Lab 4. Final Report

Wi-Fi

EE405 Electronic Design Lab

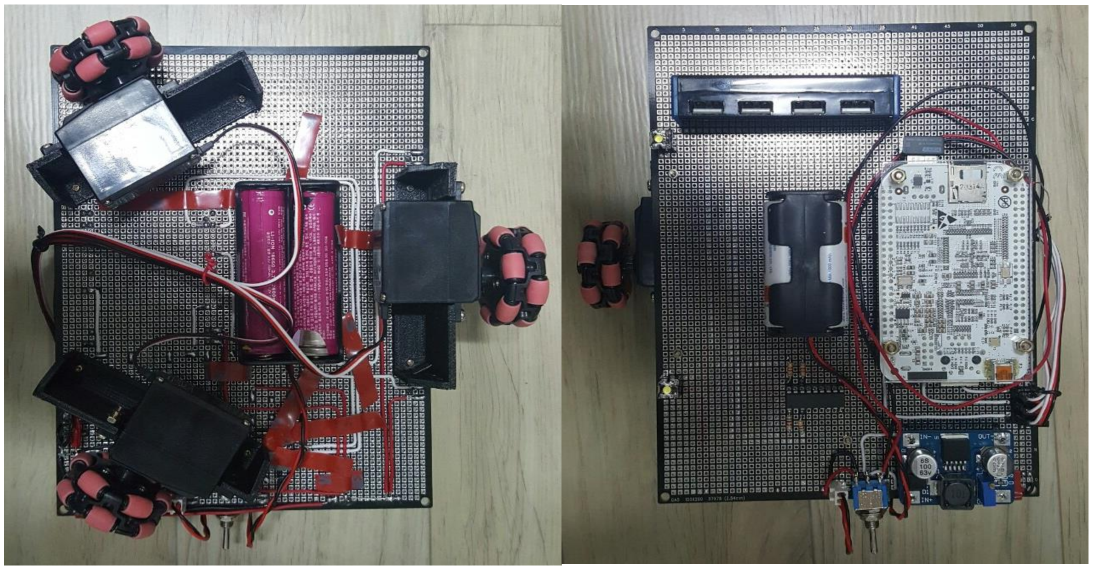
20154920 Wootae Song

11 May 2016

Table of Contents

Simple Procedures and Results combined

A. Complete TMR



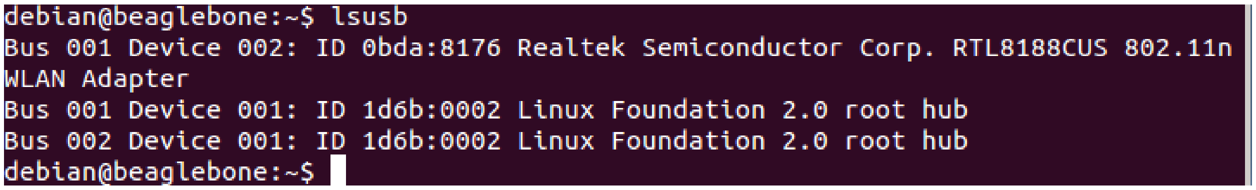
현재 회로에서 전원을 공급하는 것은 Li-Po Battery 2 개 7.4V 와 Ni-H Battery 4 개 4.8V 이다. Beaglebone 의 경우 버틸 수 있는 최대 전류가 낮고, Servo 의 경우 작동하다가 갑자기 전류가 크게 흐르는 current spike 현상이 나타날 수 있기 때문에 서로 다른 Battery 로 전원을 공급하여서 안정성을 도모했다. Beaglebone 은 5V 의 전원이 필요하기 때문에 Li-Po Battery 를 사용하되, DC/DC Converter 를 이용해 7.4V 를 5V 로 전환하여 공급한다. Servo 와 LED 의 경우 Ni-H Battery 가 4.8V 이기 때문에 그대로 전원을 공급하여 준다. Servo 는 전선이 흰색, 빨간색, 검은색이 존재하는 데, 이는 각각 PWM 신호, 5V, GND 를 연결해주어야 함을 의미한다. LED 의 경우는 LAB2. Light Control 에서 설계했던 회로 그대로 사용하였다. Prelab 4-1 의 설계도면과 다른 점은 Common GND 로 서로 다른 Battery 의 GND 를 연결해주지 않았다는 것인데, 이는 GND 를 연결하면

Beaglebone 의 안전을 위해 서로 다른 Battery 로 전원을 주었던 것이 소용이 없기 때문이다. 이 외에는 전부 동일하다.

B. Setup WiFi device driver with security

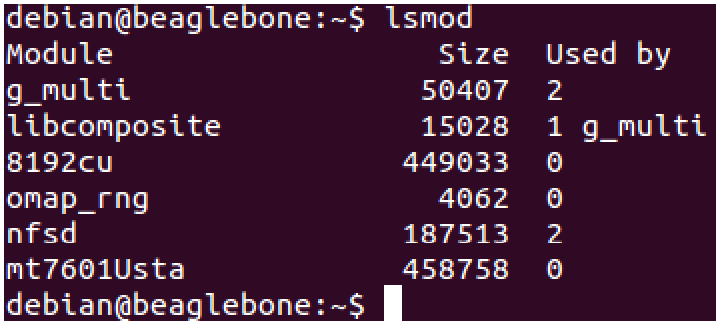
21.

In this section, we connect our Beaglebone to Wi-Fi network. We first plugged in Wi-Fi dongle to the Beaglebone's USB port. When I typed lsusb command, I got the following result:



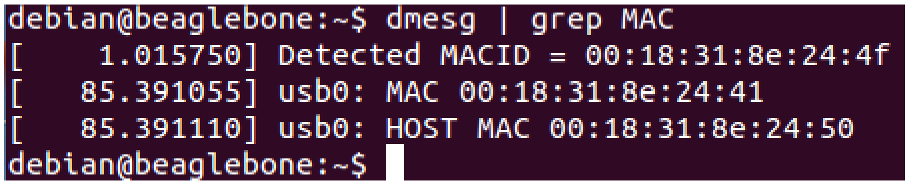
```
debian@beaglebone:~$ lsusb
Bus 001 Device 002: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS 802.11n
WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
debian@beaglebone:~$
```

As you can see in the picture above, our Beaglebone recognized the Wi-Fi dongle. The module for the Wi-Fi dongle was already installed and we could see it by typing lsmod.



```
debian@beaglebone:~$ lsmod
Module                  Size  Used by
g_multi                50407  2
libcomposite           15028  1 g_multi
8192cu                449033  0
omap_rng                4062  0
nfsd                  187513  2
mt7601Usta            458758  0
debian@beaglebone:~$
```

8192cu is the module name for our Wi-Fi dongle. Since our WiFi driver is a new one, dmesg command did not show the MAC address for our Wi-Fi dongle:



