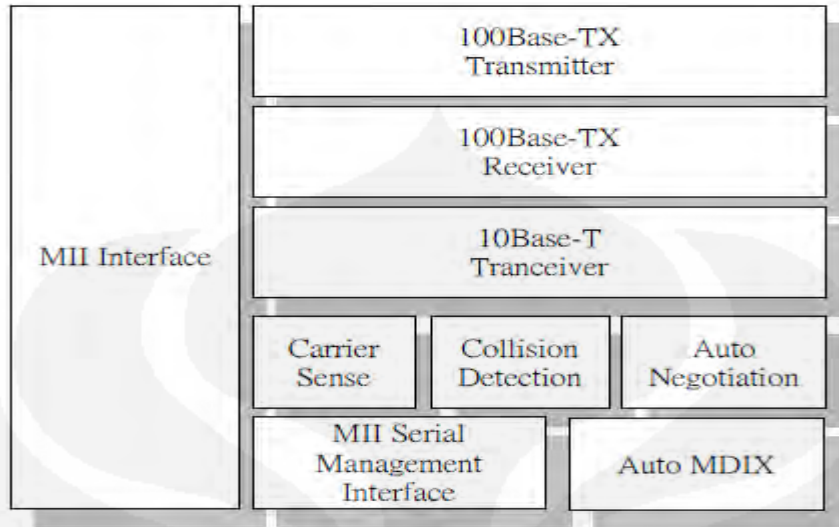


Figure 4 - DM9161A Chip General Functional Description

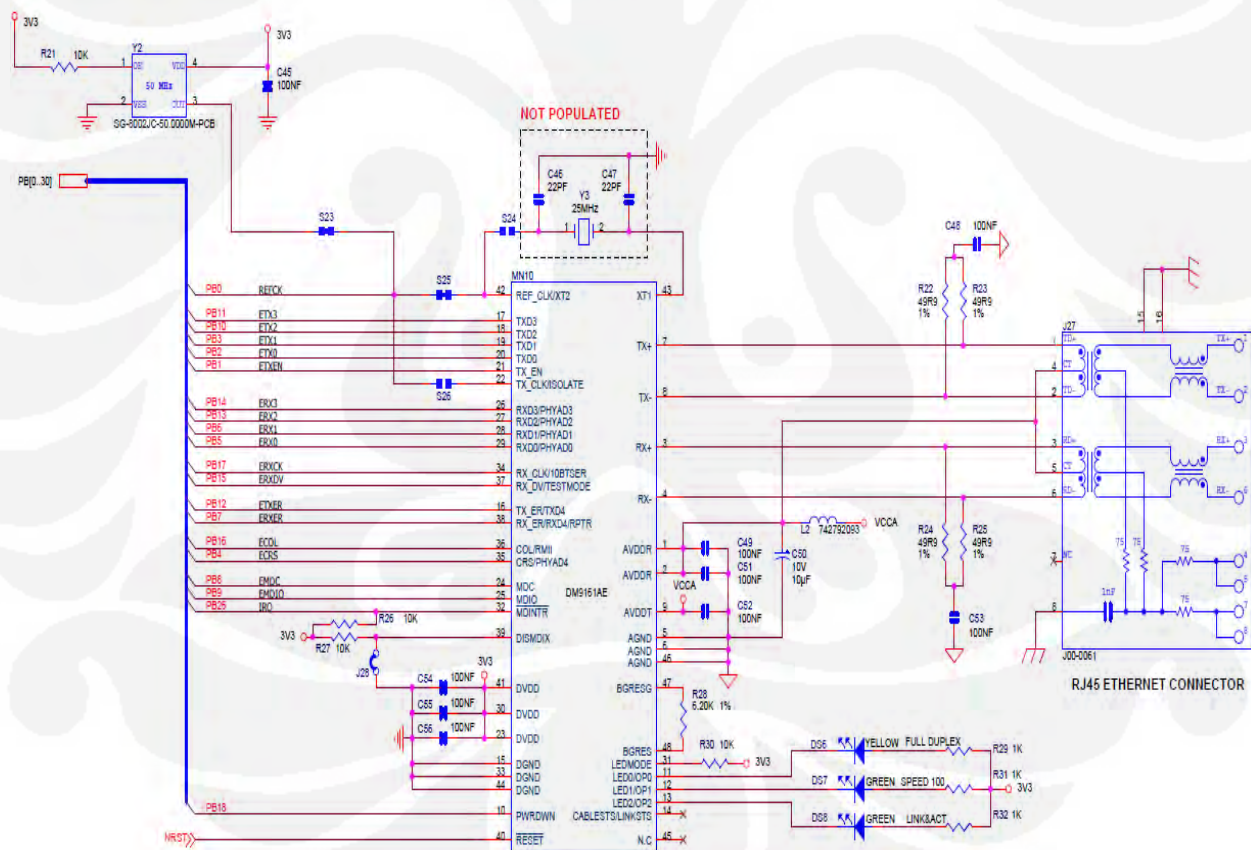


Figure 5 - DM9161A Schematic

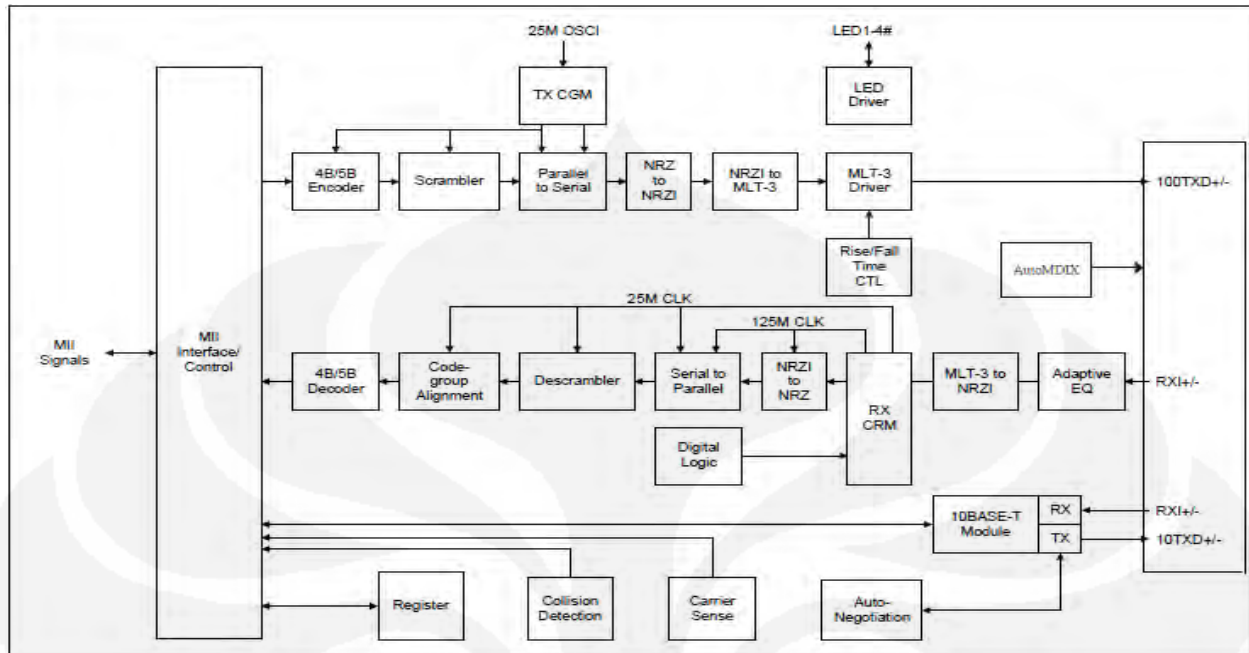


Figure 6 - Specific Functional Description

- MII Interface.** The purpose of Media Independent Interface (MII) is to provide a simple, easy to implement connection between the MAC reconciliation layer and the PHY. The MII is designed to make the differences between various media transparent to the MAC sub-layer.

MII Interface consists of a nibble wide receive data bus, a nibble wide transmit bus, and control signals for data transfer between PHY and the reconciliation layer. Table 1 shows the important bus in MII interface.

BUS	Function
TXD (transmit data)	A nibble of data that synchronous with respect of TXCLK.
TXCLK (transmit clock)	Continuous clock that provides the timing reference for the transmission transfer.
TXEN (transmit enable)	A nibble of data being presented on the MII for transmission.
TXER (transmit error)	Error detected somewhere in the frame being transmitted.
RXD (receive data)	A nibble of data that synchronous with respect of RXCLK.
RXCLK (receive clock)	Continuous clock that provides the timing reference for receiving

clock)	transfer.
RXDV (receive data valid)	PHY is presenting decoded nibbles to the MAC reconciliation sub-layer.
RXER (receive error)	Error detected somewhere in the frame transmission for the PHY to the reconciliation layer.
CRS (carrier sense)	CRS is asserted when either transmit or receive medium is being processed.

Table 1 - Busses Function in MII Interface

- **100Base-TX Transmitter.**

As shown in figure 5, 100Base-TX Transmitter consists of the functional block that converts a nibble synchronous data provided by the MII to a scrambled MLT-3 125, a million symbols per second data stream. It contains the following functional diagram:

- 1. 4B5B Encoder**

It converts 4-bit nibble data from MAC reconciliation layer into 5-bit code group for transmission. This conversion is required for control and packet data to be combined in code groups. To convert them, see appendix table 4B5B as a reference.

- 2. Scrambler**

The scrambler is required to control the radiated emission so that the total energy presented to the cable is distributed over a wide frequency range. The result is a data stream with sufficient randomization to decrease radiated emission at critical frequencies.

- 3. Parallel to Serial Converter**

It receives 5-bit scrambled data. In order to be able operated by NRZ to NRZI Encoder, the parallel data stream should be serialized.

- 4. NRZ to NRZI Encoder**

Data stream should be converted from serialized NRZ that receive from parallel to serial converter, to NRZI for TP-PDM standard compatibility over Category -5 unshielded twisted pair cable.



Figure 7 - NRZ to NRZI Encoding Example

5. NRZI to MLT-3

Then MLT-3 conversion is accomplished by converting the NRZI data stream into two binary data streams with alternately phased logic one events.

6. MLT-3 Driver

The two binary data streams are fed to the twisted pair output driver that will be compatible of the transmit transformer's primary winding, resulting in a minimal current MLT-3 Signal.

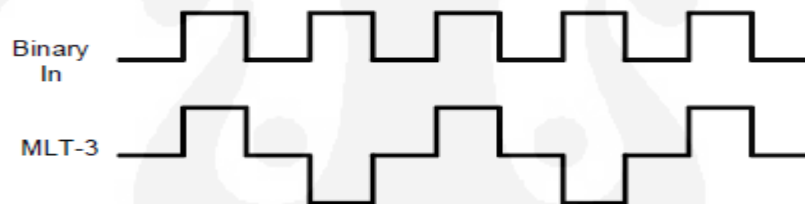


Figure 8 - MLT-3 Converter

- **100Base-TX Receiver**

Data stream received by the chip should have converted to synchronous 4-bit nibble data, so that can be processed by the MII. It contains the following functional diagram:

1. Signal Detect

It should have met the specifications mandated by ANSI XT12 TP-PMD 100Base-TX Standards for both voltage threshold and timing parameters.

2. Adaptive Equalizer

When receiving data from copper twisted at high speed, attenuation based on frequency can affect the randomness of the scrambled data stream. This variation in

signal attenuation caused by frequency variations must be compensated for to ensure the integrity of the received data. Moreover, it should have to be adaptive to ensure proper condition of received signal.

3. MLT-3 to NRZI Decoder

Then MLT-3 to NRZI applied as shown in Figure 7.

4. Clock Recovery Module

Clock Recovery Module will lock onto the current data stream and extract reference clock, so that the data stream can be processed at NRZI to NRZ Decoder.

5. NRZI to NRZ Decoder

Data stream is required to be decoded to NRZ signal to be presented to the Serial to parallel conversion block.

6. Serial to Parallel

The NRZ data stream then will be converted to parallel to be presented to the descrambler

7. Descrambler

Because the data stream is scrambled in order to minimize radiated emission for transmission data, it should have to be descrambler and present to the Code Group Alignment group before it can be processes in MII interface.

8. Code Group Alignment

Un-aligned 5B data from descrambler will be converted to 5B code group data in order to make aligned subsequent data on a fixed boundary.

9. 4B5B Decoder

Finally, Conversion from 5-bit data to 4-bit nibble data is accomplished by 4B5B decoder to be ready presented to the reconciliation layer by MII interface.

- **10 Base-T Operation**

When 10 Base-T Mode is operated, for transmission, a nibble data format will be converted to a serial bit stream then encoded my Manchester encoder. When receiving, the data stream will be decoded and converted to nibble format to be presented to the MII interface.

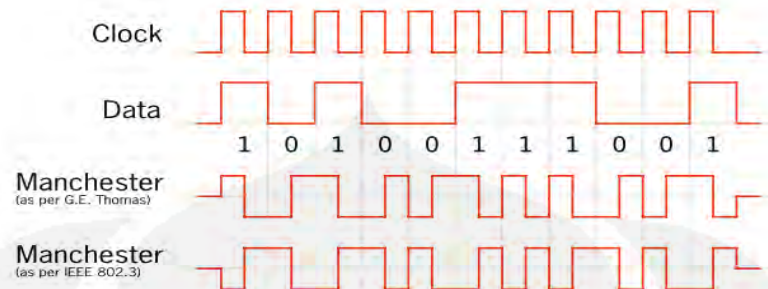


Figure 9 - Manchester Encoding Example

- **Carrier Sense**
CRS is used in half-duplex operation during transmission or reception of data in order to avoid collision in traffic.
- **Collision Detection**
Collision Detection also works in half-duplex operation when transmit and receive channels are active at the same time. It will be reported by COL signal on the MII interface.
- **Auto Negotiation**
The function of Auto-negotiation is to provide a means to exchange information between segment linked devices and to automatically configure both devices to take maximum advantages of their abilities.
- **MII Serial Management**
MII serial management interface consists of a data interface, basic register set, and a serial management interface to configure multiple PHY devices, get status and error information, and also determine the type and capabilities of the attached PHY device. The serial control interface uses a simple two wired serial interface to obtain and control physical layer. It consists of MDC (Management Data Clock), and MDI/O (Management Data Input/Output) signals which pin is bi-directional and shared up to 32 devices.
- **Auto MDIX**
Common Ethernet network cables are straight and crossover cable. This Ethernet network cable is made of 4 pair high performance cable that consist twisted pair conductor that

used for data transmission. Auto MDIX is used to detect cable connection type, so that those cables still can be worked into Ethernet interface.

