Lab 4. WiFi

I. Purpose

The purpose of this lab is to setup WiFi device driver with security, and control battery-powered three-wheeled mobile robot using remote keyboard commander on PC via WiFi.

II. Problem Statement

Problem 4. Remote keyboard commander via WiFi and Datagram Socket.
Implement a remote keyboard commander for the battery-powered three-wheeled mobile robot using WiFi.

We start from the simplest and perform step-by-step improvements.

Problem 4A. Complete TMR.
Complete TMR on PCB to include batteries, power circuit, and USB hub.

Problem 4B. Setup WiFi Device driver with security.
Setup WiFi device driver module with security for WiFi USB dongle.

Problem 4C. Test Stream Socket Example.
Test Server and Client example using stream socket.

Problem 4D. Test Datagram Socket Example.
Test Talker and Listener example with datagram socket.

Problem 4E. Remote Keyboard Commander to control TMR via WiFi.
Implement a remote keyboard commander on PC to control battery-powered three-wheeled mobile robot using WiFi and datagram socket.

III. Technical Backgrounds

First Week

A. Complete TMR

1. Power circuit including batteries

Purpose
During test, we usually require power supply operation with TMR floating.

For final test, we require battery operation with TMR on-floor.

Dual power design

Separate power for computer and actuator

Computer: Beaglebone, USB Hub.

Actuator: Servos, Lights.

Reduce noise on computer from current spike by motor, etc.

Select Batteries:

LiPo 3.7V x 2 = 7.4 V to computer: Beaglebone, USB hub

NiH 1.2V x 4 = 4.8 V to actuator: Servos, LEDs.

Power circuit

Input

Power supply 7.4V, 2 A and 5V 2A

Power supply connector

Two LiPo battery 7.4 V 2A & 4 NiMH battery.

Connectors for LiPi & NiMH batteries are required to detach when not in use.

Battery units

LiPo Batteries, Battery holder, Battery connector [Disconnect when not in use]

NiMH Batteries, Battery holder, Battery connector [Disconnect when not in use]

Switch

Power selections of power supply, LiPo batteries, or None.

Use 6P switch: Dual three positions

Dual column with 3 pins each: One for 7.4V power, another for 5V (or 4.8V) power.

Position	Function
Up	Select Batteries
Center	Off
Down	Select power supply [for long-time test with power wire]

DC/DC converter

7.4 V (LiPox2) to 5 V for computer.

Generate regulated 5V 2A power to Bone and servos from 7.4V 2A power input.

Connections

Power source	Input connector	Input selector	Power regulator
Power supply	Power supply connector -+		
No power	None	+- Toggle 3 pos. switch	DC/DC converter
LiPo batteries on holder	LiPo battery connector -+		7.4 V to 5 V

Note. Since we added DC/DC converter, the power supply is required to supply two outputs: +7.4V for CPU via DC/DC converter, +5V to Lights and Servos.

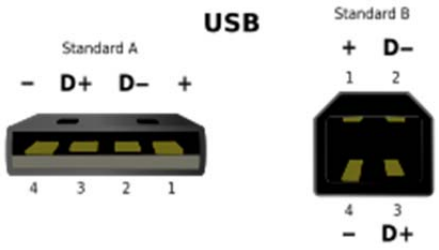
2. USB Hub

We require two USB devices: WiFi dongle and WebCam (in Lab 5), but Beaglebone has only one USB host port. Hence we add 4 port USB hub on PCB.

USB [http://en.wikipedia.org/wiki/USB]

Universal Serial Bus (**USB**) is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication, and power supply between computers and electronic devices.[2]

USB was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smartphones, PDAs and video game consoles.[3] USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.



USB Pinouts [USB, Wikipedia]
Pin 1 Vcc (+5V), Pin 2 Data-, Pin3 Data +, Pin4 Ground.

USB 1.x

Released in January 1996, USB 1.0 specified data rates of 1.5 Mbit/s (*Low Bandwidth* or *Low Speed*) and 12 Mbit/s (*Full Bandwidth* or *Full Speed*).

USB 2.0

USB 2.0 was released in April 2000, adding a higher maximum signaling rate of 480 Mbit/s called *High Speed.*

USB 3.0

USB 3.0 standard was released in November 2008, defining a new *SuperSpeed* mode.
The new *SuperSpeed* bus provides a fourth transfer mode with a data signaling rate of 5.0 Gbit/s.

B. Setup WiFi Device driver with security.

3. Hardware/software connection

PC	*Router*	*Bone*
Ethernet interface HW ---	wired -- + -- wired ---	Ethernet interface HW
Ethernet device driver SW (in kernel)	|	Ethernet device driver (in kernel)
	|	
	+ -- WiFi ---	WiFi USB dongle HW
		WiFi device driver (module)

4. WiFi

WiFi [http://en.wikipedia.org/wiki/Wi-Fi]

Wi-Fi, also spelled **Wifi** or **WiFi**, is a popular technology that allows an electronic device to exchange data or connect to the internet wirelessly using radio waves. The name is a trademark name, and was stated to be a play on the audiophile term Hi-Fi. The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards". However, since most modern WLANs are based on these standards, the term "Wi-Fi" is used in general English as a synonym for "WLAN". Only Wi-Fi products that complete Wi-Fi Alliance interoperability certification testing successfully may use the "Wi-Fi CERTIFIED" trademark.

Many devices can use Wi-Fi, e.g., personal computers, video-game consoles, smartphones, some digital cameras, tablet computers and digital audio players. These can connect to a network resource such as the Internet via a wireless network access point. Such an access point (or hotspot) has a range of about 20 meters (65 feet) indoors and a greater range outdoors. Hotspot coverage can comprise an area as small as a single room with walls that block radio waves, or as large as many square miles achieved by using multiple overlapping access points.

Uses

To connect to a Wi-Fi LAN, a computer has to be equipped with a wireless network interface controller. The combination of computer and interface controller is called a *station*. All stations share a single radio frequency communication channel. Transmissions on this channel are received by all stations within range. The hardware does not signal the user that the transmission was delivered and is therefore called a best-effort delivery mechanism. A carrier wave is used to transmit the data in packets, referred to as "Ethernet frames". Each station is constantly tuned in on the radio frequency communication channel to pick up available transmissions.