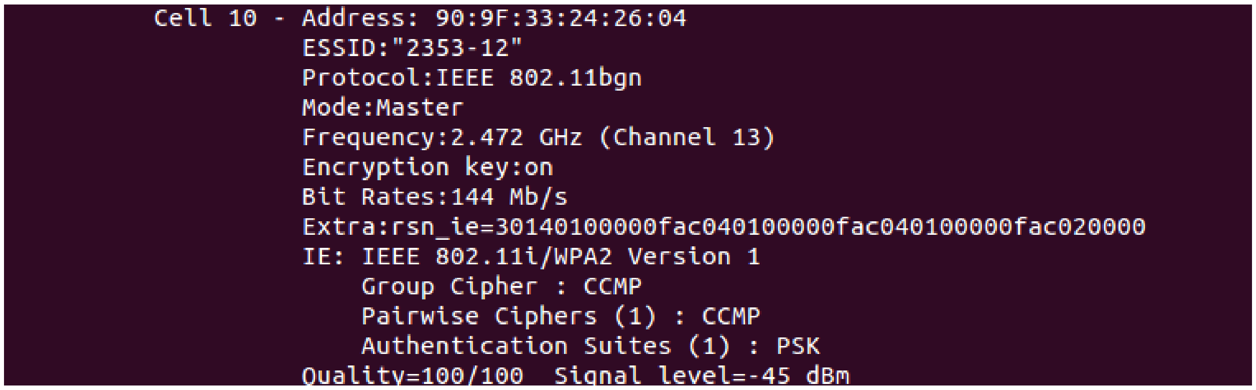
```
debian@beaglebone:~$ dmesg | grep MAC
[    1.015750] Detected MACID = 00:18:31:8e:24:4f
[   85.391055] usb0: MAC 00:18:31:8e:24:41
[   85.391110] usb0: HOST MAC 00:18:31:8e:24:50
debian@beaglebone:~$
```

22.

After plugging the Wi-Fi dongle, making sure that our Beaglebone recognize it, and Wi-Fi driver is installed, we checked whether we have Wi-Fi access on our Beaglebone. We gave ifconfig command. Although Beaglebone had field for wlan, no IP address was configured to it yet. Also, "iwconfig" command displayed that it couldn't find an accessing point. We got the same result as the lab text for both commands.

24. Check WiFi Access Points

We checked available WiFi access points by typing iwlist scanning command. We got the following results:

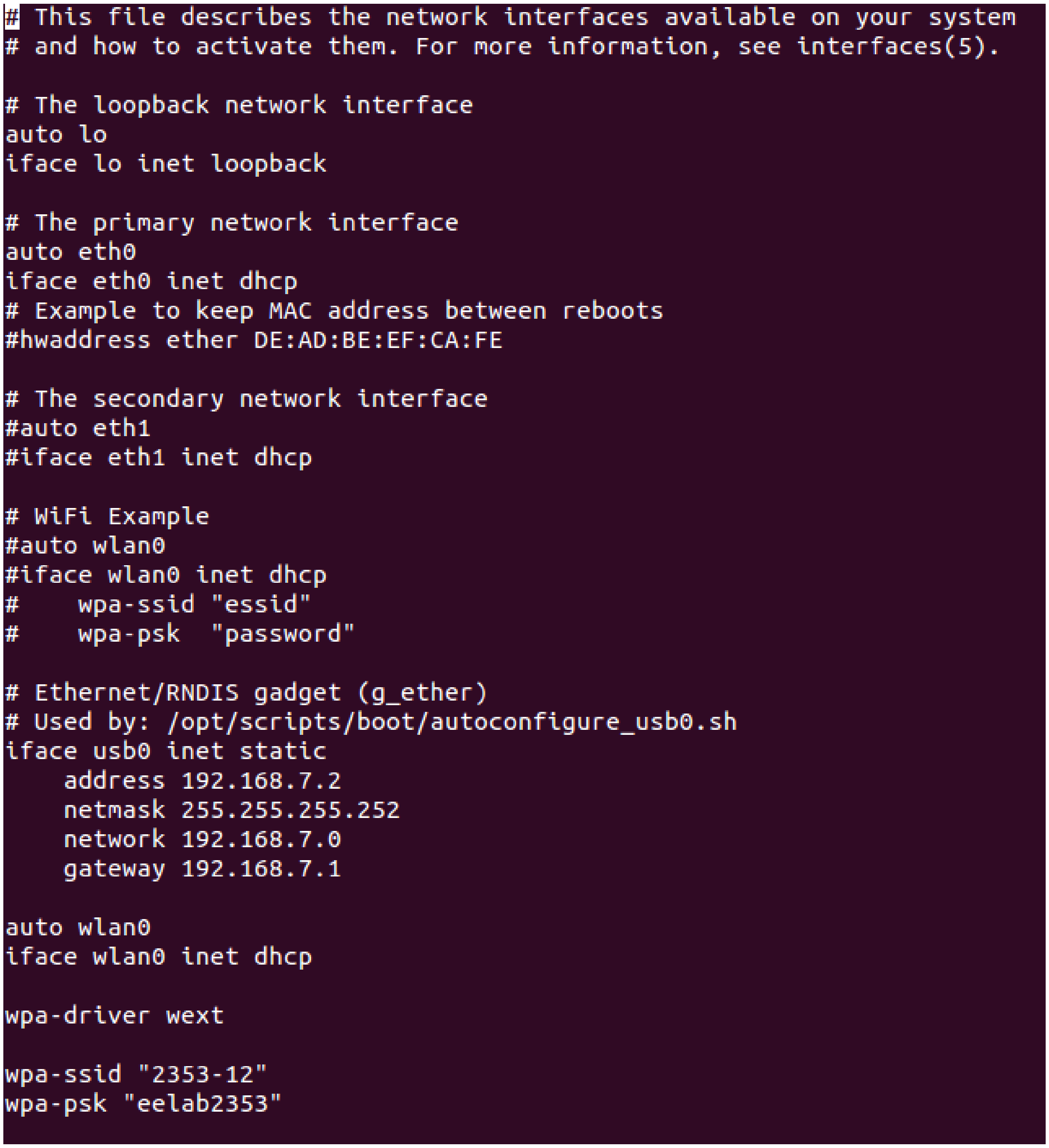


```
          Cell 10 - Address: 90:9F:33:24:26:04
                    ESSID:"2353-12"
                    Protocol:IEEE 802.11bgn
                    Mode:Master
                    Frequency:2.472 GHz (Channel 13)
                    Encryption key:on
                    Bit Rates:144 Mb/s
                    Extra:rsn_ie=30140100000fac040100000fac040100000fac020000
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : CCMP
                        Pairwise Ciphers (1) : CCMP
                        Authentication Suites (1) : PSK
                    Quality=100/100  Signal level=-45 dBm
```

Although we got around 20 cells, I only took a screen shot of cell 10 (2353-12), which is the WiFi access that we are going to connect to.

25. Setup WPA for security

In order to connect to WiFi 2353-12, we had to configure interfaces file of the Beaglebone (/etc/network/interfaces). Although the lab text had many different arguments like wpa-ap-scan, wpa-proto, wpa-pairwise, wpa-group, and wpa-key-mgmt, we only included, wpa-driver-text, wpa-ssid, and wpa-psk in our interfaces file. The interfaces file is shown below:



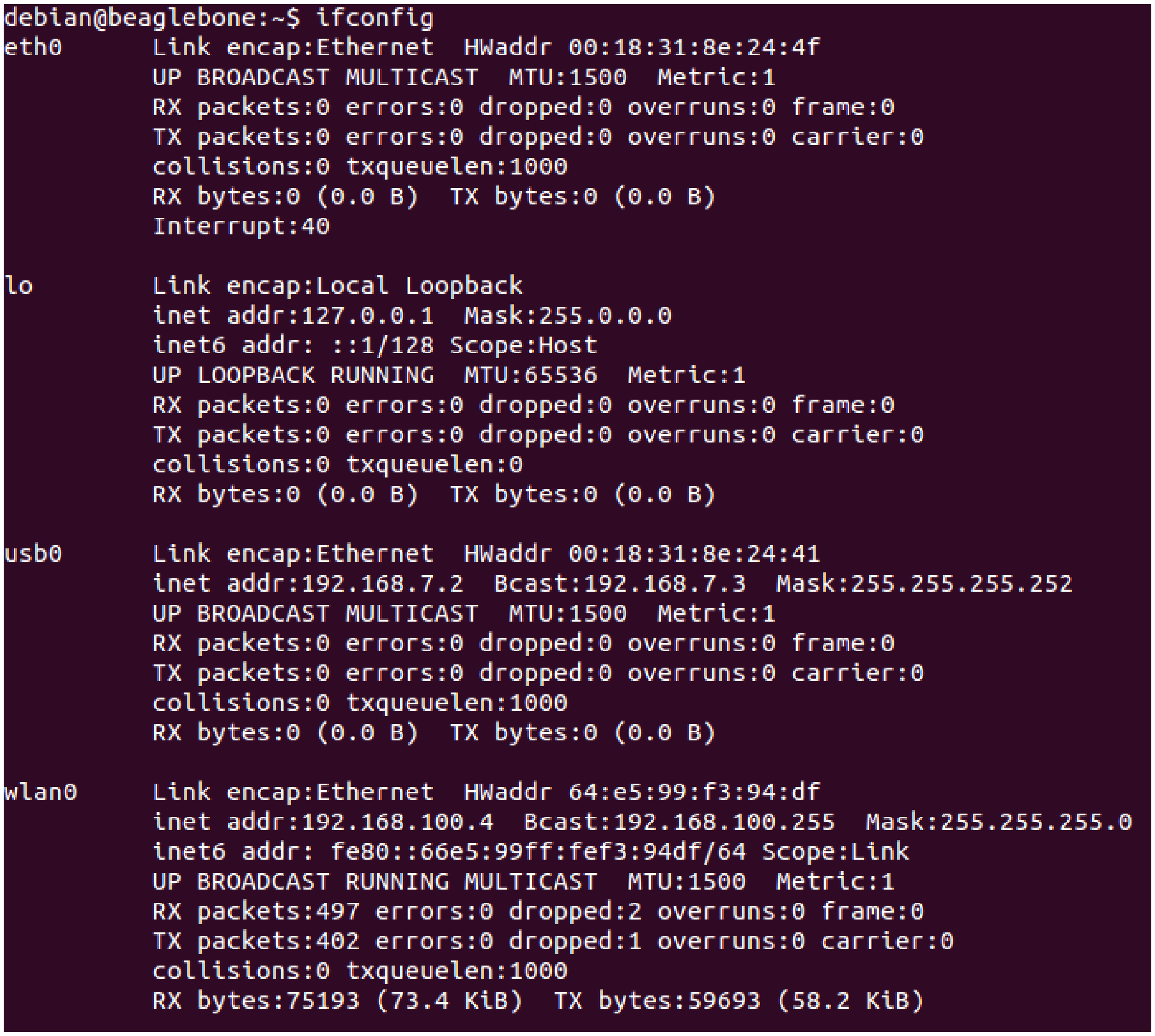
```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# 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

# The secondary network interface
#auto eth1
#iface eth1 inet dhcp

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

# Ethernet/RNDIS gadget (g_ether)
# Used by: /opt/scripts/boot/autoconfigure_usb0.sh
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.252
    network 192.168.7.0
    gateway 192.168.7.1

auto wlan0
iface wlan0 inet dhcp

wpa-driver wext

wpa-ssid "2353-12"
wpa-psk "eelab2353"
```

26. Start WiFi driver manually

After configuring the interfaces file, we started WiFi driver by giving "ifup wlan0" command.

After, the DHCP protocol searched for available IP address and assigned our Beaglebone's WiFi dongle an IP address which is 192.168.100.4.

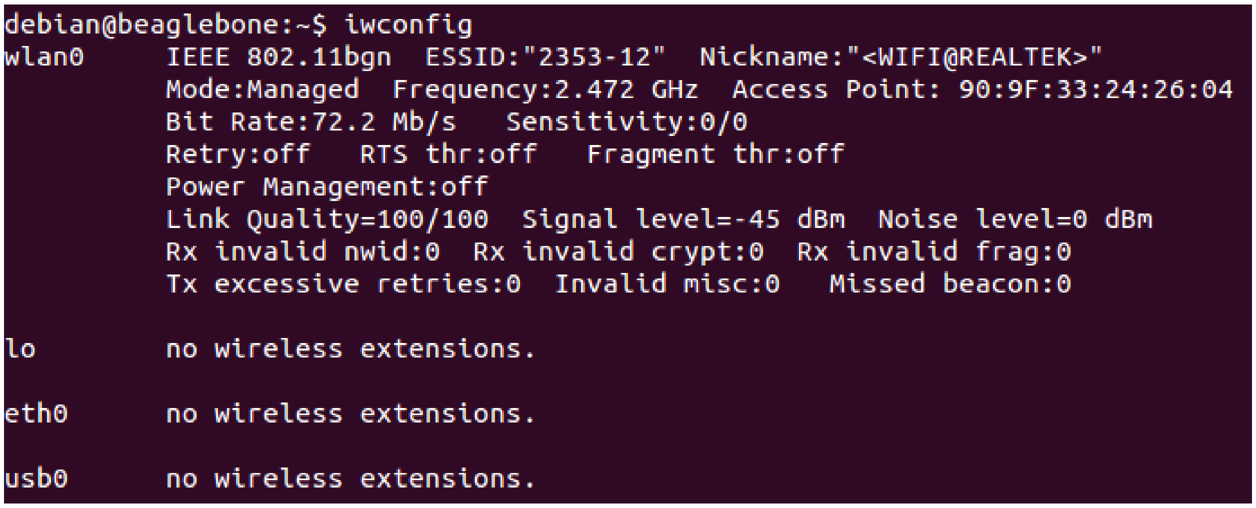
In step 22, we did not have IP address for wlan0 field. However, now we are connected to WiFi. "ifconfig" command gave:



```
debian@beaglebone:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:18:31:8e:24:4f
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb0      Link encap:Ethernet  HWaddr 00:18:31:8e:24:41
          inet addr:192.168.7.2  Bcast:192.168.7.3  Mask:255.255.255.252
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 64:e5:99:f3:94:df
          inet addr:192.168.100.4  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::66e5:99ff:fef3:94df/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:497 errors:0 dropped:2 overruns:0 frame:0
          TX packets:402 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:75193 (73.4 KiB)  TX bytes:59693 (58.2 KiB)
```

As you can see in the above picture, we now have an IP address configured: 192.168.100.4.

"iwconfig" command gave:



```
debian@beaglebone:~$ iwconfig
wlan0     IEEE 802.11bgn  ESSID:"2353-12"  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency:2.472 GHz  Access Point: 90:9F:33:24:26:04
          Bit Rate:72.2 Mb/s   Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=100/100  Signal level=-45 dBm  Noise level=0 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.

usb0      no wireless extensions.
```

As you can see in the above picture, we now have WiFi name and access point.

27, 28, 29

Even after rebooting the Beaglebone, our Beaglebone still had WiFi connection to the 2353-12.