2.4 Parallel Input/Output Controller

PIO controller provides multiplexing up to two peripheral functions on a single pin. Peripheral A and peripheral B represent Joystick Input and User LED respectively. PIO Controller manages up to 32 fully programmable input/output lines. Each I/O line dedicated as a general-purpose I/O or be assigned to a function of an embedded peripheral.

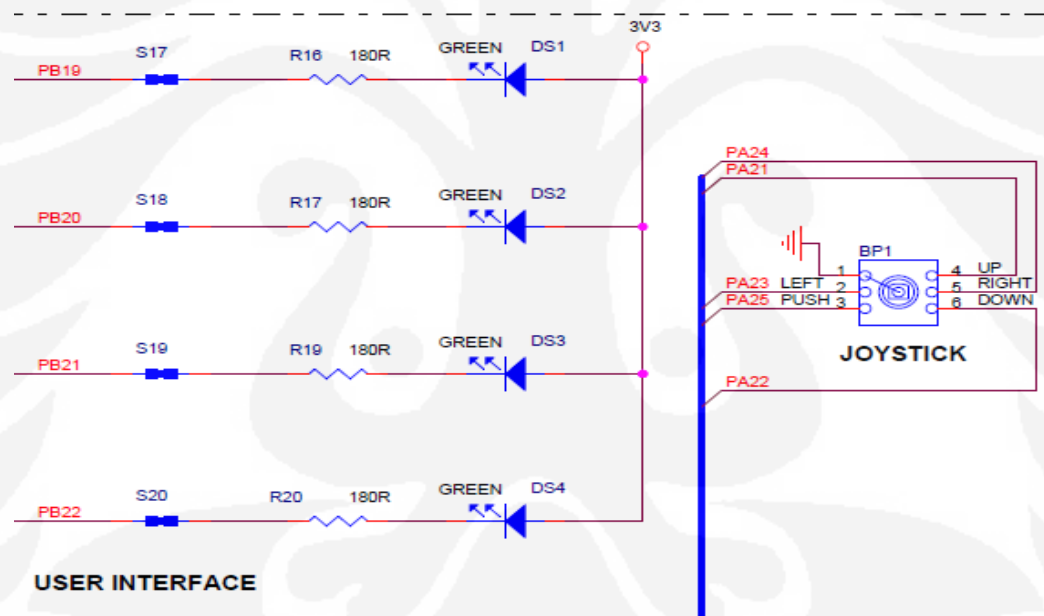


Figure 10 - PIO Schematic

Each pin is configurable according to product definition so programmer must carefully determine the configuration of the PIO controller required by their application. There are some aspects that should be consider controlling the PIO such as Pull-up resistor control, Peripheral Function Selection, Output Control, Synchronous Data output, Input data, Input Change

Interrupt, etc. Table 2 describes PIO controller that associated with a bit in each of the PIO Controller User Interface Register.

Offset	Register	Name	Access	Reset
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or ⁽²⁾ Read-write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register ⁽⁴⁾	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Table 2 - PIO Register Mapping

Chapter 3 - Communication Protocol

To establish connection between the embedded platform and other devices, we need to consider how they actually communicate each other. In this section student will explain about the protocol hierarchies, service primitive, TCP/IP Reference Model.

3.1 Protocol Hierarchies

Basically Protocol is an agreement between the communicating parties on how communication is to proceed. To reduce the design complexity, most networks are organized as a stack of layers or levels, each one built upon the one below it. The purpose of each later is to offer certain services to the higher layers, shielding those layers from the detail of how the offered services are actually implemented. Each layer is a kind of virtual machine, offering certain services to the layer above it.

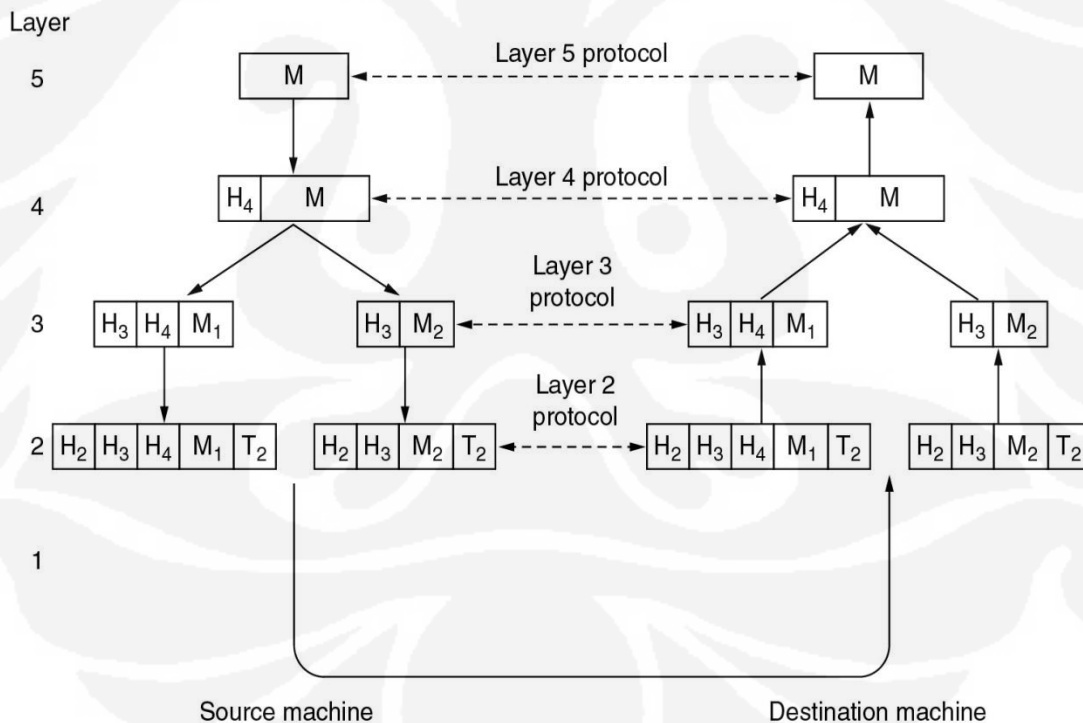


Figure 1 - Example Information Flow Supporting Virtual Communication

As illustrated in Figure 11, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data to the layer immediately below it,

until the data reach to the lowest layer. Layer 1 is the physical medium which actual communication occurs.

The entities comprising the corresponding layers on different machines are called peers. The peers may be processes or hardware devices. In other words, it is the peers that communicate by using the protocol. Between each pair of layers is an interface. The interface defines which primitive operations and services the lower layer makes available to the upper one.

3.2 Service Primitive

A service is formally specified by a set of primitives (operations) available to a user process to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packet. [5]

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

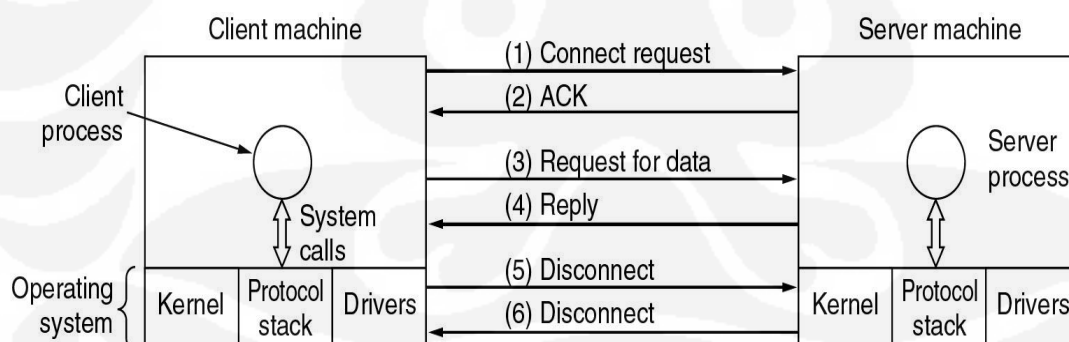


Figure 2 - Packet sent in a simple client-server interaction on a network

These primitives might be used as follows. First, the server executes LISTEN to indicate that it is prepared to accept incoming connections. Next, the client process executes CONNECT to establish a connection with the server (1). The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. When the packet arrives at

the server, it is processed by server's operating system to see if there is a listener and send back the acknowledgment (2).

The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before acknowledgement can get back to the client. Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply. After the arrival of the request packet at the server machine will process the request by uses SEND to return the answer to client (4).If it is done, it can use DISCONNECT to terminate the connection. It used a handshake scheme to end communication between server and client.

3.3 TCP/IP Reference Model

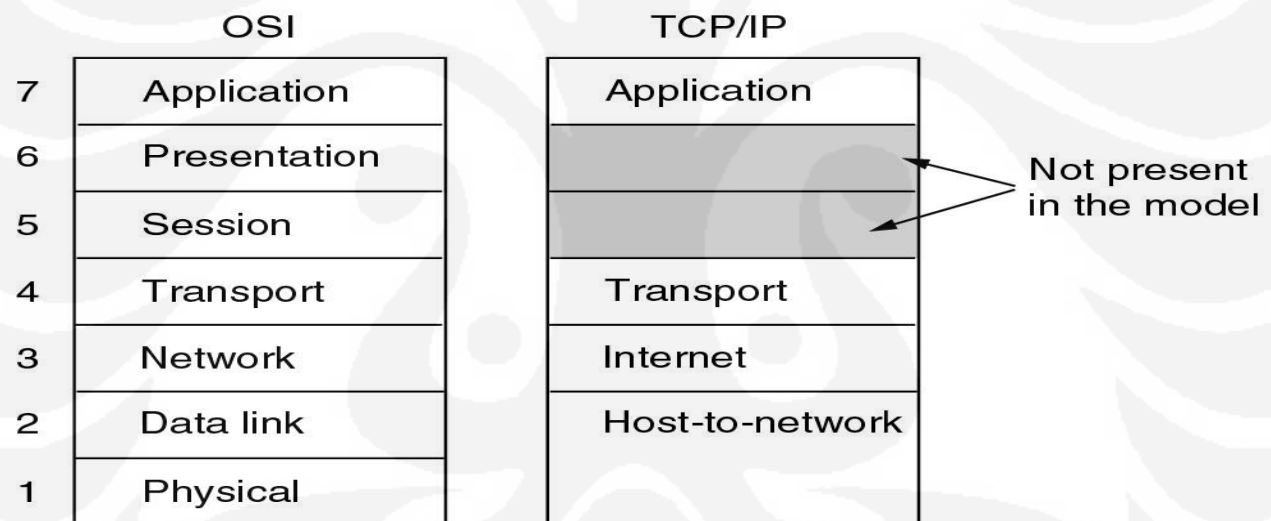


Figure 3 - the difference between OSI and TCP/IP Reference Model

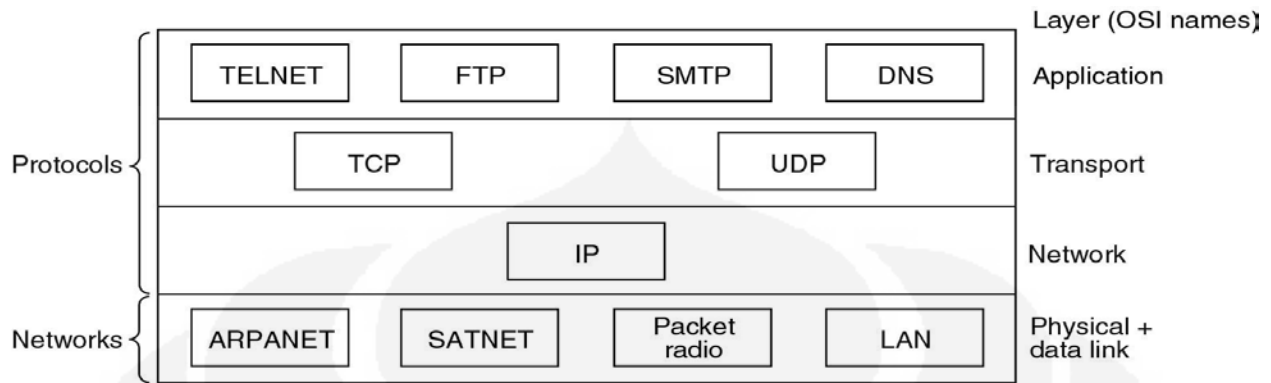


Figure 4 - Protocol and Networks in the TCP/IP model

Application Layer

The TCP/IP model does not have session and presentation layer like OSI model. Therefore, it is not necessary for both layers to be perceived, so they were not included. Application layer contains all the higher-level protocol, for example virtual terminal (Telnet), File Transfer Protocol (FTP), electronic mail (SMTP), protocol for fetching pages (HTTP) etc.

Transport Layer

Transport Layer is designed to allow peer entities on the source and destination hosts to carry on a conversation. As shown in Figure 13, there are 2 protocols that have been defined; TCP and UDP. TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error. UDP (User Datagram Protocol) is an unreliable, connectionless protocol application that does not want TCP's sequencing or flow control and wish to provide their own.

Internet Layer

The internet layer defines an official packet format and protocol called IP (Internet Protocol). The job of the internet layer is to deliver IP packets where they are supposed to go and select the best path through the network for packet to travel. ICMP, ARP also operated at this layer.

Host-to-network layer

TCP/IP reference model does not really say much about what happen in this layer, except that the host has to connect to the network using some protocol so it can send IP packets to it. This protocol is not define and varies from host to host and network to network.

3.4 TCP Protocol

Transmission Control Protocol that referred at transport layer in TCP/IP reference model is used in this implementation. It provides reliable end-to-end delivery service including data transmission and flow control. Using the reliable service, there must not any data loss and the frame has to be reassembled in the right places and makes up for Internet Protocol's (IP) deficiencies. TCP adds a great deal of functionality to the IP service compare to UDP is layered over:

- **Reliable Delivery and Round Trip Estimation.** Sequence numbers are used to coordinate which data has been transmitted and received. TCP will arrange for retransmission if it determines that data has been lost in expected time period.
- **Network Adaptation.** TCP will dynamically learn the delay characteristics of a network and adjust its operation to maximize throughput without overloading the network
- **Flow Control.** TCP manages data buffers, and coordinates traffic so its buffer will never overflow. Fast senders will be stopped periodically to keep up with slower receivers.