Range

The Wi-Fi signal range depends on the frequency band, radio power output, antenna gain, and antenna type. Line-of-sight is the thumbnail guide but reflection and refraction can have a significant impact. An Access Point compliant with either 802.11b or 802.11g, using the stock antenna might have a range of 100 m (330 ft).

Hardware

A wireless access point (WAP) connects a group of wireless devices to an adjacent wired LAN. An access

point resembles a network hub, relaying data between connected wireless devices in addition to a (usually) single connected wired device, most often an Ethernet hub or switch, allowing wireless devices to communicate with other wired devices.

Wireless adapters allow devices to connect to a wireless network. These adapters connect to devices using various external or internal interconnects such as PCI, miniPCI, USB, ExpressCard, Cardbus and PC Card. As of 2010, most newer laptop computers come equipped with built in internal adapters.

5. USB Wireless Adapter

We use TL-WL722N from TP-Link. It is a 150Mbps Wireless High Gain USB Adapter.
http://www.tp-link.pt/download/TL-WN722N.html

Signal Flow

Beaglebone		WL722N		WiFi Router
USB Host	– (USB signal) -	USB Client	– (Wireless signals) –	Wireless Access Point

6. WiFi Security

Wireless security [http://en.wikipedia.org/wiki/Wireless_security]

Wireless security is the prevention of unauthorized access or damage to computers using wireless networks. The most common types of wireless security are Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA).

WEP Wired Equivalent Privacy

WEP is a notoriously weak security standard. The password it uses can often be cracked in a few minutes with a basic laptop computer and widely available software tools. WEP is an old IEEE 802.11 standard from 1999 which was outdated in 2003 by WPA or Wi-Fi Protected Access.

WPA Wi-Fi Protected Access

WPA was a quick alternative to improve security over WEP. The current standard is WPA2; some hardware cannot support WPA2 without firmware upgrade or replacement. WPA2 uses an encryption device which encrypts the network with a 256 bit key; the longer key length improves security over WEP.

WPA2PSK

http://www.webopedia.com/TERM/W/WPA2_PSK.html

Short for **W***i-Fi* **P***rotected* **A***ccess* **2** - **P***re-***S***hared* **K***ey*, and also called WPA or *WPA2 Personal*, it is a method of securing your network using WPA2 with the use of the optional Pre-Shared Key (PSK) authentication, which was designed for home users without an enterprise authentication server.

To encrypt a network with WPA2-PSK you provide your router not with an encryption key, but rather with a plain-English passphrase between 8 and 63 characters long. Using a technology called TKIP (for Temporal Key Integrity Protocol), that passphrase, along with the network SSID, is used to generate unique encryption keys for each wireless client. And those encryption keys are constantly changed. Although WEP also supports passphrases, it does so only as a way to more easily create static keys, which are usually comprised of the

hex characters 0-9 and A-F.

TKIP Temporal Key Integrity Protocol

This stands for Temporal Key Integrity Protocol and the acronym is pronounced as tee-kip. This is part of the IEEE 802.11i standard. TKIP implements per-packet key mixing with a re-keying system and also provides a message integrity check. These avoid the problems of WEP.

AES Advanced Encryption Standard

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

The **Advanced Encryption Standard** (**AES**) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.[4]

AES is based on the Rijndael cipher[5] developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST during the AES selection process.[6] Rijndael is a family of ciphers with different key and block sizes.

For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.

AES has been adopted by the U.S. government and is now used worldwide. It supersedes the Data Encryption Standard (DES),[7] which was published in 1977. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

CCMP (Counter Mode Cipher Block Chaining Message Authentication Code Protocol, Counter Mode CBC-MAC Protocol)

http://en.wikipedia.org/wiki/CCMP

CCMP (**CCM mode Protocol**) is an encryption protocol designed for Wireless LAN products that implement the standards of the IEEE 802.11i amendment to the original IEEE 802.11 standard. CCMP is an enhanced data cryptographic encapsulation mechanism designed for data confidentiality and based upon the Counter Mode with CBC-MAC (CCM) of the AES standard.[1] It was created to address the vulnerabilities presented by WEP, a dated, insecure protocol.[1]

Note. In the ipTime Routers, WPA2PSK with AES is recommended.

C. Stream Socket

7. Stream socket

Stream socket

https://en.wikipedia.org/wiki/Stream_socket

In computer operating systems, a stream socket is a type of interprocess communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data without record boundaries, with well-defined mechanisms for creating and destroying connections and for detecting errors.

A stream socket transmits data reliably, in order, and with out-of-band capabilities.

In internet networks, stream sockets are typically implemented on top of TCP so that applications can run

across any networks using TCP/IP protocol. SCTP may also be used for stream sockets.

What is Socket?
http://beej.us/guide/bgnet/output/html/multipage/theory.html

You hear talk of "sockets" all the time, and perhaps you are wondering just what they are exactly. Well, they're this: a way to speak to other programs using standard Unix file descriptors.

What?

Ok—you may have heard some Unix hacker state, "Jeez, *everything* in Unix is a file!" What that person may have been talking about is the fact that when Unix programs do any sort of I/O, they do it by reading or writing to a file descriptor. A file descriptor is simply an integer associated with an open file. But (and here's the catch), that file can be a network connection, a FIFO, a pipe, a terminal, a real on-the-disk file, or just about anything else. Everything in Unix *is* a file! So when you want to communicate with another program over the Internet you're gonna do it through a file descriptor, you'd better believe it.