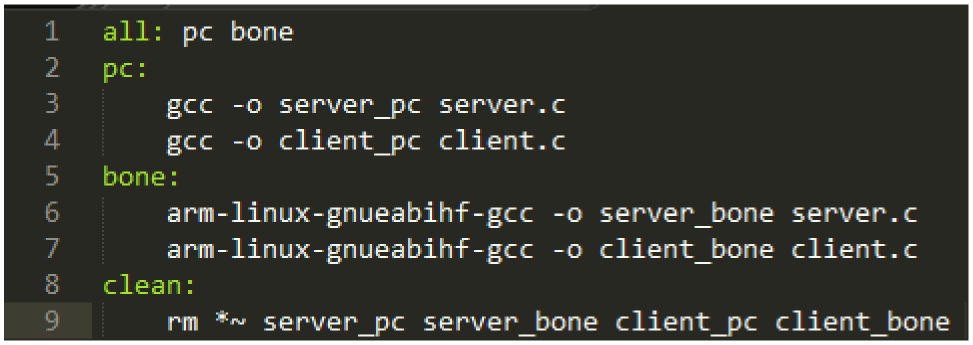
We also added the new IP address to the /etc/exports file so we can transfer files between Beaglebone and PC over WiFi.

C. Test Stream Socket

30: We created a new directory.

31: We downloaded server.c and client.c file

32: We made a makefile as follows:



```
all: pc bone
pc:
    gcc -o server_pc server.c
    gcc -o client_pc client.c
bone:
    arm-linux-gnueabihf-gcc -o server_bone server.c
    arm-linux-gnueabihf-gcc -o client_bone client.c
clean:
    rm *~ server_pc server_bone client_pc client_bone
```

For files that will be run in PC we compiled the program natively using gcc. For files that will be run in Beaglebone, we cross-compiled the files using arm-linux-gnueabihf-gcc because

Beaglebone has armhf architecture and can't run files compiled by gcc.

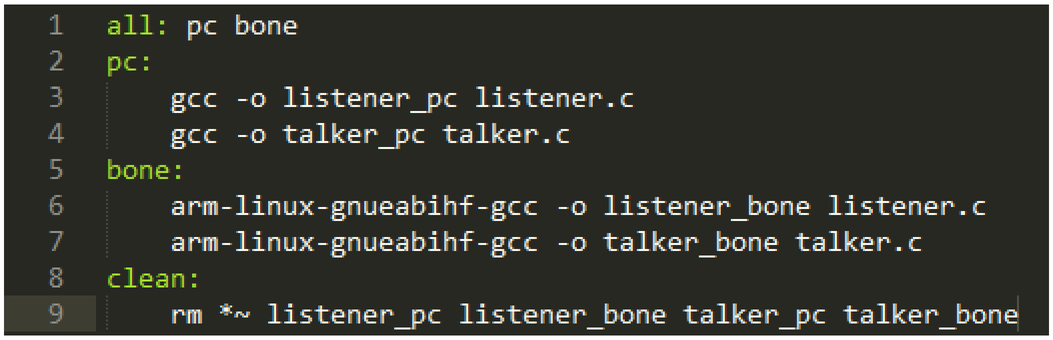
33, 34: We compiled the programs using make. When we ran PC to PC and PC to Beaglebone, we got the exact same result as the lab text.

D. Test UDP Example

40: We created a new directory

41: We downloaded talker.c and listener.c file

42: We made a makefile and compiled the files using make.



```
all: pc bone
pc:
	gcc -o listener_pc listener.c
	gcc -o talker_pc talker.c
bone:
	arm-linux-gnueabihf-gcc -o listener_bone listener.c
	arm-linux-gnueabihf-gcc -o talker_bone talker.c
clean:
	rm *~ listener_pc listener_bone talker_pc talker_bone
```

43, 44: When we ran PC to PC and PC to Beaglebone, we got the exact same result as the lab text.

E. Remote control via WiFi and UDP

Source code for Remote_Control_PC.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <termios.h>

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>
```

```
#include <arpa/inet.h>

#include <netdb.h>

#include <sys/time.h>

#define SERVERPORT "4950"


/*********** getche.c ***************/

static struct termios old, new;


/* Initialize new terminal i/o settings */

void initTermios(int echo)

{

  tcgetattr(0, &old); /* grab old terminal i/o settings */

  new = old; /* make new settings same as old settings */

  new.c_lflag &= ~ICANON; /* disable buffered i/o */

  new.c_lflag &= echo ? ECHO : ~ECHO; /* set echo mode */

  tcsetattr(0, TCSANOW, &new); /* use these new terminal i/o settings now */

}


/* Restore old terminal i/o settings */

void resetTermios(void)

{

  tcsetattr(0, TCSANOW, &old);

}


/* Read 1 character - echo defines echo mode */

char getch_(int echo)

{

  char ch;

  initTermios(echo);

  ch = getchar();

  resetTermios();

  return ch;
```

```
}

/* Read 1 character without echo */
char getch(void)
{
  return getch_(0);
}

/* Read 1 character with echo */
char getche(void)
{
  return getch_(1);
}

/*********** getche.c ***************/

double* stop(double* vv) {
  vv[0] = 0;
  vv[1] = 0;
  vv[2] = 0;
  return vv;
}

double* forwards(double* vv) {
  vv[0] = 1;
  vv[1] = 0;
  vv[2] = 0;
  return vv;
}

double* backwards(double* vv) {
  vv[0] = -1;
  vv[1] = 0;
  vv[2] = 0;
  return vv;
}

double* right(double* vv) {
  vv[0] = 0;
  vv[1] = 1;
  vv[2] = 0;
  return vv;
}

double* left(double* vv) {
  vv[0] = 0;
  vv[1] = -1;
  vv[2] = 0;
  return vv;
}

double* rotatecw(double* vv) {
  vv[0] = 0;
  vv[1] = 0;
  vv[2] = -1;
  return vv;
}

double* rotateccw(double* vv) {
  vv[0] = 0;
  vv[1] = 0;
  vv[2] = 1;
  return vv;
}
```

```
void add(double *vv, double *vv2, double *out) {
        int i = 0;
        for (i = 0; i < 3; i++) {
                out[i] = vv[i] + vv2[i];
        }
}

int timeval_subtract(struct timeval *result, struct timeval *t2, struct timeval *t1)
{
  long int diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec + 1000000 * t1->tv_sec);
  result->tv_sec = diff / 1000000;
  result->tv_usec = diff % 1000000;

  return (diff<0);
}

int main(int argc, char *argv[]) {
 char input = 0;
 double *vv;
 double *wv;
 vv = (double *)malloc(3*sizeof(double));
 int duty[3];

 // variables for socket programming
 int sockfd;
 struct addrinfo hints, *servinfo, *p;
 int rv;
 int numbytes;

 int id = 0;
 double elapsed_time;
```

```
float f_vv[3];

float gain;

int i;

int ll = 0;

int lr = 0;


double *temp = (double *) malloc(3*sizeof(double *));


// ./Remote_Control_PC hostname(ip) port#

struct timeval tvBegin, tvEnd, tvDiff;

gettimeofday(&tvBegin, NULL);


if (argc != 2) {

  fprintf(stderr,"Enter IP address\n");

            exit(1);

}


printf("    Remote_Control_PC\n");

printf("  Key input menu (without Enter):\n");

printf("q: ll;    w: vx;    e: lr;\n");

printf("a: vy;    s: stop;    d: -vy;\n");

printf("z: +w;    x: -vx;    c: -w;\n");

printf("  Speed up/down with multiple key strokes\n");

printf("'Esc' or 'Ctrl-D' key to terminate the program\n");


while (1) {

  input = getche();

  if (input == 27 || input == 4) { //Ctrl+D = EOT (Ascii: 4)


    exit(0);