TCP is provided through three mechanisms:

1. Acknowledgment.

When a receiver gets a message from a transmitter, the receiver has to acknowledge the message by sending acknowledgment to the transmitter.

2. Sliding Windows

The receiver stop the transmitter from sending messages if a message was dropped. Then the receiver tells the transmitter the number of message was expected to be continued.

3. Sequence Number

TCP uses a 32-bit sequence number that counts bytes in the data stream. Each TCP packet contains the starting sequence number of the data in that packet, and the sequence number of the last byte received from the remote peer. With this information, a sliding window is implemented and each TCP peer must track both its own sequence numbering and the numbering being used by the remote peer. If the number received is not in sequence, the receiver will tell the transmitter that the number was wrong and gave it expected number to be retransmitted.

And also when the client want to terminate the established connection it uses a handshake scheme to end communication. Client will send message with FIN flag set to indicate that the client want to terminate the connection. When server receives the message, it will send the acknowledgment first and then terminate its connection.

Chapter 4 – Real Time Operating System

GCC environment is decided to use in this project, therefore open source called FreeRTOS is used to implement a scale-able real time kernel that designed specifically for small embedded systems. It means that routine and some modules in the program implementation are based on this open-source. There are some advantages using FreeRTOS open-source are [11]:

- Preemptive, cooperative and hybrid configuration options.
- Designed to be small, simple and easy to use.
- Very portable code structure predominantly written in C.
- Support both tasks and co-routines.
- Stack-overflow detection options.

4.1 RTOS Concept

In order to develop FreeRTOS to be satisfied to our needs, it is essential to know the fundamental and the background of RTOS concept. Student will write RTOS concept such as Multitasking, Scheduling, Context Switching, and how they will be processed in running task and co-routines.

Multitasking

A conventional processor can only execute a single task at a time, on the other hand by rapidly switching between tasks a multitasking operation system can make it appear as if each task is executing concurrently. Figure 14 explain the execution pattern of three tasks with respect to time. The upper diagram demonstrates the perceived concurrent execution pattern and the lower the actual multitasking execution pattern.

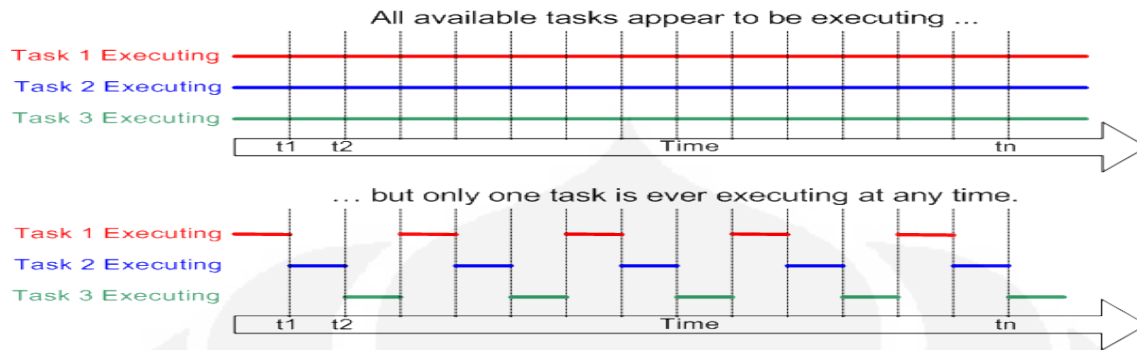


Figure 1 - Conventional VS Multitasking

Scheduling

The scheduler is part of the kernel that is responsible to decide which task should be executed at any particular time. The scheduling policy is the algorithm used by the scheduler to decide which task to execute at a specific time.

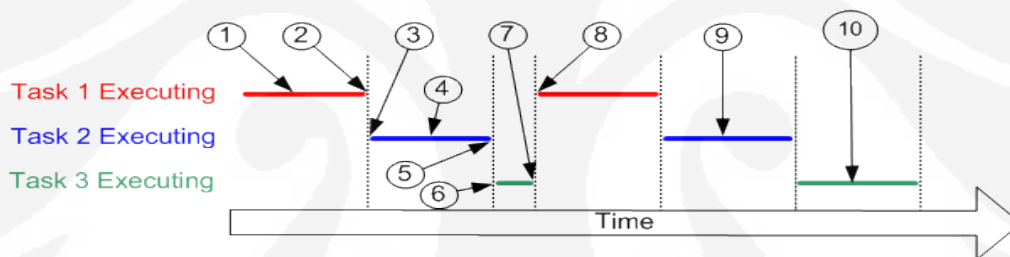


Figure 2 - Scheduling Example

Referring to the numbers in figure above:

- At (1) task 1 is executing.
- At (2) the kernel suspends task 1.
- At (3) resumes task 2.
- While task 2 is executing (4), it locks a processor peripheral for its own exclusive access.
- At (5) the kernel suspends task 2.
- At (6) resumes task 3.
- Task 3 tries to access the same processor peripheral, finding it locked task 3 cannot continue so suspends itself at (7).
- At (8) the kernel resumes task 1.

- The next time task 2 is executing (9) and it finishes.
- The next time task 3 is executing (10) and it finishes.

Context Switching

As a task executes, it utilizes the microcontroller registers and accesses RAM and ROM just as any other program. These resources such as processor register, stack, etc comprise the task execution context. While the task is suspended, other task will execute and may modify the processor register values. Upon resumption, the task will not know that the processor have been altered and result in an incorrect value.

The operating system kernel is responsible to ensure saving the context of a task as it is suspended. So, when the task is being resumed, its saved context is restored and task will continue in a correct value.

4.2 The differences between Task and Co-routines

There are several API references such as Task creation control utilities, Kernel Control, Queues, Semaphore/Mutexes and Co-routines those being used in FreeRTOS open-source as a default. However, student only added and modified some aspects that can fulfill the project requirements in Task management and Co-Routines. Therefore, student will explain the differences between those two.

TASK

In a real time application that uses an RTOS can be structured as a set of tasks and each task executes within its own context. Unfortunately, only one task within the application can be executed at any point in time and real time scheduler is responsible for deciding which task should be executed. Therefore, scheduler repeatedly starts and stops each task, and as a task has no knowledge of the scheduler activity, scheduler also responsible for context switching process when a task is being swapped out and swapped in. To achieve this one, each task is provided with its own stack, so the context will be saved to the stack of that task. As a result, it will use high RAM usage.

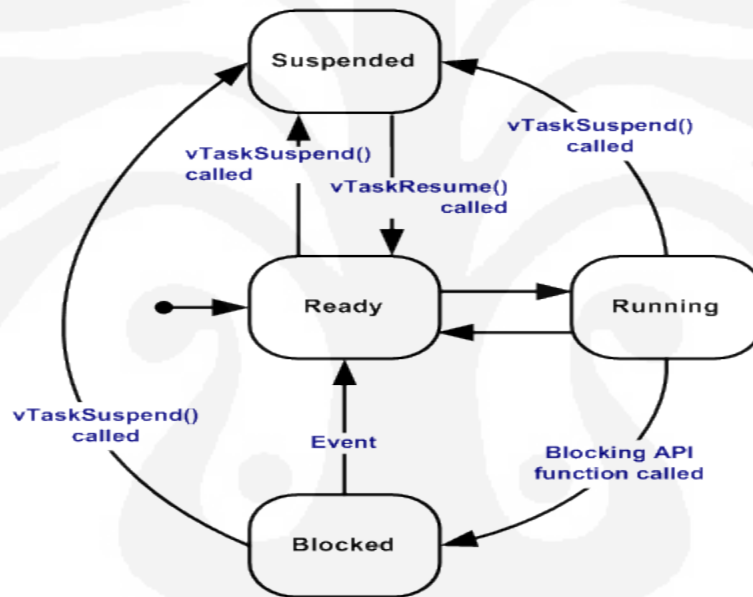


Figure 3 - Task States

A task can exist in one of the following states:

- **Running**

When a task is actually executing, it is said to be in the Running state. It is currently utilizing the processor.

- **Ready**

Ready tasks are those that are able to execute but are not executing because a different task of equal or higher priority is already in the running state. They are not blocked or suspended.

- **Blocked**

A task is said to be in the blocked state if it is currently waiting for either a temporal or external event and it will block until the delay period has expired. Blocked tasks are not available for scheduling.

- **Suspended**

Task in Suspended state are also not available for scheduling. Tasks will only enter or exit the Suspended state when explicitly commanded to do, so there are going to have two functions; Suspend and Resume.

Each task is assigned a priority. The scheduler will ensure that a task in the ready or running state will always be given processor time in preference to tasks of a lower priority that are also in the ready state. In other words, the task given processing time will always be the highest priority task that able to run.

Co-Routine

All the co-routines within an application share a single stack. This reduces the amount RAM usage compare with tasks. Co-routines use prioritized cooperative scheduling with respect to other co-routines and also its implementation is provided through a set of macros.

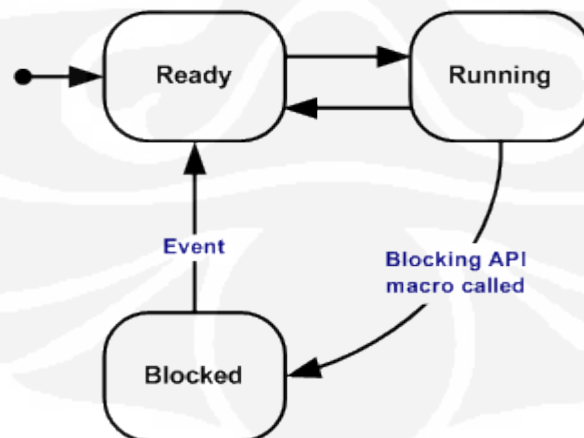


Figure 4 - Co-Routine States

A co-routine can exist in one of the following states:

- **Running**

When a co-routine is actually executing it is said to be in the running state. It is currently utilizing the processor.

- **Ready**

Ready co-routine are those that are able to execute (they are not blocked) but are not currently executing. A co-routine may be in the ready state because of 2 reasons:

1. Another co-routine of equal or higher priority is already in the running state.
2. If the application uses both tasks and co-routines, a co-routine might be in the ready state when the task is in the running state.

- **Blocked**

A co-routine is said to be in the Blocked state if it is currently waiting for either a temporal or extended event.

4.3 FreeRTOS Open-Source Application Demonstration

The table below lists the files that make up the demo projects along with a brief indication of the RTOS features demonstrated and describes each task and co-routine within the demo project.

File	Features Demonstrated
main.c	<ul style="list-style-type: none">• Starting/Stopping the kernel• Using the trace visualisation utility• Allocation of priorities
dynamic.c	<ul style="list-style-type: none">• Passing parameters into a task• Dynamically changing priorities

	<ul style="list-style-type: none"> • Suspending tasks • Suspending the scheduler
BlockQ.c	<ul style="list-style-type: none"> • Inter-task communications • Blocking on queue reads • Blocking on queue writes • Passing parameters into a task • Pre-emption • Creating tasks
ComTest.c	<ul style="list-style-type: none"> • Serial communications • Using queues from an ISR • Using semaphores from an ISR • Context switching from an ISR • Creating tasks
CRFlash.c	<ul style="list-style-type: none"> • Creating co-routines • Using the index of a co-routine • Blocking on a queue from a co-routine • Communication between co-routines
CRHook.c	<ul style="list-style-type: none"> • Creating co-routines • Passing data from an ISR to a co-routine • Tick hook function • Co-routines blocking on queues
Death.c	<ul style="list-style-type: none"> • Dynamic creation of tasks (at run time) • Deleting tasks

	<ul style="list-style-type: none"> • Passing parameters to tasks
Flash.c	<ul style="list-style-type: none"> • Delaying • Passing parameters to tasks • Creating tasks
Flop.c	<ul style="list-style-type: none"> • Floating point math • Time slicing • Creating tasks
Integer.c	<ul style="list-style-type: none"> • Time slicing • Creating tasks
PollQ.c	<ul style="list-style-type: none"> • Inter-task communications • Manually yielding processor time • Polling a queue for space to write • Polling a queue for space to read • Pre-emption • Creating tasks
Print.c	<ul style="list-style-type: none"> • Queue usage
Semtest.c	<ul style="list-style-type: none"> • Binary semaphores • Mutual exclusion • Creating tasks

Table 1 - FreeRTOS Function

Chapter 5 – TCP/IP Stack

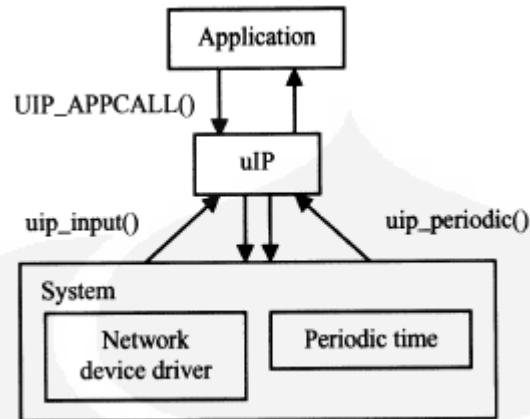
The uIP stack that student used for TCP/IP stack is an open source that is intended to make it possible to communicate using TCP/IP protocol suite even on small 8-bit micro-controllers. The size of the code is only up to a few kilobytes and RAM usage can be configured to be as low as a few hundred bytes.

The uIP TCP/IP stack becomes one of alternatives free open-source to substitute the business version that has a very expensive price in correspondence. Although the effect of business version is perfect, programmers choose some free TCP/IP stacks and improve them to satisfy their needs. The uIP stack can be run either as a task in a multitasking system, or as the main program in a single tasking system.

This uIP TCP/IP Stack has the following features: [4]

- Well documented and well commented source code
- Very small code size
- Very low RAM usage, configurable at compile time
- ARP, SLIP, IP, UDP, ICMP, and TCP protocols.
- Includes a set of example applications: web server, web client, SMTP client, Telnet server, DNS hostname resolver.
- Any number of concurrently active TCP connections.
- Any number of passively listening (server) TCP.
- Free for both commercial and non-commercial use.

5.1 Main Control Loop



The main control loop in uIP stack does two things repeatedly:

- Check if a packet has arrived from the network. (Using function `uip_input()`).
- Check if a periodic timeout has occurred. (Using function `uip_periodic()`).