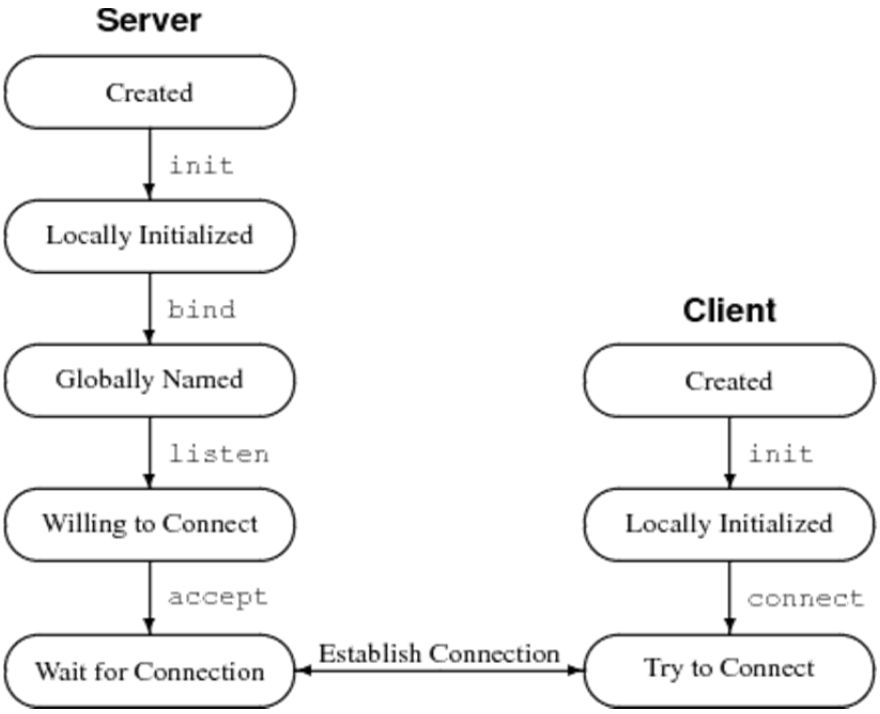
"Where do I get this file descriptor for network communication, Mr. Smarty-Pants?" is probably the last question on your mind right now, but I'm going to answer it anyway: You make a call to the socket() system routine. It returns the socket descriptor, and you communicate through it using the specialized send() and recv() (man send, man recv) socket calls.

"But, hey!" you might be exclaiming right about now. "If it's a file descriptor, why in the name of Neptune can't I just use the normal read() and write() calls to communicate through the socket?" The short answer is, "You can!" The longer answer is, "You can, but send() and recv() offer much greater control over your data transmission."

Stream sockets are reliable two-way connected communication streams. If you output two items into the socket in the order "1, 2", they will arrive in the order "1, 2" at the opposite end. They will also be error-free. I'm so certain, in fact, they will be error-free, that I'm just going to put my fingers in my ears and chant *la la la la* if anyone tries to claim otherwise.

8. System calls for socket

System calls - Initiating a Stream Connection
https://mozart.github.io/mozart-v1/doc-1.4.0/op/node12.html



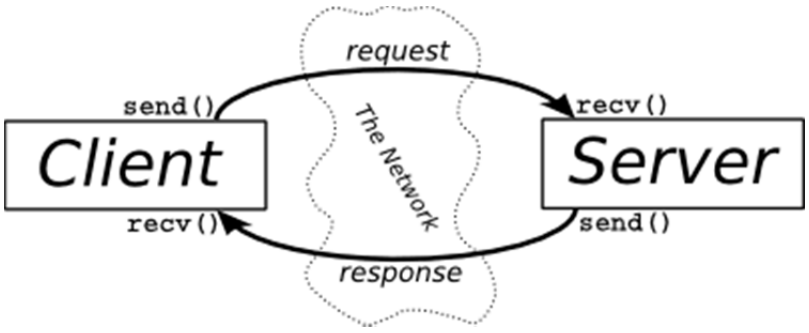
See Chapter 5. System calls or Burst
for comprehensive explanation of system calls for socket
in http://beej.us/guide/bgnet/output/html/multipage/theory.html

9. Client-Server background

http://beej.us/guide/bgnet/output/html/multipage/clientserver.html

It's a client-server world, baby. Just about everything on the network deals with client processes talking to server processes and vice-versa. Take telnet, for instance. When you connect to a remote host on port 23 with telnet (the client), a program on that host (called telnetd, the server) springs to life. It handles the incoming telnet connection, sets you up with a login prompt, etc.

The exchange of information between client and server is summarized as follows:



10. Client-server example program

See Section 6.1 & 6.2 in

http://beej.us/guide/bgnet/output/html/multipage/clientserver.html

Part 4. Datagram Socket

11. Datagram

Datagram socket [http://en.wikipedia.org/wiki/Datagram_socket]

A **datagram socket** is a type of connectionless network socket, which is the point for sending or receiving of packet delivery services.[1] Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may arrive in any order and might not arrive at the receiving computer.

UDP broadcasts sends are always enabled on a datagram socket. In order to receive broadcast packets, a datagram socket should be bound to the wildcard address. Broadcast packets may also be received when a datagram socket is bound to a more specific address.

Two Types of Internet Sockets

http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html#twotypes

Datagram sockets also use IP for routing, but they don't use TCP; they use the "User Datagram Protocol", or "UDP" (see RFC 768.)

Why are they connectionless? Well, basically, it's because you don't have to maintain an open connection as you do with stream sockets. You just build a packet, slap an IP header on it with destination information, and send it out. No connection needed. They are generally used either when a TCP stack is unavailable or when a few dropped packets here and there don't mean the end of the Universe. Sample applications: **tftp** (trivial file transfer protocol, a little brother to FTP), **dhcpcd** (a DHCP client), multiplayer games, streaming audio, video conferencing, etc.

"Wait a minute! **tftp** and **dhcpcd** are used to transfer binary applications from one host to another! Data can't be lost if you expect the application to work when it arrives! What kind of dark magic is this?"

Well, my human friend, **tftp** and similar programs have their own protocol on top of UDP. For example, the tftp protocol says that for each packet that gets sent, the recipient has to send back a packet that says, "I got it!" (an "ACK" packet.) If the sender of the original packet gets no reply in, say, five seconds, he'll re-transmit the packet until he finally gets an ACK. This acknowledgment procedure is very important when implementing reliable SOCK_DGRAM applications.

For unreliable applications like games, audio, or video, you just ignore the dropped packets, or perhaps try to cleverly compensate for them. (Quake players will know the manifestation this effect by the technical term: *accursed lag*. The word "accursed", in this case, represents any extremely profane utterance.)

Why would you use an unreliable underlying protocol? Two reasons: speed and speed. It's way faster to fire-and-forget than it is to keep track of what has arrived safely and make sure it's in order and all that. If you're sending chat messages, TCP is great; if you're sending 40 positional updates per second of the players in the world, maybe it doesn't matter so much if one or two get dropped, and UDP is a good choice.