  } else if (input == 'w') {

            temp = forwards(temp);
```

```
        add(vv, temp ,vv);
} else if (input == 's') {
        vv = stop(vv);
} else if (input == 'a') {
        temp = left(temp);
        add(vv, temp, vv);
} else if (input == 'd') {
        temp = right(temp);
        add(vv, temp, vv);
} else if (input == 'z') {
        temp = rotateccw(temp);
        add(vv, temp, vv);
} else if (input == 'c') {
        temp = rotatecw(temp);
        add(vv, temp, vv);
} else if (input == 'x') {
        temp = backwards(temp);
        add(vv, temp, vv);
} else if (input == 'q') {
        if(ll == 0) {
                ll = 1;
        } else {
                ll = 0;
        }
} else if (input == 'e') {
        if (lr == 0) {
                lr = 1;
        } else {
                lr = 0;
        }
}
```

```
gain = 1;


for (i = 0; i < 3; i++) {
        f_vv[i] = gain * vv[i];
}


memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;



if ((rv = getaddrinfo(argv[1], SERVERPORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
}

 // loop through all the results and make a socket
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("talker: socket");
        continue;
    }
    break;
}



if (p == NULL) {
    fprintf(stderr, "talker: failed to create socket\n");
    return 2;
}
```

```
char message[50];
sprintf(message, "%d ", id);
gettimeofday(&tvEnd, NULL);
        char *temp2 = malloc(30*sizeof(char *));
        char *temp3 = malloc(30*sizeof(char *));
        memset(temp2, 0, sizeof(temp2));
        memset(temp3, 0, sizeof(temp3));
timeval_subtract(&tvDiff, &tvEnd, &tvBegin);
sprintf(temp3, "%ld.%06ld ", tvDiff.tv_sec, tvDiff.tv_usec);
strncpy(temp2, temp3, strlen(temp3)-4);
strcat(temp2, " ");
strcat(message, temp2);



sprintf(temp2, "%.2f ", f_vv[0]);
strcat(message, temp2);
        sprintf(temp2, "%.2f ", f_vv[1]);
strcat(message, temp2);
sprintf(temp2, "%.2f ", f_vv[2]);
strcat(message, temp2);
sprintf(temp2, "%d ", ll);
strcat(message, temp2);
sprintf(temp2, "%d", lr);
strcat(message, temp2);


        printf("\nKey '%c'. cmd: %s\n", input, message);
        id++;

message[strlen(message)] = '\0';

if ((numbytes = sendto(sockfd, message, sizeof(message), 0,
      p->ai_addr, p->ai_addrlen)) == -1) {
        perror("talker: sendto");
        exit(1);
    }
    usleep(100000);


    freeaddrinfo(servinfo);

    close(sockfd);
  }
  free(vv);
  free(temp);
}
```

Explanation:

Basically, I combined the previous codes and implemented UDP transfer in this code. This code reads in user input from standard input stream. User has 11 options for input. Q: turn left led on, W: go forward, E: turn right led on, A: go left, S: stop, D: go right, Z: rotate counter clock wise, X: go backwards, C: rotate clockwise, and esc and Ctrl-D for quit. After reading from the user input, I generate vv matrix from the input. After, I generate message that will be sent to the server. The message format is “id time vx vy vw ll lr”. Id describes the order of the message, time describes the time elapsed from the start of the program which can be calculated using gettimeofday function, vx vy vz is made using the forwards, stop, left, right, rotatecww, rotatecw, backwards functions. Lastly, ll and lr is toggled when a user types q or e. After getting all the information, I can generate the message using strcat function, which concatenate strings. Since the variables are either integer or floats, I had to make a buffer and put the variables into the buffer before I concatenate. For UDP implementation, I first set up variables using getaddrinfo, which will give the information of server that I will connect to. After, I create socket that will allow me to make a connection to the server (beaglebone). Unlike TCP, UDP do not need to use function connect before sending the data. As a result, we send the data right away by using the function sendto. The data we are sending is the message that I generated above. After I send the message, I close the socket, and the procedure repeats until the user types esc or ctrl-D.

Source code for WiFi_Control_TMR.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <poll.h>
#define MYPORT "4950"    // the port users will be connecting to
#define MAXBUFLEN 150
#define SYSFS_GPIO_DIR "/sys/class/gpio"
#define POLL_TIMEOUT (3 * 1000) /* 3 seconds */
#define MAX_BUF 64

int gpio_export(unsigned int gpio)
{
  int fd, len;
  char buf[MAX_BUF];

  fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
  if (fd < 0) {
    perror("gpio/export");
    return fd;
  }
  len = snprintf(buf, sizeof(buf), "%d", gpio);
  write(fd, buf, len);
  close(fd);
  return 0;
}

int gpio_unexport(unsigned int gpio)
{
  int fd, len;
  char buf[MAX_BUF];

  fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
  if (fd < 0) {
    perror("gpio/export");
    return fd;
  }

  len = snprintf(buf, sizeof(buf), "%d", gpio);
  write(fd, buf, len);
  close(fd);
  return 0;
}

int gpio_set_dir(unsigned int gpio, unsigned int out_flag)
```

```
{
  int fd, len;
  char buf[MAX_BUF];

  len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR  "/gpio%d/direction", gpio);

  fd = open(buf, O_WRONLY);
  if (fd < 0) {
    perror("gpio/direction");
    return fd;
  }

  if (out_flag)
    write(fd, "out", 4);
  else
    write(fd, "in", 3);

  close(fd);
  return 0;
}

int gpio_set_value(unsigned int gpio, unsigned int value)
{
  int fd, len;
  char buf[MAX_BUF];
  len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

  fd = open(buf, O_WRONLY);
  if (fd < 0) {
    perror("gpio/set-value");
    return fd;
  }


  if (value)
    write(fd, "1", 2);
  else
    write(fd, "0", 2);


  close(fd);
  return 0;
}


void ind_run(int on, int PWM) {
  FILE *file;
  if (PWM == 1) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/run", "w");
  } else if (PWM == 2) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/run", "w");
  } else if (PWM == 3) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/run", "w");
  } else {
    fprintf(stderr, "Please enter a correct PWM\n");
    exit(1);
  }
  if(on) {
    fprintf(file, "%d", 1);
  } else {
    fprintf(file, "%d", 0);
  }
  fclose(file);
}
```

```
void period(int period, int PWM) {
  FILE *file;
  if (PWM == 1) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/period", "w");
  } else if (PWM == 2) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/period", "w");
  } else if (PWM == 3) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/period", "w");
  } else {
    fprintf(stderr, "Please enter a correct PWM\n");
    exit(1);
  }

  if (period > 0) {
    fprintf(file, "%d", period);
  } else {
    fprintf(stderr, "Please enter a valid period\n");
  }

  fclose(file);
}

void duty(unsigned int duty, int PWM) {
  FILE *file;
  if (PWM == 1) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/duty", "w");
  } else if (PWM == 2) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/duty", "w");
  } else if (PWM == 3) {
    file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/duty", "w");
  } else {
    fprintf(stderr, "Please enter a correct PWM\n");
    exit(1);
  }

  if (duty > 0) {
    fprintf(file, "%d", duty);
    fclose(file);
  }
}

void vvTOwv(double* vv, double* wv) {
  double r = 0.02;
  double L = 0.10;
  double T[3][3] = {{0, 1/r, L/r}, {-1.1547/r, -0.5/r, L/r}, {1.1547/r, -0.5/r, L/r}};
  double Tv = 0;
  int i, j;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      Tv = Tv + vv[j]*T[i][j];
    }
    wv[i] = Tv;
    Tv = 0;
  }
}

void quit() {
  ind_run(0, 1);
  ind_run(0, 2);
  ind_run(0, 3);
}
```

```
void run(double* wv) {
 int p=3000000;
 unsigned int duty1;
 unsigned int duty2;
 unsigned int duty3;
 unsigned int stopDuty = 1500000;

 if (wv[0] > 0) {
  duty1 = 1533400+(2999939-1533400)*wv[0]*0.01;
 } else if (wv[0] < 0) {
  duty1 = 1400000+(1400000-660)*wv[0]*0.01;
 } else {
  duty1 = stopDuty;
 }

 if (wv[1] > 0) {
  duty2 = 1530440+(2999999-1530440)*wv[1]*0.01;
 } else if (wv[1] < 0) {
  duty2 = 1395000+(1395000-720)*wv[1]*0.01;
 } else {
  duty2 = stopDuty;
 }

 if (wv[2] > 0) {
  duty3 = 1538700+(2999939-1538700)*wv[2]*0.01;
 } else if (wv[2] < 0) {
  duty3 = 1410000+(1410000-720)*wv[2]*0.01;
 } else {
  duty3 = stopDuty;
 }
```

```
  ind_run(0, 1);
  ind_run(0, 2);
  ind_run(0, 3);

  period(p, 1);
  period(p, 2);
  period(p, 3);

  duty(duty1, 1);
  duty(duty2, 2);
  duty(duty3, 3);

  ind_run(1, 1);
  ind_run(1, 2);
  ind_run(1, 3);
}

int main(void)
{
    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;
    int numbytes;
    struct sockaddr_storage their_addr;
    char buf[MAXBUFLEN];
    socklen_t addr_len;
    char s[INET6_ADDRSTRLEN];

    int id;
    float time_elapsed;
    float vx;
    float vy;

    float vw;

    int ll;

    int lr;

    double vv[3];

    double wv[3];


    gpio_export(20);

    gpio_set_dir(20, 1);


    gpio_export(7);

    gpio_set_dir(7, 1);


    while(1) {

      memset(&hints, 0, sizeof hints);

      hints.ai_family = AF_UNSPEC; // set to AF_INET to force IPv4

      hints.ai_socktype = SOCK_DGRAM;

      hints.ai_flags = AI_PASSIVE; // use my IP


      if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {

        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));

        return 1;

      }


      // loop through all the results and bind to the first we can

      for(p = servinfo; p != NULL; p = p->ai_next) {

        if ((sockfd = socket(p->ai_family, p->ai_socktype,

            p->ai_protocol)) == -1) {

          perror("listener: socket");

          continue;

        }
        if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("listener: bind");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "listener: failed to bind socket\n");
        return 2;
    }

    freeaddrinfo(servinfo);

    printf("listener: waiting to recvfrom...\n");

    addr_len = sizeof their_addr;
    if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1 , 0,
        (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }
    buf[numbytes] = '\0';

    printf("listener: packet is %d bytes long\n", numbytes);
    printf("listener: packet contains \"%s\"\n", buf);

    id = atoi(strtok(buf, " "));
```