5.2 Architecture Specific Functions

uIP requires a few functions to be implemented specifically for the architecture to run. C language implementations are given as part of the uIP distribution. Below is the basic function of uIP stack:

1. Checksum Calculation

The TCP and IP protocols implement a checksum that covers the data and header portions of the TCP and IP packets. Since the calculation of this checksum is made over all bytes in every packet being sent and received it is important that the function that calculates the checksum is efficient.

While uIP includes a generic checksum function, it also leaves it open for an architecture specific implementation of the two functions `uip_ipchksum()` and `uip_tcpchksum()`.

2. 32-bit Arithmetic

The TCP protocol uses 32-bit sequence numbers, and a TCP implementation will have to do a number of 32-bit additions as part of the normal protocol processing. Since 32-bit arithmetic is not natively available on many of the platforms for which uIP is intended, uIP leaves the 32-bit additions to be implemented by the architecture specific module and does not make use of any 32-bit arithmetic in the main code base.

While uIP implements a generic 32-bit addition, there is support for having an architecture specific implementation of the `uip_add32()` function.

3. Memory Management

The uIP stack does not use explicit dynamic memory allocation. Instead, it uses a single global buffer for holding packets and has a fixed table for holding connection state. The global packet buffer is large enough to contain one packet of maximum size. When a packet arrives from the network, the device driver places it in the global buffer and calls the TCP/IP stack. If the packet contains data, the TCP/IP stack will notify the corresponding application. Because the data in the buffer will be overwritten by the next incoming packet, the application will either have to act immediately on the data or copy the data into a secondary buffer for later processing.

The total amount of memory usage for uIP depends heavily on the applications of the particular device in which the implementations are to be run. The memory configuration determines both the amount of traffic the system should be able to handle and the maximum amount of simultaneous connections.

4. Application Program Interface (API)

The Application Program Interface (API) defines the way the application program interacts with the TCP/IP stack. The most commonly used API for TCP/IP is the BSD socket API which is used in most Unix systems and has heavily influenced the Microsoft Windows WinSock API. Because the socket API uses stop-and-wait semantics, it requires support from an underlying multitasking operating system. Since the overhead of task management, context switching and allocation of stack space for the tasks might be too high in the intended uIP target architectures, the BSD socket interface is not suitable for our purposes.

5.3 The uIP raw API

The "raw" uIP API uses an event driven interface where the application is invoked in response to certain events. An application running on top of uIP is implemented as a C function that is called by uIP in response to certain events. uIP calls the application when data is received, when data has been successfully delivered to the other end of the connection, when a new connection has been set up, or when data has to be retransmitted. The application is also periodically polled for new data. The application program provides only one callback function; it is up to the application to deal with mapping different network services to different ports and connections. Because the application is able to act on incoming data and connection requests as soon as the TCP/IP stack receives the packet, low response times can be achieved even in low-end systems.

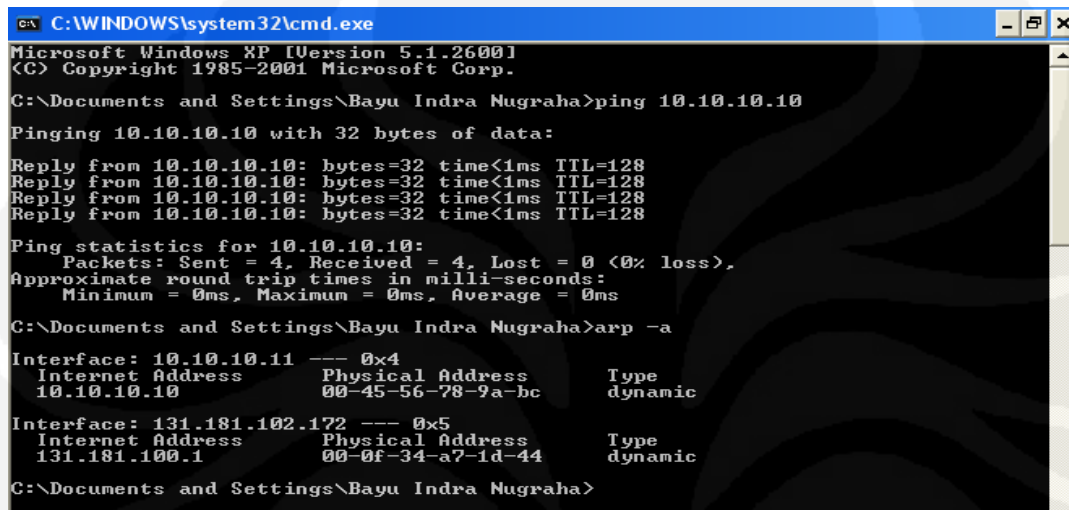
Interface function	Application event
uip_listen()	Start listening on a port
uip_send()	Send data on the current connection
uip_acked()	Sent data has been acknowledged
uip_newdata()	Remote host has sent new data
uip_datalen()	The size of the incoming data
uip_connect()	Connect to a remote host
uip_connected()	The current connection has just been connected
uip_poll()	Application is being polled
uip_close()	Close the current connection
uip_abort()	Abort the current connection
uip_stop()	Stop the current connection

Table 1 - uIP Interface Function

5.4 uIP Simple Application

Hello World (ICMP)

Hello World uIP application is an example showing how to write applications with protosockets function. The protosocket library in uIP provides functions for sending data without having to deal with retransmissions and acknowledgements, as well as functions for reading data without having to deal with data being split across more than one TCP segment.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Bayu Indra Nugraha>ping 10.10.10.10

Pinging 10.10.10.10 with 32 bytes of data:

Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.10.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Bayu Indra Nugraha>arp -a

Interface: 10.10.10.11 --- 0x4
    Internet Address      Physical Address      Type
    10.10.10.10           00-45-56-78-9a-bc    dynamic

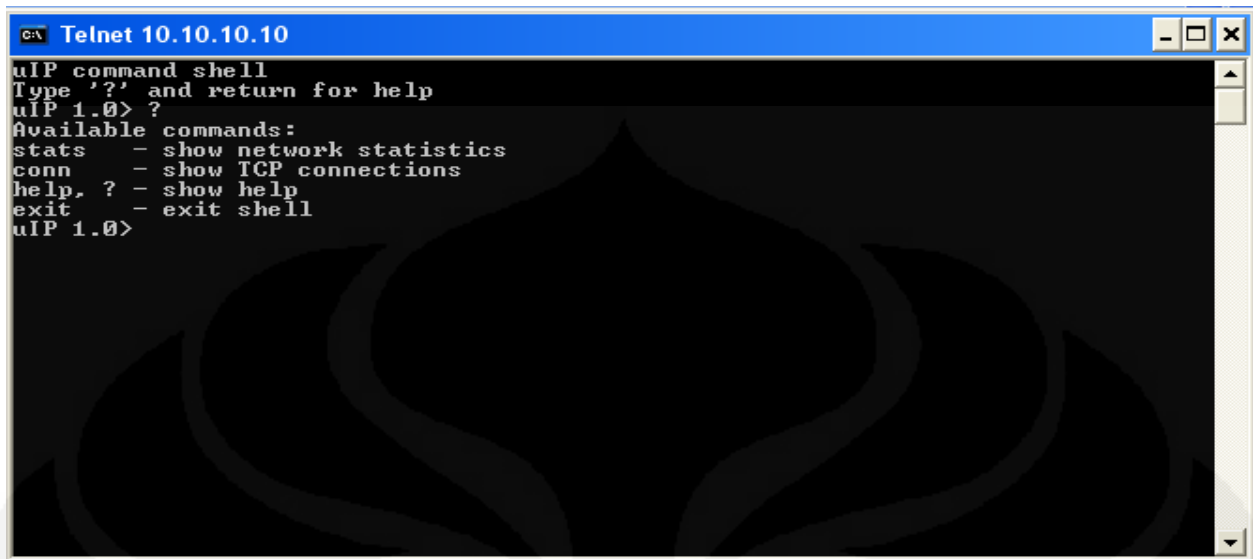
Interface: 131.181.102.172 --- 0x5
    Internet Address      Physical Address      Type
    131.181.100.1         00-0f-34-a7-1d-44    dynamic

C:\Documents and Settings\Bayu Indra Nugraha>
```

Figure 1 - uIP basic ICMP Demo

Telnet

The purpose of this application is to provide a TCP bidirectional interactive communications facility in port 23. Typically, telnet provides access to a command-line interface on a remote host via a virtual terminal connection which consists of an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). User data is interspersed in-band with TELNET control information.

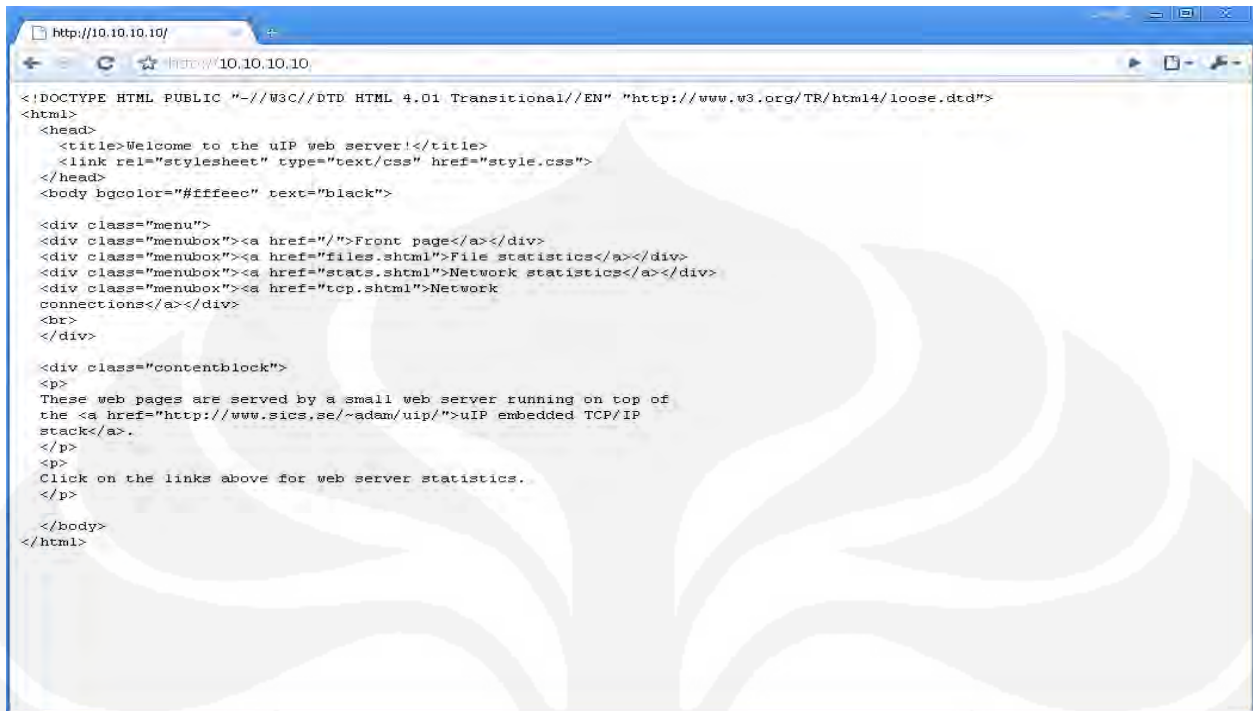
A screenshot of a Telnet window titled "c:\ Telnet 10.10.10.10". The window contains a text-based interface for a uIP command shell. The text displayed is: "uIP command shell", "Type '?' and return for help", "uIP 1.0> ?", "Available commands:", "stats - show network statistics", "conn - show TCP connections", "help. ? - show help", "exit - exit shell", and "uIP 1.0>". The window has a standard Windows-style title bar with minimize, maximize, and close buttons.

```
c:\ Telnet 10.10.10.10
uIP command shell
Type '?' and return for help
uIP 1.0> ?
Available commands:
stats - show network statistics
conn - show TCP connections
help. ? - show help
exit - exit shell
uIP 1.0>
```

Figure 2 - uIP Basic Telnet Server Demo

Web Server

The application that responsible for accepting HTTP requests from *clients* (user agents such as web browsers), and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Welcome to the uIP web server!</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body bgcolor="#ffffff" text="black">

<div class="menu">
<div class="menubox"><a href="/">Front page</a></div>
<div class="menubox"><a href="files.shtml">File statistics</a></div>
<div class="menubox"><a href="stats.shtml">Network statistics</a></div>
<div class="menubox"><a href="tcp.shtml">Network
connections</a></div>
<br>
</div>

<div class="contentblock">
<p>
These web pages are served by a small web server running on top of
the <a href="http://www.sics.se/~adam/uip/">uIP embedded TCP/IP
stack</a>.
</p>
<p>
Click on the links above for web server statistics.
</p>
</div>
</body>
</html>
```

Figure 3 - uIP Basic Web Server Demo

Chapter 6 – Algorithm and Design Implementation

There are 4 development environment options that can be used in implementation;

1. YAGARTO
2. uC/OS-II Micrium
3. IAR
4. Rowley Crossworks for ARM

Except for option 3, these environments are all built around the free compiler tool-chain GCC. Furthermore, YAGARTO is being used in this project since it is free open-source software, has some technical documentation done by software developers and there is no memory limitation issue compare with other. Student chose GCC environment because there are more references on building various API applications even though it is more complicated than other development software. Since it is completely free to experiment and deploy pre-configured demo applications to ensure student to start with a known good and working project, student could develop them until they meet the objectives of the project.

Student use 2 GCC open-sources with C language as a basic demo applications. There are:

1. FreeRTOS, a mini Real Time Kernel
2. uIP stack, TCP/IP stack that provides TCP/IP connectivity

Both open-sources are licensed by GNU General Public that guarantees the freedom to share and change them. Student is responsible to obey GNU General Public restriction by still showing the copyright of the software and offer the license which give the student legal permission to copy, distribute and modify the software.

The new module is created from the combination of FreeRTOS and uIP stack. Several API applications that student have been used to achieve the requirement of the project is

discussed in this section. Some ineffective API's that work during the implementation will not be removed since it will make the code structure become unbalance.