What's the difference between sockets (stream) vs sockets (datagrams)? Why use one over the other?

http://stackoverflow.com/questions/4688855/whats-the-difference-between-streams-and-datagrams-in-network-programming

A stream socket is like a phone call -- one side places the call, the other answers, you say hello to each other (SYN/ACK in TCP), and then you exchange information. Once you are done, you say goodbye (FIN/ACK in TCP). If one side doesn't hear a goodbye, they will usually call the other back since this is an unexpected event; usually the client will reconnect to the server. There is a guarantee that data will not arrive in a different order than you sent it, and there is a reasonable guarantee that data will not be damaged.

A datagram socket is like passing a note in class. Consider the case where you are not directly next to the person you are passing the note to; the note will travel from person to person. It may not reach its destination, and it may be modified by the time it gets there. If you pass two notes to the same person, they may arrive in an order you didn't intend, since the route the notes take through the classroom may not be the same, one person might not pass a note as fast as another, etc.

So you use a stream socket when having information in order and intact is important. File transfer protocols are a good example here. You don't want to download some file with its contents randomly shuffled around and damaged!

You'd use a datagram socket when order is less important than timely delivery (think VoIP or game protocols), when you don't want the higher overhead of a stream (this is why DNS is primarily a datagram protocol, so that servers can respond to many, many requests at once very quickly), or when you don't care too much if the data ever reaches its destination.

To expand on the VoIP/game case, such protocols include their own data-ordering mechanism. But if one packet is damaged or lost, you don't want to wait on the stream protocol (usually TCP) to issue a re-send request -- you need to recover quickly. TCP can take up to some number of minutes to recover, and for real-time protocols like gaming or VoIP even three seconds may be unacceptable! Using a datagram protocol like UDP allows the software to recover from such an event extremely quickly, by simply ignoring the lost data or re-requesting it sooner than TCP would.

VoIP is a good candidate for simply ignoring the lost data -- one party would just hear a short gap, similar to what happens when talking to someone on a cell phone when they have poor reception. Gaming protocols are often a little more complex, but the actions taken will usually be to either ignore the missing data (if subsequently-received data supercedes the data that was lost), re-request the missing data, or request a complete state update to ensure that the client's state is in sync with the server's.

12. System calls for datagram socket

Datagram socket calls:

- sendto()
- recvfrom()

See Section 5.8 in http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html#sendtorecv.

13. Datagram Socket Programming example

See Section 6.3 lsitener.c and talker.c
in http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html#sendtorecv

Part V. Remote Commander via WiFi

See V. Design.

IV. Equipment and Parts

1. Lab equipment

Router (Wired and Wifi capable)
IBM PC with Windows and Linux (Dual boot)
Embedded board Beaglebone with cables and 4GB SD

2. Electronic parts

ID	Part No	Description	Qty/ group	Unit price
41	Fairman 18650	LiPo Battery, 18650 type	2	5,600
42	18650 2 cells holder	18650 2 cells holder	1	1,000
43	NiMH	NiMH battery	4	
44	NiMH 4 cells holder	NiMH 4 cells battery holder	1	
45	EP-VD(30V)	Step-down DC-DC converter	1	12,600
46	MTS-203	Power toggle switch, three pos.	1	630
47	N100mini	ipTime Wireless USB Dongle	1	8,000
48	XH 400	4 port USB Hub	1	4,300
49	SC S2	2 Cell 18650 LiPo Charger	1	24,900
50	LCD-78	NiMH 4 cell charger	1	16,160

Note. In order to test with batteries, you need to charge batteries fully beforehand, using LiPo charger and NiMH charger.

V. Design

Pre-report for first week

1. Design Problem 4A. Complete TMR.

Power circuit contains the following:

- Two LiPo battery: Type 18650. Providing 3.7 V x 2 = 7.4 V, Current up to about 2 A.
- Two LiPo battery holder: Providing LiPos attach & detatch (for recharging LiPo's using LiPo charger).
- Four NiMH battery: Providing 1.2 V x 4 = 4.8 V, Current up to about 2 A.
- Four NiMH battery holder: Providing NiMHs attach & detatch (for recharging using NiMH charger).
- Power connector from power supply: Use 1x4 pin header. Pin 1, 4: Ground, Pin 2, 3: +7.4 V. Why? Reverse connection is allowed.

- Power connector from LiPo battery: Use 1x4 pin header also. The same pin assignments. Allow disconnecting LiPos when not in use.
- Power connector from NiMH battery: Use 1x4 pin header also. The same pin assignments. Allow disconnecting NiMH when not in use.
- Power toggle switch: Three position switch. Selects power source – Power supply or LiPo/NiMH battery. Center position exists – No power input.
- Step-down DC-DC converter: Provide 5V 2A output to Bone from Power supply or LiPos.

See Backgrounds for details.

Add USB hub.

2. **Search internet security and compare WEP and WPA2 (Problem 4B).**

Pre-report for second week

3. **Compare sockets: stream and datagram (Problem 4C/D).**

4. **Design Problem 4E. Remote keyboard commander**

Prepare remote keyboard commander program on PC and control TMR program on Bone, which communicate via network.

The following is a suggestion. You can design your own.

A. System architecture

Split functionality: Distributed system

PC Remote commander to Bone
Remote keyboard input using raw key input mode.
Send datagram command via Ethernet.

Router Handle routing from wired PC to wireless Bone.

Bone Actuation of TMR
Receive datagram command via WiFi.
React according to the command.

B. Design of command packet

Raw key input command on PC

Contains single key.
Assignment of keys: The same as Lab 3.

```
Q: LL          W: Forward     E: RL
```

```
A: Left       S: Stop       D: Right
Z: CCW        X: Backward   C: CW
```

'ESC' or 'Ctrl-D' key terminates.