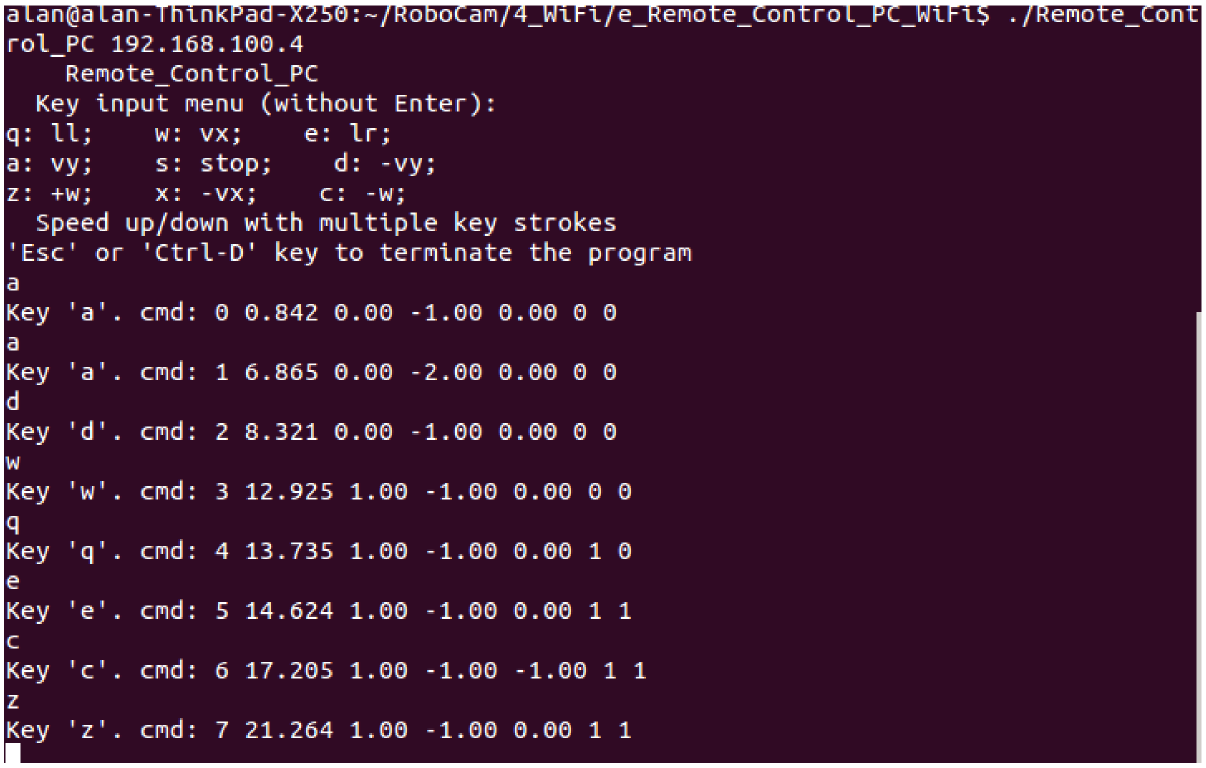
```
        time_elapsed = atof(strtok(NULL, " "));
        vx = atof(strtok(NULL, " "));
        vy = atof(strtok(NULL, " "));
        vw = atof(strtok(NULL, " "));
        ll = atoi(strtok(NULL, " "));
        lr = atoi(strtok(NULL, " "));

        vv[0] = vx;
        vv[1] = vy;
        vv[2] = vw;



        if(vv[0] == 0 && vv[1] == 0 && vv[2] == 0) {
          quit();
          gpio_set_value(20, ll);
          gpio_set_value(7, lr);
          close(sockfd);
          continue;
        }
        vvTOwv(vv, wv);
        run(wv);
        gpio_set_value(20, ll);
        gpio_set_value(7, lr);
        close(sockfd);
    }
    unexport(20);
    unexport(7);
    return 0;
}
```

Explanation:

In this code, we receive the message that is sent by Remote_Control_PC program. First, we make a socket and bind. Binding is required in receiver because it assigns port number to the socket. Therefore, the sender knows which port number to send the data to. After binding, I used recvfrom function to receive data from the sender. After reciving the data, I parsed the message so that I can get the variables that I need. I used strtok function with token " " (space) because the message is divided in spaces. As a result, I could extract vx, vy, vw, ll, and lr variables. After putting vx, vy, vw variable in to a buffer, I converted the vv in to angular velocity wv using vvTOwv function that I made in Lab 3. After that I echoed duty using run function that I made in Lab 3. Before you run this program, you need to run Acquire_Triple_PWMs.sh program before. In run function, I set duty value based on values in wv buffer. After, it calls functions period, duty, and ind_run functions, which opens run, period, and duty file for each three servomotors and write appropriate values that were calculated before. As a result, period and duty value is written, and the servomotor is ran. After, I used the functions from Lab 2 to control the light. I exported the GPIO, set direction to out, and set value to the variable that I got from parsing. This program runs unless you quit it forcefully. Exporting, setting direction, and setting value are same as manipulating period, duty, and run value. To take an advantage of Linux file system, we just open the file using fopen function and write to the file using fprintf funciton. As a result, we are able to manipulate servomotors and leds with given messages from Remote_Control_PC program.