6.1 General Program Loop

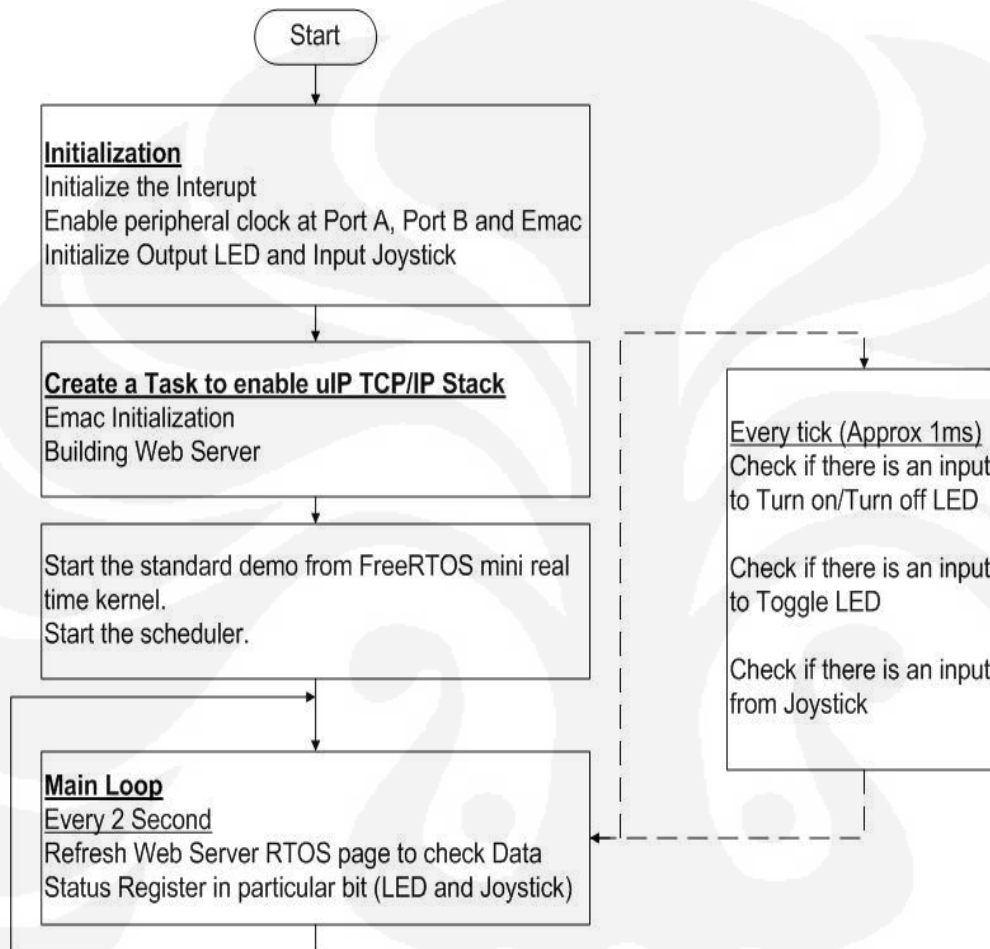


Figure 1 - General Program Loop

Figure 22 describes how the whole program works with infinity loop. The brief explanation of the operations are:

Initialization

Initialize the interrupt. When using JTAG debugger, the hardware is not always initialized to the correct default state. Make sure that issue does not make all interrupt to be masked at the start.

Enable the peripheral clock. It is essential to turn on the clock at peripherals Port A, Port B and the EMAC in order to be able to change bit state at clock transitions in specific time. By “Turn off” the clock mean we block the clock signal to that peripheral.

- **Initialize Output LED and Input Joystick.** To make I/O peripheral to be able work properly, some register in PIO control peripheral need to be enabled. This section will be discussed later.

Create a task to enable uIP TCP/IP Stack. This is how student combine the FreeRTOS and uIP stack by creating a task that provided by FreeRTOS that implement uIP TCP/IP Stack in API function.

EMAC Initialization. To be able to establish the connection and synchronization between EMAC to DM9161A chip, it is necessary to do some general steps for the EMAC initialization;

- Initialize both Tx and Rx descriptor used by the EMAC.
- Enable the Management Data Input/Output bit in MAC Control Register.
- Function to be able read and write value into a PHY register.
- Function to detect MAC and PHY
- EMAC initialization to initialize the Ethernet.
- EMAC initialization to receive packets.
- EMAC initialization to be able to send a packet through EMAC and PHY.
- ICMP, IP, TCP Checksum Calculation.
- Function to be able to send and receive frames.

Building Web Server. Web Server is build by developing the basic web server demo in uIP TCP/IP stack. There are some C project files those become fundamental of the web server in this project;

- **Httpd.c:** This file contains of the macro, or a rule that specifies how a certain input sequence used in HTTPD CGI function. Furthermore, Web server initialization is stored in this file by setting up TCP application in port 80.

- **httpd-fsdata.c:** This file stores HTML script to create web server design. The implementation in this file is using hexadecimal code so that can be read by the processor. As a result ASCII HTML script needs to be manually converted.
- **httpd-fs.c:** This file stores the network statistic that monitored in port 80.
- **httpd-cgi.c:** This file stores web server script interface that can be inserted to httpd-fsdata.c. Student uses functions here in order to create some input/output applications at the web-server.
- **uIP-Task.c:** This file stores the implementation functions that become a bridge from the web server to the embedded platform register.

Start the standard demo from FreeRTOS mini time real time kernel. Several task, co-routine, queue and semaphores are being looped and priorities will be implemented.

Main Loop

Every 2 second: The web server will check the Data Status Register in particular bit in order to check the User LED and Joystick Input status. The web server will be refreshed every 2 second to minimize the load of the network.

Every Tick in Approximately 1 ms

Check an Input user interface from the web server to turn on/turn off or toggle the LED: The bit status for I/O peripheral can be controlled from web server. Student uses 1ms tick to read input user interface status and then update the particular I/O peripheral status.

Check Joystick Input Data Register Status: It has the same concept with above operation. Student uses 1ms tick to read Input from joystick and then update the particular I/O peripheral status and the status at web-server.

6.2 PIO Algorithm

A. Reading Joystick Input

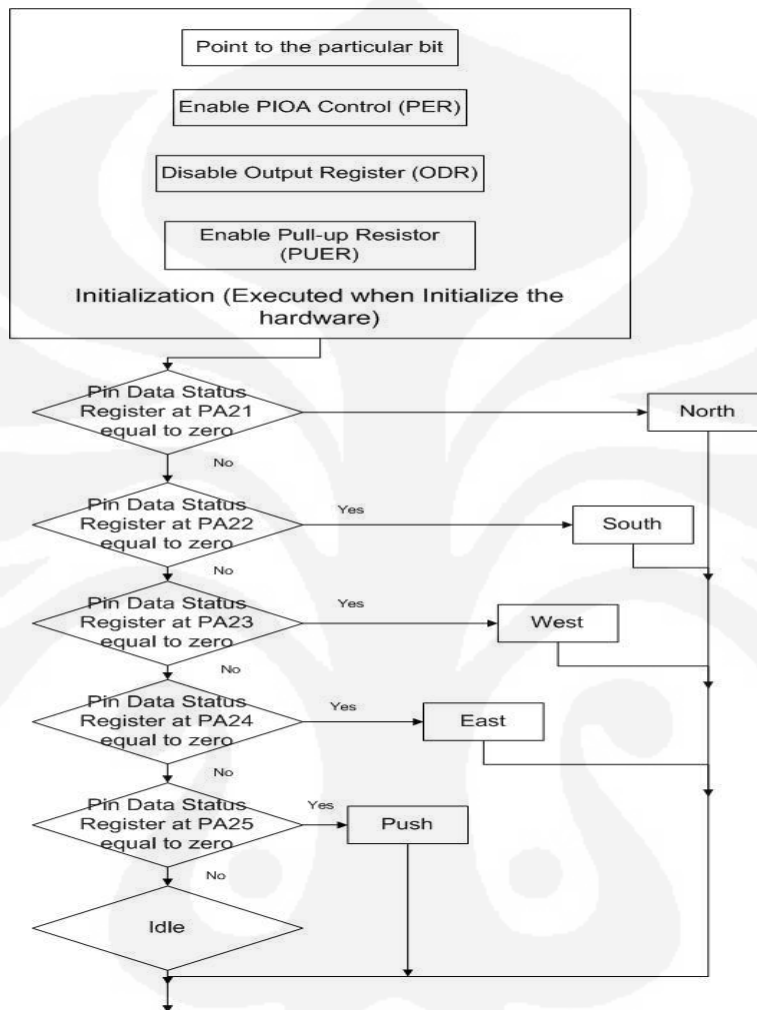


Figure 2 - Reading Joystick Algorithm

Initialization for the input algorithm is executed during hardware initialization at the very beginning of the program. The brief explanations of figure 22 are:

- **Point to the particular bit:** There are 6 wires for Joystick Input those represent the 5 data directions and a ground. Student defines those bits those have bit address PIOA21, PIOA22, PIO23, PIO24, and PIO25 as North, South, West, East, and Push respectively.
- **Enable PIOA Control (PER):** The purpose is to enable the PIO controller and ready to be used in the implementation.

- **Disable Output Register (ODR):** The purpose of disabling the output register is to restrict the data status register so that it only can be updated by controlling the joystick manually.
- **Enable Pull-up Resistor (PUER):** When a joystick contact is closed, it will connect the related port bit to ground. Otherwise the port bit will be floating. The floating status can be avoided by enabling the internal pull-up resistor.
- **Check Data Status Register:** In every 1 ms tick, each Data Status Register will be monitored and also will be uploaded to the web-server.

B. Set the LED Status to turn on/turn off the LED

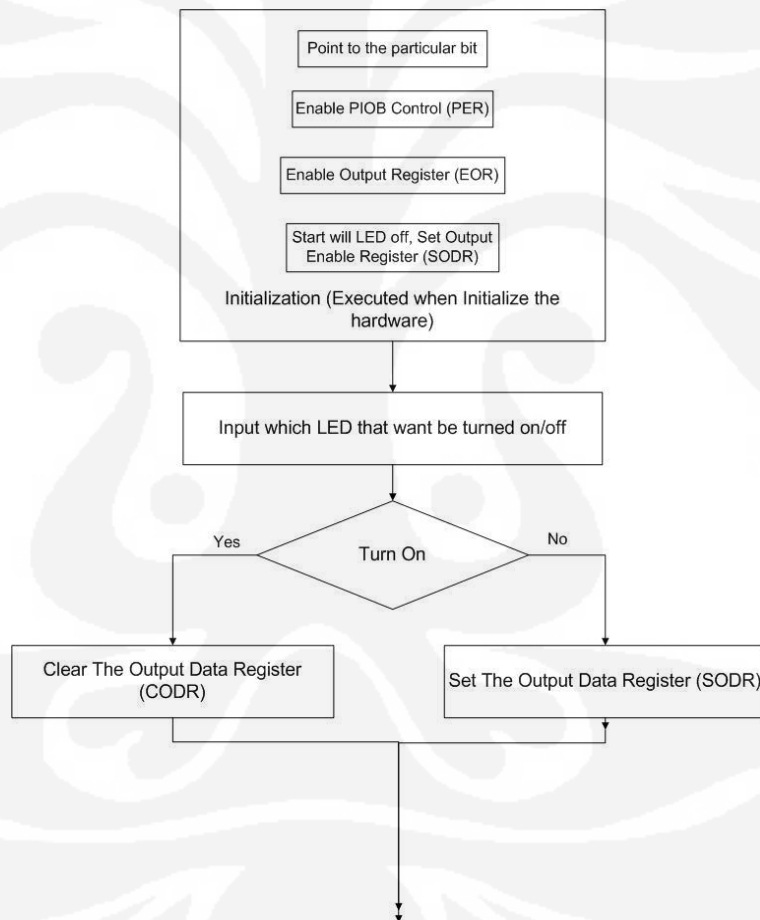


Figure 3 - Algorithm to Set LED Status

Initialization for the input algorithm is executed during hardware initialization at the very beginning of the program. The brief explanation should be like below:

- **Point to the particular bit.** There are 4 address bit that stand for the LEDs; PIOB 19, PIOB20, PIOB21, PIOB22. They are representing LED DS1, DS2, DS3 and DS4 respectively.
- **Enable PIOB Control (PER).** The purpose is to enable the PIO controller and ready to be used in the implementation.
- **Enable Output Register (EOR).** The purpose enabling this register is to be able to update the LED status as an output.
- **Set Output Enable Register (SODR).** The purpose is to make sure the LED is being turned off for the initialization.
- **Set the LED Status.** First, LED which wants to be turned on/off should be defined. Assigning Clear Output Data Register will turn on the LED and assigning Set Output Data Register will turn off the LED.