Command packet design

Let's use command ASCII string composed of

command_id	Sequential integer starting from 1.
time	Elapsed time from the start of this program
	String composed of NNN.MMM with unit of sec & resolution of ms.
ivx ivy iw	Translational/angular velocity command (%)
rl ll	Light right & left command (binary 0 off, 1 on)]

A space separates each field

For example, if you type S, W, W, D keys sequentially, it generates three command packets:

```
"0 1.23    0 0 0  0 0"
"1 3.27   10 0 0  0 0"
"2 8.45   20 0 0  0 0"
"3 12.18  20 10 0  0 0"
```

Hence each command packet contains less than 30 ASCII characters: Relatively short but contains sufficient information.

C. Program design

Remote_Commander_PC.c on PC

0. Print title
1. Get argument of destination IP of TMR.
2. Init datagram socket.
3. Display Key input menu.
5. Loop
    A. Raw mode key input (without Enter key) – getch() fu.
    B. Break if ESC or Ctrl=D
    C. Manage speed to vx, vy, and w; and Lights to RL, LL.
    D. Generate command packet ASCII string.
    E. Send command packet via datagram socket packet via network to TMR.
    F. Print information: key and cmd.
    G. usleep 100 ms (Less load).

In A, use the file getche.c file edited in Lab1C.
You may use two source files: one for main and key processing, and the other for networking (similar to talker.c).

WiFi_Control_TMR.c on Bone

Pre-requisite: Acquire_Triple_PWMs.sh is executed a priori.
0. Print Title
1. Set control parameters – gain etc.
2. Init PWM sysfs.
3. Init GPIO_LED
4. Open datagram socket and bind
5. Loop
    A. Recv datagram socket cmdstr
    B. Parse command to variables
    C. TMR Kinematics to get wheel vel cmd (PWM duty in %)
    D. Control three PWM duties (in ns)
    E. Control lights
    F. Print info
    G. usleep 100 ms (Less load).
6. Stop PWMs
7. Close socket // Reverse order of 2, 3, and 4
8. Close GPIO_LED
9. Close PWM sysfs files

You may use two source files: one for control TMR, and the other for networking (similar to listener.c).

D. Design Notes

Port numbers

Use the same port "4950" for both program: The port used in talker/listener.

Use port "4960" for image tx/rx in Lab 5.

VI. Lab Procedures

We use Wired floating TMR and Wireless on-floor TMR Breadboard among three configurations below:

Configurations:	Breadboard	**Wired floating TMR**	**Wireless on-floor TMR**
TMR	None	Floating	On-floor
Power	Power supply	Power supply	Battery.
Ethernet	Wired	Wired	WiFi
USB cable	YES	N	N
USB Hub	N	Y	Y
Number of wires	P/E/U	P/E	None
Usage	Initial	Long-term I/O test	Short-term move test

First Week

A. Complete TMR

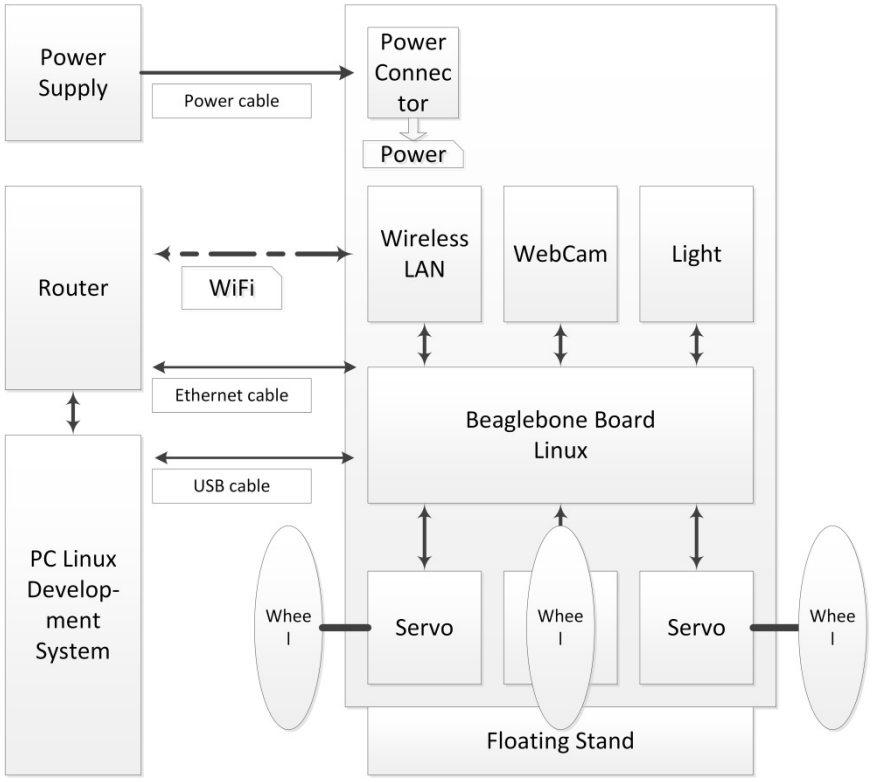
11. Wire Power circuit including batteries as prepared.

12. Place USB Hub on PCB.

B. Setup WiFi device driver with security

20. Setup hardware

Refer Fig 0.1 RoboCam - Wired.



Start Beaglebone

Refer Startup_Shutdown_Sequence.docx for startup and shutdown sequence.

21. Configure Network Router

Setup Router Encryption

PC with Ubuntu
Check IP of wired Ethernet using "ifconfig": For example, 192.168.100.12

On PC, login to WiFi Router in the Web as follows:
Launch a Web browser, such as FireFox.
In the address window, login to WiFi Router. For example
http://192.168.0.1
Note that, the IP address of WiFi Router depends on the initial setup. Find it from User's manual of your

router. Otherwise, configure the router.

In the Wireless LAN Management window, select authentication and encryption as
WPA2PSK + AES (Recommended)

Click 'Apply' button.

Note your router's network name (SSID) and WiFi network password.

Exit from the router.

22. FIND the WiFi device driver

Check the WiFi dongle used. Type

```
# lsusb
….
Bus 001 Device 004: ID 0cf3:9271 Atheros Communications, Inc. AR9271 802.11n
```

Check WiFi driver module with "lsmod"

```
# lsmod
Module                  Size  Used by
ath9k_htc              65054  0
ath9k_common            3070  1 ath9k_htc
ath9k_hw              358732  2 ath9k_common,ath9k_htc
ath                    14859  3 ath9k_common,ath9k_htc,ath9k_hw
mac80211              424813  1 ath9k_htc
cfg80211              354018  3 ath,mac80211,ath9k_htc
rfkill                 16672  1 cfg80211
….
```

Seems it is the right WiFI device driver module: ath9k_X.