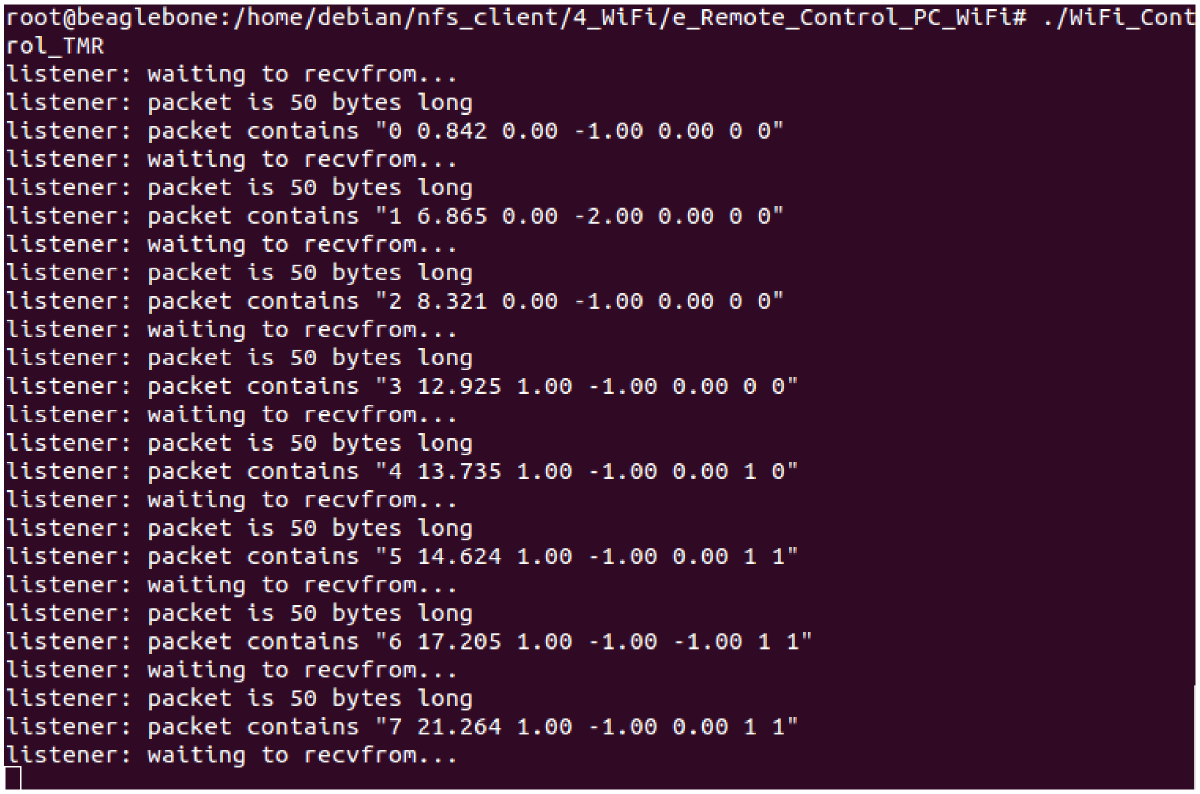
After compiling the two files, we ran Remote_Control_PC file on PC and WiFi_Control_TMR on Beaglebone. We could see that UDP packets is successfully transferred from PC to Beaglebone. After running Acquire_Triple_TMR.sh file to get access to the server motors, we could control Beaglebone using two files: Remote_Control_PC and WiFi_Control_TMR.

Remote_Control_PC: PC



```
alan@alan-ThinkPad-X250:~/RoboCam/4_WiFi/e_Remote_Control_PC_WiFi$ ./Remote_Cont
rol_PC 192.168.100.4
    Remote_Control_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 to terminate the program
a
Key 'a'. cmd: 0 0.842 0.00 -1.00 0.00 0 0
a
Key 'a'. cmd: 1 6.865 0.00 -2.00 0.00 0 0
d
Key 'd'. cmd: 2 8.321 0.00 -1.00 0.00 0 0
w
Key 'w'. cmd: 3 12.925 1.00 -1.00 0.00 0 0
q
Key 'q'. cmd: 4 13.735 1.00 -1.00 0.00 1 0
e
Key 'e'. cmd: 5 14.624 1.00 -1.00 0.00 1 1
c
Key 'c'. cmd: 6 17.205 1.00 -1.00 -1.00 1 1
z
Key 'z'. cmd: 7 21.264 1.00 -1.00 0.00 1 1
```

WiFi_Control_TMR:



```
root@beaglebone:/home/debian/nfs_client/4_WiFi/e_Remote_Control_PC_WiFi# ./WiFi_Cont
rol_TMR
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "0 0.842 0.00 -1.00 0.00 0 0"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "1 6.865 0.00 -2.00 0.00 0 0"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "2 8.321 0.00 -1.00 0.00 0 0"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "3 12.925 1.00 -1.00 0.00 0 0"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "4 13.735 1.00 -1.00 0.00 1 0"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "5 14.624 1.00 -1.00 0.00 1 1"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "6 17.205 1.00 -1.00 -1.00 1 1"
listener: waiting to recvfrom...
listener: packet is 50 bytes long
listener: packet contains "7 21.264 1.00 -1.00 0.00 1 1"
listener: waiting to recvfrom...
```

As you can see pictures above, the messages are transferred from PC to Beaglebone well. We also checked that the Beaglebone robot moved well accordingly as we typed key input. To sum up, we connected the Beaglebone to WiFi and controlled Beaglebone's movement wirelessly.

Discussion

There was problem of connecting the Beaglebone to the Internet using WiFi, taking 5 hours to connect. The lab text had many other arguments in /etc/njetwork/interfaces such as wpa-ap-scan, wpa-proto, wpa-pairwise, wpa-group, and wpa-key-mgmt. We got the WiFi working after removing all those arguments and putting only wpa-driver, wpa-ssid, and wpa-psk.

There is also problem of making the Beaglebone robot increase its speed when a same key is pressed multiple times. There are three motors in our Robocam robot and each have different deadband and relationship between duty and angular speed. As a result, it is almost impossible to implement increasing speed when same key is pressed multiple times by just scaling duty value based on angular velocity. As a result, we decided to hard code the duty value to implement the speeding up mechanism. We decided that there are three different speed level. If we press w once, the Robocam moves in forward direction with speed level 1. If we press w one more time, the Robocam moves in forward direction with slightly faster speed of level 2, and so on up to level 3. We spent a lot of time finding the duty values for each wheels that satisfy these levels. While doing that we approached another problem. We gave servo 1 duty value of 1460000. When we gave servo 2 duty value of 1543000, the speed of servo 1 changed although we did not change the duty value of servo 1. We realized that the speed of servo changes when multiple servos are running at the same time. This made us harder to find the duty values that we want. Although we found the duty values we wanted, we did not implement in our code because the TA said we do not need to implement the speeding up mechanism. The duty values that we found is as follows:

speed	duty 1	duty 2	duty 3

clock wise rotation: speed 1	1537000	1532000	1539000
speed 2	1540000	1600000	11550000
speed 3	1538000	1535000	1548000
counter clock wise: speed 1	1416000	1410000	1423000
speed 2	1397500	1397500	1397500
speed 3	1000000	1000000	1000000
forward (w): speed 1	0	1460000	1543000
speed 2	0	1450000	1545000
speed 3	0	1000000	2000000
backwards (x): speed 1	0	1535000	1460000
speed 2	0	1540000	1400000
speed 3	0	2000000	1000000

Discussion Questions

1) Compare TCP and UDP

Transmission control protocol (TCP) and user datagram protocol (UDP) are transport layer protocols in computer networking. Protocols define set of rules such as format of messages sent and received. Using TCP or UDP as transport layer protocols, applications in application layer can send and receive messages among network entities. Transport layer protocols provide logical communication between application processes running on different hosts. TCP is widely used today and relatively reliable and UDP is a no-frills service.