C. Toggle LED

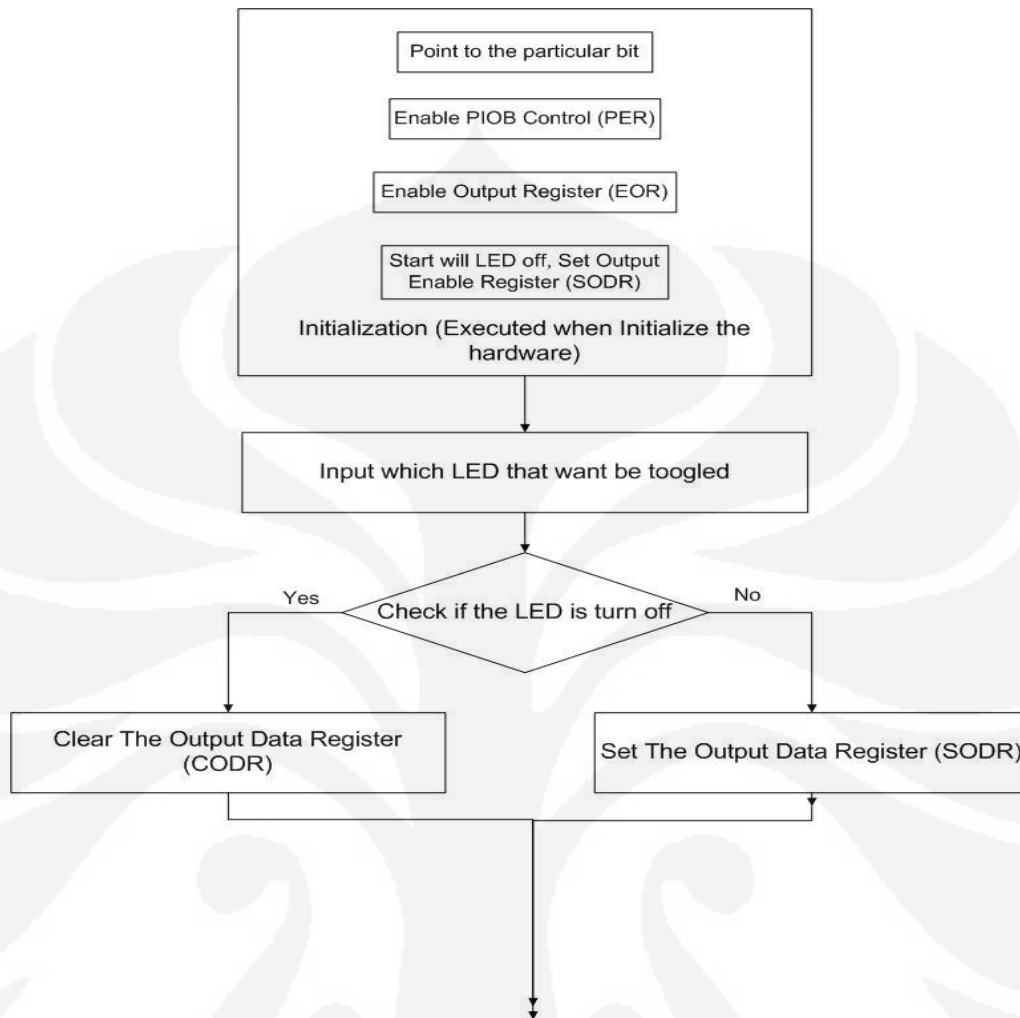


Figure 4 - Algorithm to toggle the LED

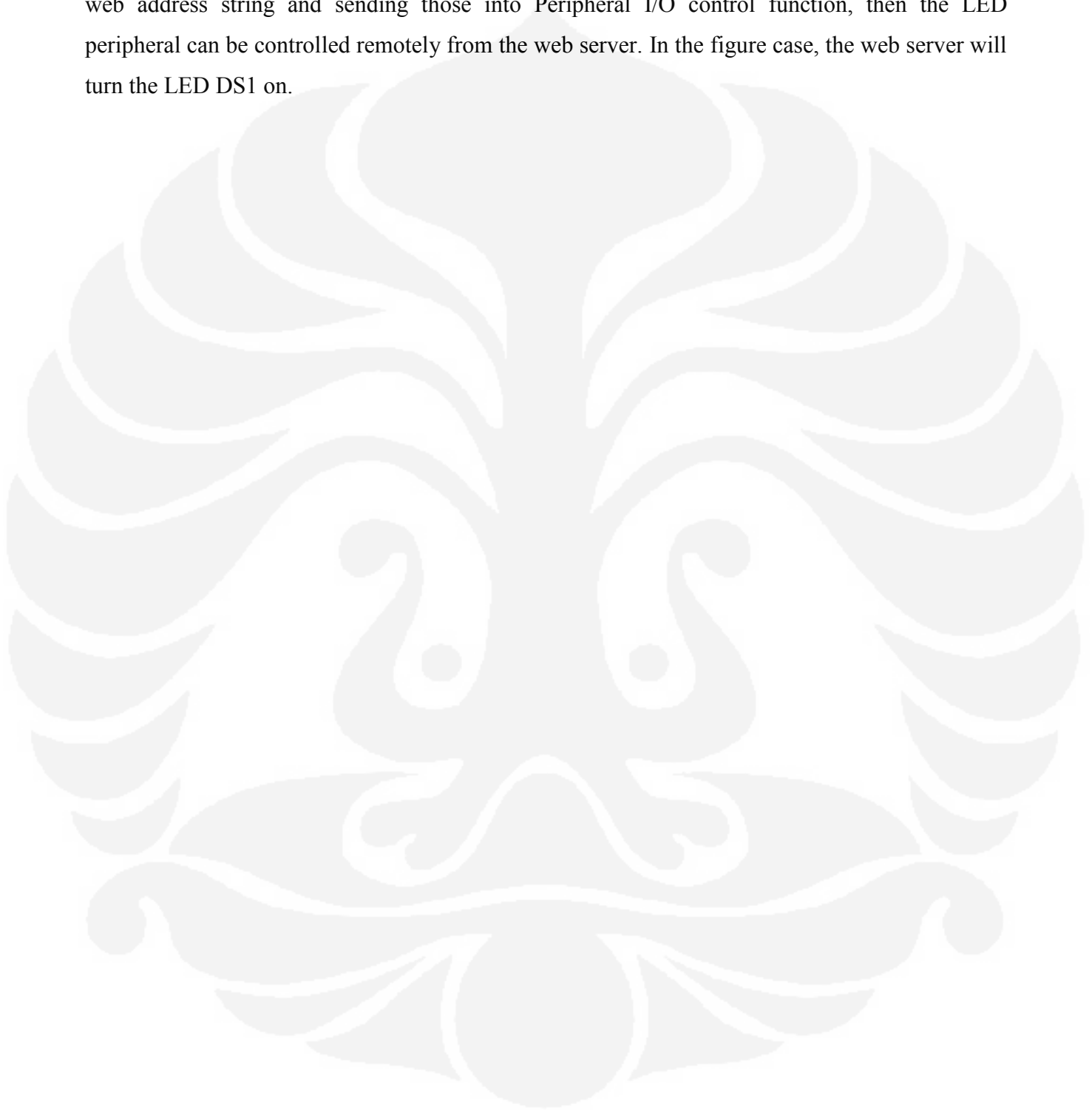
- The delay will be done by controlling the tick port in particular time.
- When calling this toggle function, first it will check the LED Data Status Register and then do the opposite operation from it. The operation will still work until there is an input from Input User Interface to stop calling this toggle function.

D. Transmitting buffer data in Web Server User interface as an input



Figure 5 - Web Address

uIP TCP/IP stack is allow us to control the web address as shown in figure 25. As we can see the value of LED and ON are 1 and 0 respectively. By separating these numbers from the web address string and sending those into Peripheral I/O control function, then the LED peripheral can be controlled remotely from the web server. In the figure case, the web server will turn the LED DS1 on.



Chapter 7 – Software Demonstration

The following operations are going to be demonstrated:

- Possible Network Diagrams
- ICMP and ARP
- Web Server Page RTOS Stats
- Web Server Page TCP Stats
- Web Server Page Connection
- Web Server Page IO

7.1 Possible Network Diagrams

There are some real world cases that can be implemented so that the microcontroller can be accessed remotely. The figures below show network diagrams that is suitable for the network implementation.

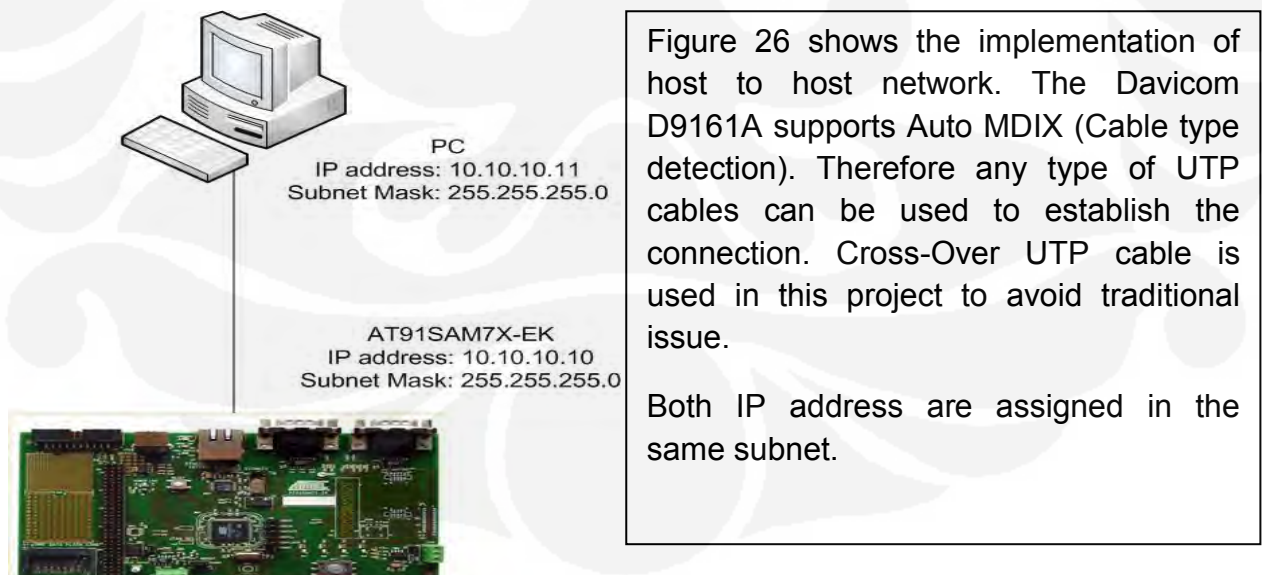
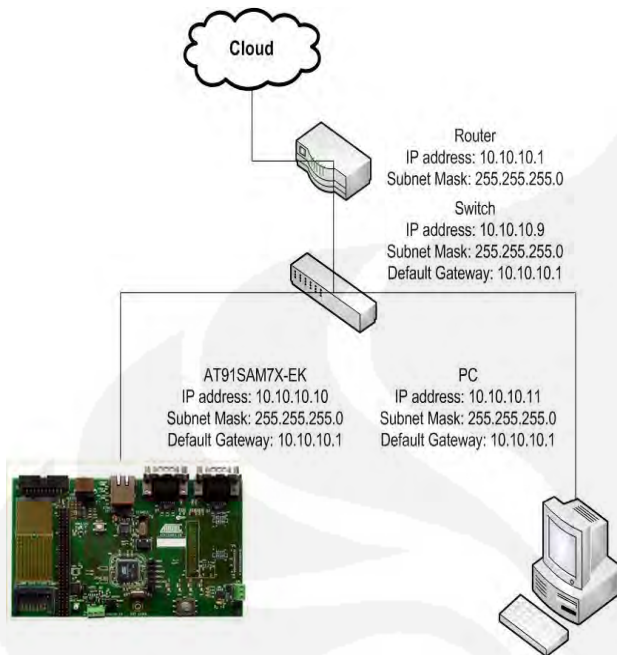


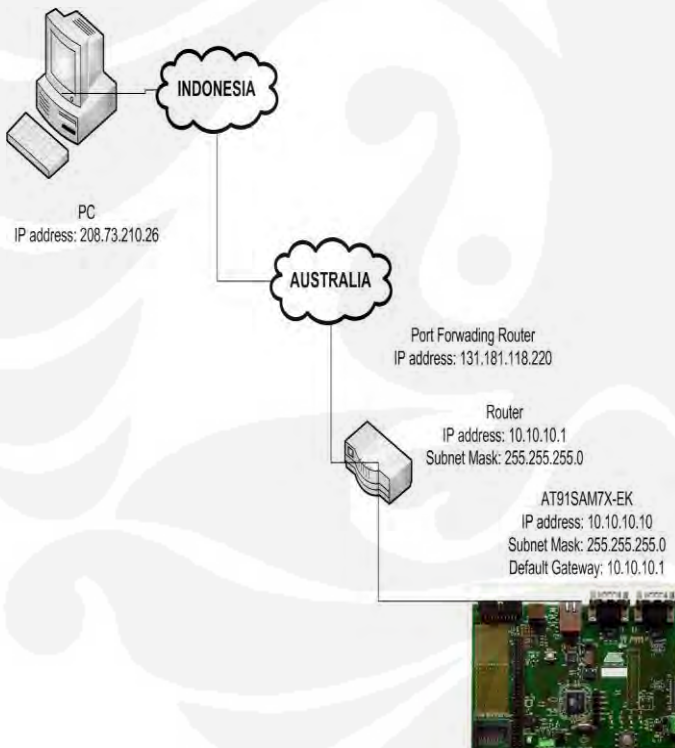
Figure 1 - Host to Host Network



By adding a switch into network diagram, it is still possible to access the embedded platform through Local Area Network.

All IP addresses must be in the same subnets so that all electronic devices can be communicated each other.

Figure 2 - Local Area Network



By adding a router as a gateway, student also could access the embedded platform from Wide Area Network.

The router should be configured to forward the embedded platform's port so that the microcontroller has a public/internet IP address.

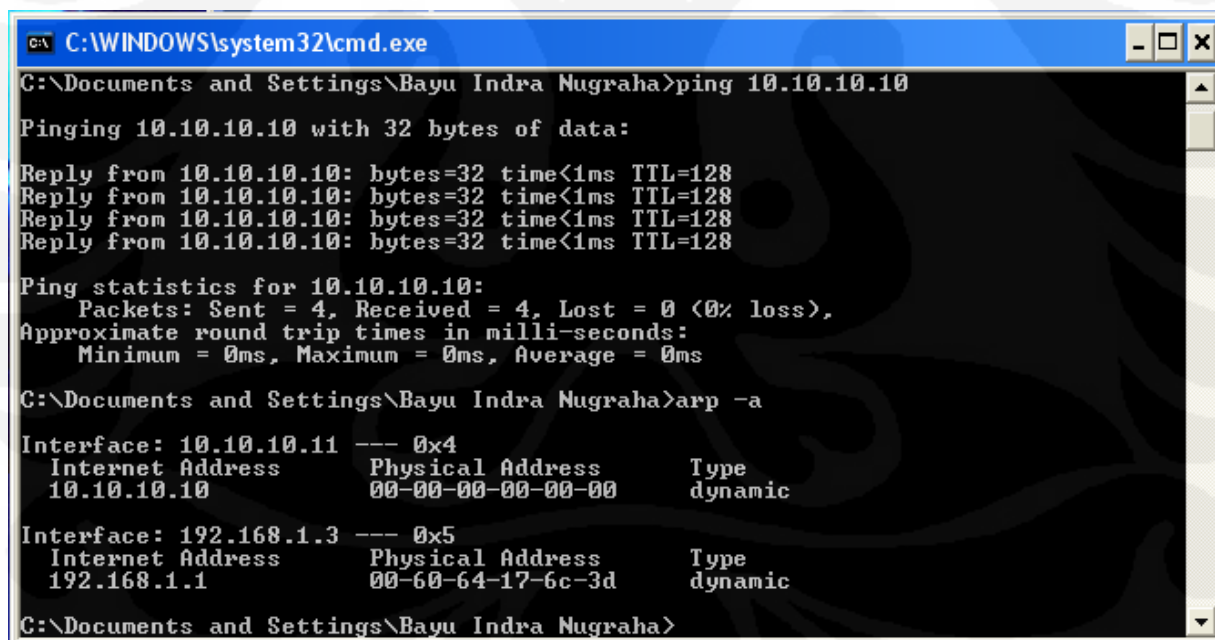
It means if the public IP address is accessed, basically it will point to the local embedded platform IP

Figure 3 - Wide Area Network

7.2 ICMP and ARP

Ping is a command that uses ICMP Protocol to check connectivity through network. It is important to check connectivity between a host and embedded platform before performing any other task. Note that if the ping command shows no connectivity, it simply means the packets cannot be delivered. The **Address Resolution Protocol (ARP)** is the method for finding a host's link layer (hardware) address when only its IP address or some other network layer address is known. So it simply means MAC address of the hardware can be detected.