23. Check WiFi configuration on Bone

Test ifconfig:

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr d4:94:a1:8b:0b:17
          inet addr:192.168.100.18  Bcast:192.168.100.255  Mask:255.255.255.0
……
wlan0     Link encap:Ethernet  HWaddr 64:e5:99:f4:db:54
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
……
```

This shows wired Ethernet connection with IP 192.168.100.18.
wlan0 is detected, but no IP is assigned.

Test iwconfig:

```
# iwconfig
wlan0     IEEE 802.11bgn  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
……
```

This also shows that wlan0 device driver is on, but IP is not yet configured.
We have to set IP for wlan0 WiFi.

24. Get WiFi utility tools

Get the **wireless-tools** package (see SynapticHowto).

```
# sudo apt-get install wireless-tools
… Already the newest version
```

It contains utilities: iwconfig, iwlist, etc.

wpa_passphrase utility will generate a 256-bit pre-shared WPA key from your ASCII passphrase. It is provided by the wpa_supplicant package.

If not installed, first thing is to install it:

```
# sudo apt-get install wpasupplicant
wpasupplicant is already the newest version.
```

25. Install WiFi reset service [Specific to Debian]

Refer WiFi Beaglebone Black
https://learn.adafruit.com/setting-up-wifi-with-beaglebone-black/configuration

WiFi reset service
The next step is to install a small script that will reset the WiFi interface by bringing it down and back up again automatically on boot. **With the latest 3.8 kernel I found this reset script was necessary to get reliable performance with Realtek and some Atheros WiFi adapters, so don't skip installing it!** Later kernels or more stable adapters might not require the reset service, but it can't hurt to install it for them too.

To install the service connect again to the device in a terminal with SSH and execute these commands:

```
# su
# cd ~
# ntpdate -b -s -u pool.ntp.org
# apt-get update && apt-get install git
# git clone https://github.com/adafruit/wifi-reset.git
Cloning into 'wifi-reset'...
# cd wifi-reset
```

```
# chmod +x install.sh
```

Run install.sh

```
# ./install.sh
Installing wifi reset service to /opt/wifi-reset.
Installing systemd service to run at boot.
Enabling systemd service.
```

Note that if you'd ever like to disable the WiFi reset service you can execute this command to do so:

```
# systemctl disable wifi-reset.service
```

Reboot Bone

```
# sudo reboot
```

26. Setup WPA for security

a. Generate the WPA key

Generate the WPA key with format:

```
# wpa_passphrase your_network_ssid your_passphrase
```

i.e.,

```
# wpa_passphrase iptime_bkkim_N604R <b8>
```

NOTE. Add "_" after wpa. <b8> means pre-configured WPA2 password of WiFi Router.

You get the reply such as:

```
network={
        ssid="iptime_bkkim_N604R"
        #psk="XXXXXXXX"                          // Changed to XX....
        psk=066fd14050032fb0085e56d828e5cc2cd36f2a1fb4f3842a686c614d08ece481
}
```

Store the psk long string to a file.

b. Edit /etc/network/interfaces as follows:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
```

```
iface eth0 inet dhcp
# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE

# WiFi Example
#auto wlan0
#iface wlan0 inet dhcp
#    wpa-ssid "essid"
#    wpa-psk  "password"

# Ethernet/RNDIS gadget (g_ether)
# ... or on host side, usbnet and random hwaddr
# Note on some boards, usb0 is automaticly setup with an init script
# in that case, to completely disable remove file [run_boot-scripts] from the boot pa
rtition
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.0
    network 192.168.7.0
    gateway 192.168.7.1

######### Appended for wlan0 WiFi
auto wlan0
iface wlan0 inet dhcp
    # Specify your Router SSID correctly:
    wpa-ssid “iptime_bkkim_N604R”
    # Specify your WiFi password with the string from wpa_passphrase correctly:
    wpa-psk XXXX……XXXX

    ## The followings seems optional, but helpful
    # wpa-driver
    wpa-driver wext
     # if your SSID is hidden, change value to 2
    wpa-ap-scan 1
    # type WPA for WPA1, RSN for WPA2
    wpa-proto RSN
    # type CCMP for AES, TKIP for TKIP
    wpa-pairwise CCMP
    # type CCMP for AES, TKIP for TKIP
    wpa-group CCMP
    # type WPA-PSK for shared key (most common), WPA-EAP for enterprise radius server
    wpa-key-mgmt WPA-PSK
```

c. Edit Wpa_supplicant.conf

Refer http://linux.die.net/man/5/wpa_supplicant.conf

Some people have found that this doesn't always work, so the next thing to do is to edit the configuration file for the wpasupplicant program. Do this by issuing:

```
# sudo vi  /etc/wpa_supplicant.conf
```

Basically, you add pretty much the same information here as you did to the interfaces file, except without the wpa- part. So, my file looks like this:

```
ap_scan=1
ctrl_interface=DIR=/var/run/wpa_supplicant
network={
    ssid="iptime_bkkim__N604R"
    scan_ssid=0
    psk=XXXX……XXXX    // Set to string from wpa_passphrase
    key_mgmt=WPA-PSK
    proto=RSN
    pairwise=CCMP
    group=CCMP
}
```

NOTE. Add Quotation on ssid. No Quote on psk.
/var/run/wpa_supplicant may not exist. Ok. It will be created and used.