TCP is a connection-oriented protocol. Before sending and receiving messages using TCP, sender and receiver go through "3-way hand shaking phase" to avoid half open connection and incarnation overlap. This is done by sending SYN bit. As a result, TCP requires 4 tuple of information when it makes a connection: source IP address, source port number, destination IP address, and destination port number. IP address is a unique number that every device that is connected to the internet has. In order to specify which device I want to send a message to, you need to know the destination IP address. In a receiving host, there may be many processing running. Once a receiving host receives a message, it needs to know which process to forward to. The port number solves this problem. TCP is full duplex, which means that it supports bi-directional data flow in a single connection. Therefore, source IP address and port number is needed. On the other hand, UDP is a connectionless protocol because it does not go through hand shaking process before it sends messages. A sender who is using UDP does not contact the receiver before it sends a message. As a result, UDP only requires two tuple of information: destination IP address and destination port number.

There are many functions that transfer layer protocols can provide such as reliable, in-order delivery, congestion control, flow control, timing, bandwidth guarantee, security, and many others. TCP provides reliable, which means that it guarantees that message that you send with TCP will get to the receiver for sure. Also, TCP provides in-order delivery, which means that the messages you send will be received at the receiver in order. TCP also offers flow control and congestion control. Flow control mechanism the sending rate of sender so that the sender does not overflow receiver's buffer by transmitting messages too much and fast. Receiver's buffer may overflow when sender's sending rate is faster than receiver's processing rate. Flow control can avoid such overflow. Congestion control mechanism controls sender's sending rate so that it does not overwhelm the network. If many users are using network and sender does not implement congestion control, packets (messages in small chunks) will get lost, you will experience long delays, there will be large queuing delays, sender must perform retransmissions, network resources are wasted, and many other bad things happen. In order to prevent these bad things from happening, TCP provides congestion control. It limits the sender's sending rate when there are large traffics in the network. Since TCP provides reliable and in-order delivery, it is used by HTTP (hypertext transfer protocol), FTP (file transfer protocol), SMTP (simple mail transfer protocol), and other application that requires reliable and in-order delivery.

Like I mentioned above, UDP is a no frills protocol that provides none of functions listed above. UDP does not guarantee in-order delivery and your packets may not even get to the receivers. Why is UDP used then? We use UDP because it is fast. Since there is no connection establishment (handshake) before sending and receiving is done, UDP can send messages right away. Also, there is no connection state when in UDP. In TCP’s connection-oriented state, sender and receiver maintains buffer, congestion-control parameters, sequence and acknowledgement number, and many other parameters. Since UDP is connectionless protocol, it does not require a lot of parameters. Also, compared TCP’s 20 bytes header, UDP segment (message) has only 8 bytes. Therefore, UDP pays less header tax. Also, it does not implement flow control and congestion control. Therefore, senders can send messages any time they want with UDP. The main reason for UDP usage is that it is simple, fast, and you have more control. It is used in application that does not require reliable data transfer nor in-order delivery like VoIP which can tolerate some packet loss and non-in-order delivery. Because UDP does not implement congestion control and flow control, there is controversial over using UDP. High loss rates caused by UDP senders cause the TCP senders to decrease their sending rates, crowding out of TCP sessions.

2) Compare Ethernet broadcast, multicast, and unicast.

Broadcast, multicast, and unicast are the different methods of transferring data from one host to another.

In unicast messages are sent from one point to another point. That is, there is exactly one sender and exactly one receiver. Messages are transmitted from one sender to one receiver. As a result, if you want to send a same message to multiple receivers, you need to send multiple unicast messages to each receivers (you need to specify the IP addresses for each receivers). The messages sent in unicast are generally private information or information that is dedicated to one receiver. For example, in this lab, we send movement information (move forward, backwards, left, right, etc) from PC to Beaglebone. In that case, we are using unicast because there is one sender, PC, and one receiver, Beaglebone. TCP's point to point characteristic (one sender and one receiver) also describes unicast property. It is also used in web surfing and file transfers. We generally use unicast for internet.

In broadcast, messages are sent from one point to all the points. That is, there is one sender and multiple receivers. Messages are transmitted from one sender to all receivers that are connected at once. It is efficient because sender only sends one information and every receiver can see it. In broadcast, receivers have no choice of receiving what information that it wants to receive. Receivers have to at least look at the broadcast messages. As a result, we are only able to broadcast our broadcast domain, generally a local subnet. We used broadcast in this lab when we used the function "ifup wlan0". The host broadcasted "DHCP discover" messages and found an available IP address.

Multicast is somewhat a combination of broadcast and unicast. Messages are transmitted from the sender to multiple receivers. Unlike broadcast that sends messages to all the receivers, multicast sends messages only to the receivers that are interested in receiving the message. To receive multicast message, a host has to belong to multicast groups, which a host can join or leave at any time. It is used in multimedia delivery and stock exchanges. It has scaling problem in large networks. We generally do not use multicast in internet.

3) Summarize purpose and usage of wpa-passphrase

WPA, Wi-Fi Protected Access (WPA), is used in securing wireless computer networks. Basically, it is used to only allow credential users to connect to your network. WPA-passphrase is the password that you can use in WPA2PSK, Wi-Fi Protected Access 2 - Pre-Shared Key. It generates a 256-bit pre-shared WPA key from plain English passphrase that you provide. It is a way of securing your network using WPA2, the next version of WPA, using a password. By providing passphrase that is between 8 and 63 characters long, you can encrypt your network. TKIP (for temporal key integrity protocol) is used to generate unique encryption keys, which are constantly changed, for each wireless client. To sum up, with WPA2PSK, which is designed for home users, you are configuring each WLAN node with a plain English passphrase which are converted to unique encryption keys by TKIP instead of providing your router with an encryption key. Because the encryption keys from passphrase created by TKIP are constantly changed, it is more secure than just providing encryption keys yourself. When client connects, client needs to provide the password to be connected to the WLAN. Without network security or with weak security, hackers can hijack a TCP connection and inject malicious packets to your network.

In this lab, we configured our Beaglebone's interfaces file (/etc/netowrk/interfaces) to connect to Wi-Fi. We had to provide wpa-ssid, which is the name of the Wi-Fi, and wpa-psk, which is the passphrase of the Wi-Fi. If we do not know the password, we cannot connect to the Wi-Fi. By having wpa-passphrase, we can secure our network from outsiders and hackers.

Reference

DesignLab_RoboCam_Lecture4_E.pdf

DesignLab_RoboCam_Lab4_E.pdf

Computer Networking A Top-Down Approach

Beej's Guide to Network Programming