In order to test the system, we can examine by using **ping** and **arp-a** command in Command Prompt Window. Figure 29 shows the connection between host and embedded platform was successful. However, the ARP shows that the MAC address of the embedded platform is 00:00:00:00:00:00 due to the combination error between FreeRTOS and uIPstack. As long student could just broadcast the packet, the packet will still delivered, but that will also clog our network with unnecessary broadcasts.



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Bayu Indra Nugraha>ping 10.10.10.10

Pinging 10.10.10.10 with 32 bytes of data:

Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128
Reply from 10.10.10.10: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.10.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Bayu Indra Nugraha>arp -a

Interface: 10.10.10.11 --- 0x4
    Internet Address      Physical Address      Type
    10.10.10.10           00-00-00-00-00-00    dynamic

Interface: 192.168.1.3 --- 0x5
    Internet Address      Physical Address      Type
    192.168.1.1           00-60-64-17-6c-3d    dynamic

C:\Documents and Settings\Bayu Indra Nugraha>
```

Figure 4 - ICMP and ARP Result

7.3 Web Server

Web Server Page – RTOS Stats

In this web server page, student put 3 statistics that represent the LED Output Status, Joystick Input status and Task statistic. Those statistics will be updated every 2 seconds instead of 1 second to minimize the network load. LED Output Status and Joystick Input Status will read and receive the Data Status Register from particular I/O peripheral bit. And, Task statistic is read from FreeRTOS's function called vTasklist() that show the FreeRTOS tasks that executed in running state.

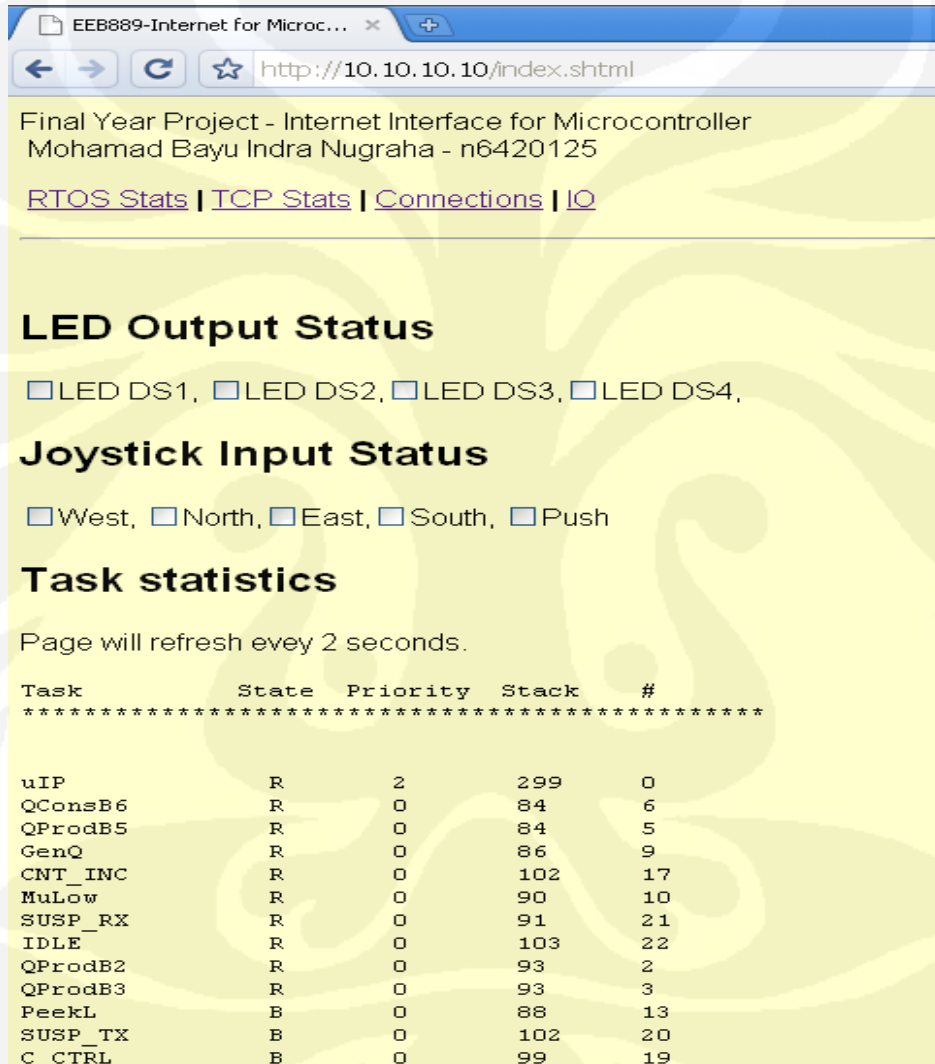


Figure 5 - Web Server - RTOS Page Stats

Web Server – TCP stats

This page shows the uIP statistic about the network performances that defines in uip_stats() in default. It determines the IP, ICMP, and TCP packets status from Ethernet traffic network.

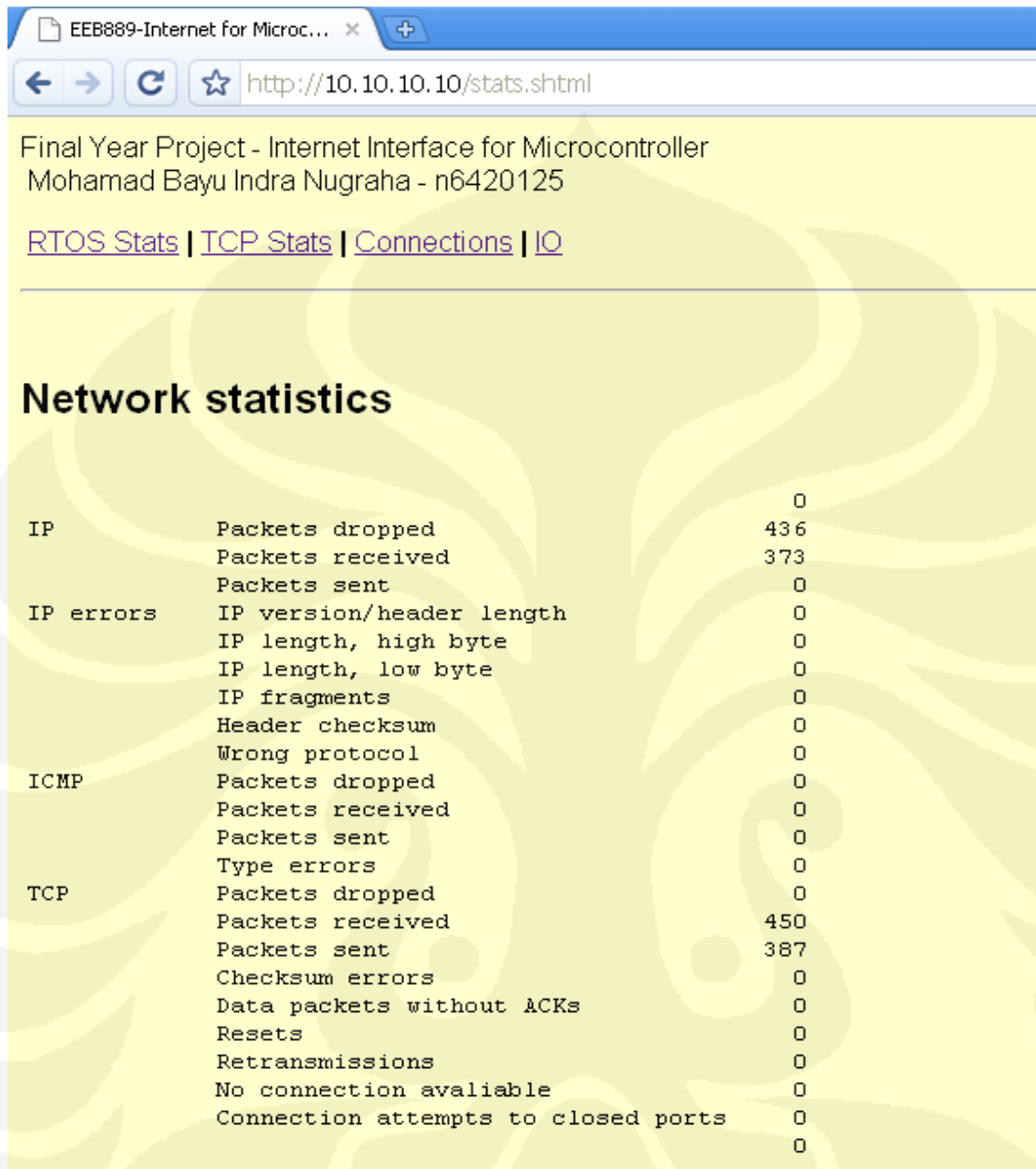


Figure 6 - Web Server - TCP Stats

Web Server – Connection Page

This page shows the embedded platform's network traffic and service discovery. It determines the statistic log of activity in Ethernet-based network.

Local	Remote	State	Retransmissions	Timer Flags
80	10.10.10.11:1799	TIME-WAIT	0	9
80	10.10.10.11:1802	TIME-WAIT	0	1
80	10.10.10.11:1806	ESTABLISHED	0	4

Figure 7 - Web Server - Connection Page

Web Server – IO Page

This page shows the GUI Interface that able to send buffers into the embedded platform by clicking the ‘update IO’ options. Student built two GUI interfaces to send bytes to turn on/off and toggle LED. The LED status can be seen in RTOS stats page.

LED IO

Use the check box to set the LED state, then click "Update IO" to send the new state to the microcontroller.

LED DS1 ▼ Turn On ▼ Update IO

Toggle DS1 ▼ 0 Update IO

Figure 8 - Web Server - IO Page

Summaries and Conclusion

Atmel's AT91SAM7X256 is a member of a series of highly integrated flash microcontrollers based on the 32-bit ARM RISC processor including an 802.3 Ethernet MAC. Therefore we can design and program the platform to be accessed through a web server that can be used to control, transmit, and receive data to the Input Output peripherals at the board remotely. Evaluation of EMAC, Davicom DM9161A chip, and PIO Controller are needed to be synchronized with real time operating systems to ARM processor.

Transmission Control Protocol that referred at transport layer in TCP/IP reference model is used in this implementation. It provides reliable end-to-end delivery service including data transmission and flow control. Using the reliable service, there must not any data loss and the frame has to be reassembled in the right places and makes up for Internet Protocol's (IP) deficiencies.

GCC environment is decided to use in this project, therefore student use 2 GCC open-sources with C language as a basic demo application. There are:

1. FreeRTOS, a mini Real Time Kernel
2. uIP stack, TCP/IP stack that provides TCP/IP connectivity

FreeRTOS is used to implement a scale-able real time kernel that designed specifically for small embedded systems. It means that routine and some modules in the program implementation are based on this open-source. There are some advantages using FreeRTOS open-source are:

- Preemptive, cooperative and hybrid configuration options.
- Designed to be small, simple and easy to use.
- Very portable code structure predominantly written in C.
- Support both tasks and co-routines.
- Stack-overflow detection options.

The uIP stack that student used for TCP/IP stack is an open source that is intended to make it possible to communicate using TCP/IP protocol suite even on small 8-bit micro-controllers. uIP requires a few functions to be implemented specifically for the architecture to run based on Transmission Control Protocol. C language implementations are given as part of the uIP distribution.

As the result, Peripheral I/O can be controlled remotely from the web server by combining FreeRTOS and uIP stack. Furthermore, via testing, this web server worked stably. Data can be transmitted and received into embedded platform reliably and it is suitable for mini web LAN.

GUI Interface

Final Year Project Code\Demonstration\webserver\httpd-cgi.c

Network Stats

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

static unsigned short
generate_tcp_stats(void *arg)
{
    struct uip_conn *conn;
    struct httpd_state *s = (struct httpd_state *)arg;

    conn = &uip_conns[s->count];
    return sprintf((char *)uip_appdata, UIP_APPDATA_SIZE,

"|<td>%d</td><td>%u.%u.%u.%u:%u</td><td>%s</td><td>%u</td><td>%u</td><td>%
c %c</td></tr>\r\n",
        htons(conn->lport),
        htons(conn->ripaddr[0]) >> 8,
        htons(conn->ripaddr[0]) & 0xff,
        htons(conn->ripaddr[1]) >> 8,
        htons(conn->ripaddr[1]) & 0xff,
        htons(conn->rport),
        states[conn->tcpstateflags & UIP_TS_MASK],
        conn->nrtx,
        conn->timer,
        (uip_outstanding(conn)) ? '*':' ',
        (uip_stopped(conn)) ? '!':' ');
}
/*-----
*/
static
PT_THREAD(tcp_stats(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);

    for(s->count = 0; s->count < UIP_CONNS; ++s->count) {
        if((uip_conns[s->count].tcpstateflags & UIP_TS_MASK) != UIP_CLOSED) {
            PSOCK_GENERATOR_SEND(&s->sout, generate_tcp_stats, s);
        }
    }

    PSOCK_END(&s->sout);
}

|  |

```

uIP Stats – IP TCP ICMP Checksum

```

char *pcStatus4,*pcStatus3,*pcStatus2,*pcStatus1;
unsigned long ulString;

/*-----
*/
static unsigned short
generate_net_stats(void *arg)
{
    struct httpd_state *s = (struct httpd_state *)arg;
    return snprintf((char *)uip_appdata, UIP_APPDATA_SIZE,
                   "%5u\n", ((uip_stats_t *)&uip_stat)[s->count]);
}

static
PT_THREAD(net_stats(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);

    #if UIP_STATISTICS

        for(s->count = 0; s->count < sizeof(uip_stat) / sizeof(uip_stats_t);
            ++s->count) {
            PSOCK_GENERATOR_SEND(&s->sout, generate_net_stats, s);
        }

    #endif /* UIP_STATISTICS */

    PSOCK_END(&s->sout);
}
/*-----
*/

```


FreeRTOS Stats

```
extern void vTaskList( signed char *pcWriteBuffer );
static char cCountBuf[ 32 ];
long lRefreshCount = 0;
static unsigned short
generate_rtos_stats(void *arg)
{
    lRefreshCount++;
    sprintf( cCountBuf, "<p><br>Refresh count = %ld", lRefreshCount );
    vTaskList( uip_appdata );
    strcat( uip_appdata, cCountBuf );

    return strlen( uip_appdata );
}
/*-----*/

static
PT_THREAD(rtos_stats(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_rtos_stats, NULL);
    PSOCK_END(&s->sout);
}
/*-----*/
```

Input User Interface – LED Turn on/Off

```
char *Input1ON, *Input1OFF;
generate_input(void *arg)
{
    sprintf( uip_appdata,
        "<select name=\"LED\"><option value=\"0\">LED DS1</option><option
value=\"1\">LED DS2</option><option value=\"2\">LED DS3</option><option
value=\"3\">LED DS4</option></select><select name=\"ON\"><option
value=\"0\">Turn On</option><option value=\"1\">Turn
Off</option></select>", Input1ON, Input1OFF);

    return strlen( uip_appdata );
}

static
PT_THREAD(led_input(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_input, NULL);
}
```

```

    PSOCK_END(&s->sout);
}

```

Input User Interface – Toggle LED

```

generate_toggle(void *arg)
{
    sprintf( uip_appdata,
            "<select name=\"Tog\"><option value=\"0\">Toggle
DS1</option><option value=\"1\">Toggle DS2</option><option value=\"2\">Toggle
DS3</option><option value=\"3\">Toggle DS4</option></select><input
type=\"textarea\" name=\"De\" value=\"0\" %s>", Input1ON, Input1OFF);

    return strlen( uip_appdata );
}

static
PT_THREAD(led_toggle(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_toggle, NULL);
    PSOCK_END(&s->sout);
}

```

LED Stats

```

char *pcStatus4,*pcStatus3,*pcStatus2,*pcStatus1;
//extern unsigned long uxParTextGetLED( unsigned long uxLED );

static unsigned short generate_io_state( void *arg )
{
    if( GetLED(4) ){
        pcStatus4 = "checked";
    }
    else{
        pcStatus4 = "";
    }
    if( GetLED(3) ){
        pcStatus3 = "checked";
    }
    else{
        pcStatus3 = "";
    }
    if( GetLED(2) ){

```

```

        pcStatus2 = "checked";
    }
    else{
        pcStatus2 = "";
    }
    if(GetLED(1) ){
        pcStatus1 = "checked";
    }
    else{
        pcStatus1 = "";
    }

    sprintf( uip_appdata,
        "<input type=\"checkbox\" name=\"LED1\" value=\"1\" %s>LED DS1,
<input type=\"checkbox\" name=\"LED2\" value=\"1\" %s>LED DS2,<input
type=\"checkbox\" name=\"LED3\" value=\"1\" %s>LED DS3,<input
type=\"checkbox\" name=\"LED0\" value=\"1\" %s>LED DS4, "\
        "<p>",
        pcStatus1, pcStatus2, pcStatus3, pcStatus4 );

    return strlen( uip_appdata );
}

static PT_THREAD(led_io(struct httpd_state *s, char *ptr)
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_io_state, NULL);
    PSOCK_END(&s->sout);
}

```

Joystick Stats

```

char *North ,*South,*East,*West, *Push , *InputStatus;
static unsigned short generate_joystick( void *arg )
{
    if( GetJoystick(5) ){
        Push = "checked";
    }
    else{
        Push = "";
    }
    if( GetJoystick(4) ){
        East = "checked";
    }
    else{
        East = "";
    }
    if( GetJoystick(3) ){
        West = "checked";
    }
    else{
        West = "";
    }
}

```

```

    }
    if( GetJoystick(2) ){
        South = "checked";
    }
    else{
        South = "";
    }
    if(GetJoystick(1) ){
        North= "checked";
    }
    else{
        North = "";
    }

    sprintf( uip_appdata,
        "<input type=\"checkbox\" name=\"West\" value=\"1\" %s>West,
<input type=\"checkbox\" name=\"North\" value=\"1\" %s>North,<input
type=\"checkbox\" name=\"East\" value=\"1\" %s>East,<input type=\"checkbox\"
name=\"South\" value=\"1\" %s>South, <input type=\"checkbox\" name=\"Push\"
value=\"1\" %s>Push\"
        "<p>",
        West, North, East, South, Push );

    return strlen( uip_appdata );
}

static PT_THREAD(joystick_status(struct httpd_state *s, char *ptr))
{
    PSOCK_BEGIN(&s->sout);
    PSOCK_GENERATOR_SEND(&s->sout, generate_joystick, NULL);
    PSOCK_END(&s->sout);
}

```

Reading Web Address and Separate value we want

```
/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

int nLEDNum;
int nLEDOn;

void vProcessInput( char *pcInput )
{
char *c,*LED, *Temp, *ON, *To, *De;
//ToggleInput LED;
/* Turn the LED on or off depending on the checkbox status. */

/*Check the address website*/
c = strstr( pcInput, "?" );
LED= strstr( pcInput, "LED=");
ON = strstr( pcInput, "ON=");
To = strstr( pcInput, "Tog=");
De = strstr( pcInput, "De=");
if( c ){
    if(LED && ON){
        nLEDNum = atoi(LED+4);
        nLEDOn = atoi(ON+3);
        SetLED(nLEDNum,nLEDOn);
    }
    if(To && De){
        nLEDNumber = atoi(To+4);
        nTickRate = atoi(De+3);
    }
}
}
```

Delay to Toggle LEDs

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

void vApplicationTickHook(void)
{
    if(nTickRate != 0){

static unsigned portLONG Count = 0, ErrorFound = pdFALSE;

/* The rate at which LEDs will toggle if an error has been found in one or
more of the standard demo tasks. */

const unsigned portLONG ErrorFlashRate = 1000*nTickRate / portTICK_RATE_MS;

/* The rate at which LEDs will toggle if no errors have been found in any
of the standard demo tasks. */

const unsigned portLONG NoError = nTickRate*1000 / portTICK_RATE_MS;
Count++;
if( ErrorFound != pdFALSE )
{
    /* We have already found an error, so flash the LED with the
appropriate
frequency. */

    if( Count > ErrorFlashRate ){
        Count = 0;
        ToggleLED( nLEDNumber);
    }
}
else
{
    if( Count > NoError )
    {
        Count = 0;
        ToggleLED( nLEDNumber );
    }
}
}
}

```

Main Program

```

/* Standard includes. */
#include <stdlib.h>

/* Scheduler includes. */
#include "FreeRTOS.h"
#include "task.h"

/* Demo application includes. */
#include "parallelIO.h"
#include "uip_task.h"
#include "BlockQ.h"
#include "blocktim.h"
#include "flash.h"
#include "QPeek.h"
#include "dynamic.h"
#include "httpd.h"
#include "uIP_Task.h"
#include "httpd-cgi.h"

/* Priorities for the demo application tasks. */
#define UIP_PRIORITY ( tskIDLE_PRIORITY + 2 )
#define mainUSB_PRIORITY ( tskIDLE_PRIORITY + 2 )
)
#define mainBLOCK_Q_PRIORITY ( tskIDLE_PRIORITY + 1 )
#define mainFLASH_PRIORITY ( tskIDLE_PRIORITY + 2 )
#define mainGEN_QUEUE_TASK_PRIORITY ( tskIDLE_PRIORITY )

/* The task allocated to the uIP task is large to account for its use of the
sprintf() library function. Use of a cut down printf() library would allow
the stack usage to be greatly reduced. */
#define UIP_STACK ( configMINIMAL_STACK_SIZE * 6 )

/*-----*/
/*
 * Configure the processor for use with the Atmel demo board. Setup is
minimal
 * as the low level init function (called from the startup asm file) takes
care
 * of most things.
 */
static void HardwareInit( void );

/*-----*/
/*
 * Starts all the other tasks, then starts the scheduler.
 */
int main( void )
{

```

```

/* Setup any hardware that has not already been configured by the low
level init routines. */
HardwareInit();

/* Start the task that handles the TCP/IP and WEB server functionality.
*/
xTaskCreate( vuIP_Task, "uIP", UIP_STACK, NULL, UIP_PRIORITY, NULL );

/* Start the standard demo tasks. */
vStartBlockingQueueTasks( mainBLOCK_Q_PRIORITY );
vCreateBlockTimeTasks();
vStartLEDFlashTasks( mainFLASH_PRIORITY );
vStartGenericQueueTasks( mainGEN_QUEUE_TASK_PRIORITY );
vStartQueuePeekTasks();
vStartDynamicPriorityTasks();

/*Start the scheduler*/
vTaskStartScheduler();
/* We should never get here as control is now taken by the scheduler.
*/
return 0;
}
/*-----*/

static void HardwareInit( void )
{
    portDISABLE_INTERRUPTS();

    /* When using the JTAG debugger the hardware is not always initialised
to
the correct default state. This line just ensures that this does not
cause all interrupts to be masked at the start. */
    AT91C_BASE_AIC->AIC_EOICR = 0;

    /* Most setup is performed by the low level init function called from
the
startup asm file. */

    /* Enable the peripheral clock. */
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOA;
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOB;
    AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_EMAC;

    /* Initialise the LED outputs for use by application tasks.*/
    LEDInitialise();
    JoystickInitialise();
}

```


Parallel IO Controller

```

/*
Final Year Project
Internet Interface for Microcontroller
Mohamad Bayu Indra Nugraha
n6420125
*/

/* Scheduler includes. */
#include "FreeRTOS.h"

/* Demo application includes. */
#include "parallelIO.h"

/*-----
 * Simple parallel port IO routines for the LED's. LED's can be set, cleared
 * or toggled.
 *-----*/

/* Joystick inputs used*/
#define North      ( 1 << 21 ) /* PA21 */
#define South     ( 1 << 22 ) /* PA22 */
#define East      ( 1 << 23 ) /* PA23 */
#define West      ( 1 << 24 ) /* PA24 */
#define Push      ( 1 << 25 ) /* PA25 */
/*LED Outputs used*/
#define DS1       ( 1 << 19 ) /* PB19 */
#define DS2       ( 1 << 20 ) /* PB20 */
#define DS3       ( 1 << 21 ) /* PB21 */
#define DS4       ( 1 << 22 ) /* PB22 */

/*Put The I/O Parts into an Array to make me writing program easier*/
#define LEDs      ( nLED_Mask[ 0 ] | nLED_Mask[ 1 ] | nLED_Mask[ 2 ] |
nLED_Mask[ 3 ] )
#define Joystick  ( nJoy_Mask[ 0 ] | nJoy_Mask[ 1 ] | nJoy_Mask[ 2 ] |
nJoy_Mask[ 3 ] | nJoy_Mask[ 4 ])

/*Pointing each array to the particular bit*/
const unsigned portLONG nLED_Mask[ 4 ]= { DS1, DS2, DS3, DS4 };
const unsigned portLONG nJoy_Mask[ 5 ] = { North, South, East, West,
Push };

/* LED Initialization */
void LEDInitialise( void )
{
    /* Configure the PIO Lines corresponding to LED1 to LED4 to be outputs.
*/
    AT91C_BASE_PIOB->PIO_PER = LEDs; // Enable The PIOB Control
    AT91C_BASE_PIOB->PIO_OER = LEDs; // Output Enable Register
    /* Start with all LED's off. */
    AT91C_BASE_PIOB->PIO_SODR = LEDs; // Set Output Enable Register

```

```

}

/* Joystick Initialization */
void JoystickInitialise( void )
{
    /*Configure the PIO Lines correspondng to Joystick to be inputs*/
    AT91C_BASE_PIOA->PIO_PER = Joystick; // Enable The PIOA Control
    AT91C_BASE_PIOA->PIO_ODR = Joystick; // Output Disable Register to
closed the Joystick contact
    AT91C_BASE_PIOA->PIO_PPUER = Joystick; // Enabling Intenal pull-up
resistor
}

void SetLED( unsigned portBASE_TYPE nLED, signed portBASE_TYPE Value )
{
    /*Check if LED number is not bigger than 4*/
    if( nLED < ( portBASE_TYPE ) 4 )
    {
        /*if Value is not equal to zero, it will turn off the LED*/
        /*else it will turn on the LED*/
        if( Value ){
            AT91C_BASE_PIOB->PIO_SODR = nLED_Mask[ nLED ];
        }
        else{
            AT91C_BASE_PIOB->PIO_CODR = nLED_Mask[ nLED ];
        }
    }
}
/*-----*/

void ToggleLED( unsigned portBASE_TYPE nLED )
{
    /*Check if LED number is not bigger than 4*/
    if( nLED < ( portBASE_TYPE ) 4 ){
        if( AT91C_BASE_PIOB->PIO_PDSR & nLED_Mask[ nLED ] ){
            AT91C_BASE_PIOB->PIO_CODR = nLED_Mask[ nLED ];
        }
        else{
            AT91C_BASE_PIOB->PIO_SODR = nLED_Mask[ nLED ];
        }
    }
}
/*-----*/

/*This Function is to check the LED Status*/
unsigned portBASE_TYPE GetLED(int nLED)
{
    return !( AT91C_BASE_PIOB->PIO_PDSR & nLED_Mask[ nLED - 1 ] );
}
/*This Function is to check the Joystick Input Status*/
unsigned portBASE_TYPE GetJoystick(int nJoystick )
{
    return !(AT91C_BASE_PIOA->PIO_PDSR & nJoy_Mask [ nJoystick - 1]);
}

void ControlLEDfromJoystick (void)

```

```
{  
    /*if North is pressed, then LED DS2 will turn on*/  
    if(GetJoystick(1)){  
        SetLED(1,0);  
    }  
    /*if South is pressed, then LED DS4 will turn on*/  
    else if(GetJoystick(2)){  
        SetLED(3,0);  
    }  
    /*if West is pressed, then LED DS1 will turn on*/  
    else if(GetJoystick(3)){  
        SetLED(0,0);  
    }  
    /*if East is pressed, then LED DS3 will turn on*/  
    else if(GetJoystick(4)){  
        SetLED(2,0);  
    }  
    /*if Push is pressed, then All LEDs will turn off*/  
    else if(GetJoystick(5)){  
        SetLED(0,1);  
        SetLED(1,1);  
        SetLED(2,1);  
        SetLED(3,1);  
    }  
}
```