27. Start WiFi driver manually

If wlan0 is already up, stop it manually

```
#ifdown wlan0
```

Start WiFi driver manually

```
# ifup wlan0
ioctl[SIOCSIWENCODEEXT]: Invalid argument
ioctl[SIOCSIWENCODEEXT]: Invalid argument
Internet Systems Consortium DHCP Client 4.2.2
Copyright 2004-2011 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlan0/f8:1a:67:0f:79:9e
Sending on   LPF/wlan0/f8:1a:67:0f:79:9e
Sending on   Socket/fallback
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 3
DHCPDISCOVER on wlan0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on wlan0 to 255.255.255.255 port 67
DHCPOFFER from 192.168.100.1
DHCPACK from 192.168.100.1
```

```
bound to 192.168.100.17 -- renewal in 2807 seconds.
```

Check

```
# ll /var/run/wpa_supplicant
total 0
drwxr-x---  2 root root  60 Feb  7 03:00 ./
drwxr-xr-x 19 root root 740 Feb  7 03:00 ../
srwxrwx---  1 root root   0 Feb  7 03:00 wlan0=
```

Check if IP of wlan0 is established.

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr d4:94:a1:86:bc:b6
          inet addr:192.168.100.18  Bcast:192.168.100.255  Mask:255.255.255.0
….....
wlan0     Link encap:Ethernet  HWaddr 64:e5:99:f4:db:54
          inet addr:192.168.100.17  Bcast:192.168.100.255  Mask:255.255.255.0
……
```

We got IP of WiFi as 192.168.100.17!

28. Test automatic WiFi setup on power on

Restart Bone

```
# sudo reboot
```

Ssh again, for example,

```
$ ssh bkkim@192.168.100.18
```

Enter password.
Or use minicom.

Check if IP of wlan0 is automatically established.

```
# ifconfig
….....
wlan0     Link encap:Ethernet  HWaddr 64:e5:99:f4:db:54
          inet addr:192.168.100.17  Bcast:192.168.100.255  Mask:255.255.255.0
```

Note. May require setup time of one minute or more.

29. Re-configure NFS server

Now Beaglebone has two network connections:

Ethernet	with IP 192.168.100.18
WiFi	with IP 192.168.100.17

We need to re-configure NFS server on PC to allow NFS via WiFi also.
Modify file /etc/exports to allow both Ethernet and WiFi:

```
……
# Allow NFS to Bone via Ethernet
/home/bkkim/Lab    192.168.100.18(rw,sync,no_root_squash,no_subtree_check)
# Allow NFS to Bone via WiFi
/home/bkkim/Lab    192.168.100.17(rw,sync,no_root_squash,no_subtree_check)
```

Don't' forget to type

```
$ sudo exportfs –a
```

Do we need to re-configure NFS client?

C. Test Stream Socket

30. Create a working directory

```
$ mkdir -p DesignLab/4_WiFi/c_Stream_Socket
```

31. Get server.c and client.c

Visit http://beej.us/guide/bgnet/output/html/multipage/index.html
or http://beej.us/guide/bgnet/output/html/multipage/clientserver.html.

Go to Section 6.1. A Simple Stream Server
Download server.c

Go to Section 6.2. A Simple Stream Client
Download client.c

32. Make

Edit Makefile to generate

server_pc, client_pc	Executable on PC
server_bone, client_bone	Executable on Bone

Make

$ make

33. Test PC to PC

Run client without server

PC Term 1

PC term 2

```
$ ./client_pc 192.168.100.12
client: connect: Connection refused
client: failed to connect
```

This is due to server is not working.

Run server and then client.

PC Term 1	PC term 2
$ ./server_pc	
server: waiting for connections...	
	$ ./client_pc 192.168.100.12
	client: connecting to 192.168.100.12
server: got connection from 192.168.100.12	
	client: received 'Hello, world!'
	// One more client?
	$ ./client_pc 192.168.100.12
	client: connecting to 192.168.100.12
server: got connection from 192.168.100.12	
	client: received 'Hello, world!'

Notice that server_pc is still waiting for another client.
Note that you can use 127.0.0.1 (IP of myself) instead of PC IP.
Input Crtl+C to quit the server.

34. Test PC to Bone

Test server on Bone, client on PC, via wired Ethernet.
Assume

PC eth IP:	192.168.100.12
Bone eth IP:	192.168.100.18

Bone Term 1	PC term 2
# ./server_bone	
server: waiting for connections...	
	// Client via wired Ethernet
	$ ./client_pc 192.168.100.18
	client: connecting to 192.168.100.18
server: got connection from 192.168.100.12	
	client: received 'Hello, world!'

Test again server on Bone via WiFi, client on PC via Ethernet.
Assume

PC eth IP:	192.168.100.12
Bone wlan IP:	192.168.100.17

Bone Term 1 PC term 2

```
// Server on Bone
# ./server_bone
server: waiting for connections...
                                        //Client on PC via Ethernet & WiFi
                                        $ ./client_pc 192.168.100.17
                                        client: connecting to 192.168.100.17
server: got connection from 192.168.100.12
                                        client: received 'Hello, world!'
```

Note that we can get the same service using Ethernet or WiFi.
How is this possible?
The router can route eth to eth, and eth to wlan, and wlan to wlan.

D. Test UDP Example

40. Create a working directory

$ mkdir -p DesignLab/4_WiFi/d_Datagram_Socket

41. Get talker.c and listener.c

Get listener.c and talker.c

in Section 6.3, http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html#sendtorecv

42. Make

Edit Makefile to generate

listener_pc, talker_pc	Executable on PC
listener_bone, talker_bone	Executable on Bone

Make

$ make

43. Test PC to PC

PC Term 1 PC term 2

```
$ listener_pc
listener: waiting to recvfrom...
                                        $ ./talker_pc 192.168.100.12
                                        usage: talker hostname message

                                        $ ./talker _pc 192.168.100.12 "Hi,there!”
```

```
                                        talker: sent 10 bytes to 192.168.100.12
listener: got packet from 192.168.100.3
listener: packet is 10 bytes long
listener: packet contains "Hi, there!"
```

Note that talker_pc is stopped after receiving a packet.
Note that you can use 127.0.0.1 (IP of myself) instead of PC IP.

What happens if Talker is run without running Listener?

44. Test PC to Bone

Test server on Bone, client on PC, via wired Ethernet.
Assume

PC eth IP:	192.168.100.12
Bone eth IP:	192.168.100.18

Bone Term 1 PC term 2

```
# ./listener_bone
listener: waiting to recvfrom...
                                        $ ./talker_pc 192.168.100.18 "Can you hear
me?"
                                        talker: sent 16 bytes to 192.168.100.18

listener: got packet from 192.168.100.12
listener: packet is 16 bytes long
listener: packet contains "Can you hear me?"
```

Test again server on Bone, client on PC, via WiFi.
Assume

PC eth IP:	192.168.100.12
Bone wlan IP:	192.168.100.17

Bone Term 1 PC term 2

```
Listener on Bone
# ./listener_bone
listener: waiting to recvfrom...
                                        $ ./talker_pc 192.168.100.17 "How are you,
Bone?"
                                        talker: sent 18 bytes to 192.168.100.17

listener: got packet from 192.168.100.12
listener: packet is 18 bytes long
listener: packet contains "How are you, Bone?"
```

E. Remote Commander via WiFi and UDP

50. Make a working directory

Working directory: e_Remote_Commander

51. Build Remote_Commander_PC.c

Edit Remote_Commander_PC.c

Native compile to get Remote_Commander_PC.

52. Test Remote_Commander_PC.c

Test standalone.

Nobody is listening, but we can go on!

```
$ ./Remote_Commander_PC 192.168.100.18
    Remote_Commander_PC
  Key input menu (without Enter):
q: ll;   w: vx;    e: lr;
a: vy;   s: stop;  d: -vy;
z: +w;   x: -vx;   c; -w;
  Speed up/down with multiple key strokes.
'ESC' or 'Ctrl-D' key terminates.
Key 's'. cmd: 1 8.926 0 0 0 0 0
Key 'w'. cmd: 2 10.648 10 0 00 0 0
Key 'w'. cmd: 3 11.368 20 0 0 0 0
Key 'x'. cmd: 4 11.943 10 0 0 0 0
……
```

Lights control:

```
$ ./Remote_Commander_PC 192.168.100.18
    Remote_Commander_PC
  Key input menu (without Enter):
q: ll;   w: vx;    e: lr;
a: vy;   s: stop;  d: -vy;
z: +w;   x: -vx;   c; -w;
  Speed up/down with multiple key strokes.
'ESC' or 'Ctrl-D' key terminates.
Key 'q'. cmd: 1 1.877 0 0 0 0 1
Key 'e'. cmd: 2 2.166 0 0 0 1 1
Key 'q'. cmd: 3 2.574 0 0 0 1 0
Key 'e'. cmd: 4 2.846 0 0 0 0 0
```

53. Build WiFi_Control_TMR.c

Edit WiFi_Control_TMR.c.

Cross compile.

54. Test – Wired floating TMR

Start Bone

Run as superuser

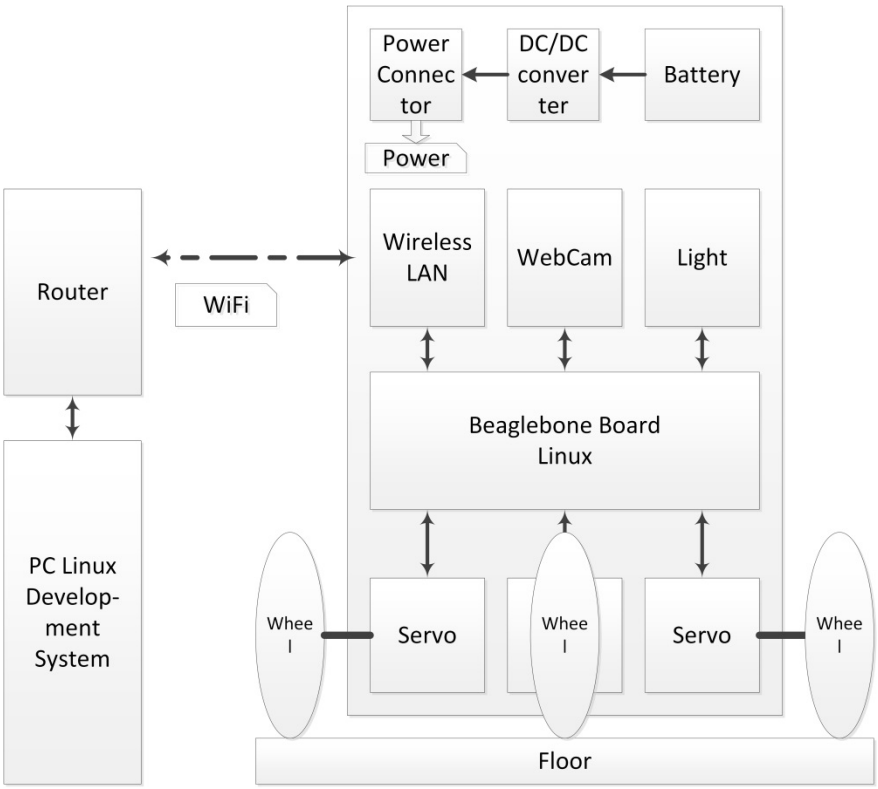
```
# su
# cd Lab/DesignLab/4_.../e_...www
# ./Acquire_triple_PWMs.sh

# ./WiFi_Control_TMR
…
… waiting to recvfrom
```

PC Term

```
# ./Keyboard_WiFi_PC
```

55. Test – On-floor MR (Battery operated, using WiFi)

Refer Fig 0.2.



Insert two charged LiPo batteries to LiPo holder on Bone TMR.
Locate Bone TMR on floor!

Operate Bone TMR with Battery.

Test WiFi connection: iwconfig, ping.

Run as superuser

```
# su
# cd Lab/DesignLab/4_.../e_...www
# ./Acquire_triple_PWMs.sh

# ./WiFi_Control_TMR
…
… waiting to recvfrom
```

PC Term

```
# ./Keyboard_WiFi_PC
```

If successful, *demonstrate to TA!*

VII. Final Report

Discussion for the following question should be included in the report.

1) What is Router? AP?

2) Compare TCP and UDP.

3) Compare Ethernet broadcast, multicast, and unicast.

4) Summarize purpose and usage of wpa-passphrase.

5) Set your own topic to discuss related to Lab 4, and explain summary of your search result.

VIII. References

[1] Beaglebone Rev A6 System Reference Manual, http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf

[2] AM3359 Datasheet, AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. J), http://www.ti.com/lit/ds/symlink/am3359.pdf

[3] Technical Reference Manual - AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. O), http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf