Final Project Report

Xinxin Zhang, Jiaqi Zhang, Heng Zhou

I. Hardware Setup

In this lab, we connect LCD display, shift register (SN74hc595), driver IC for DC motor (TB6612FNG), and Bluetooth (RN-42) with Beagle Bone Black (BBB). (see the Fig1. and Fig 2. below)

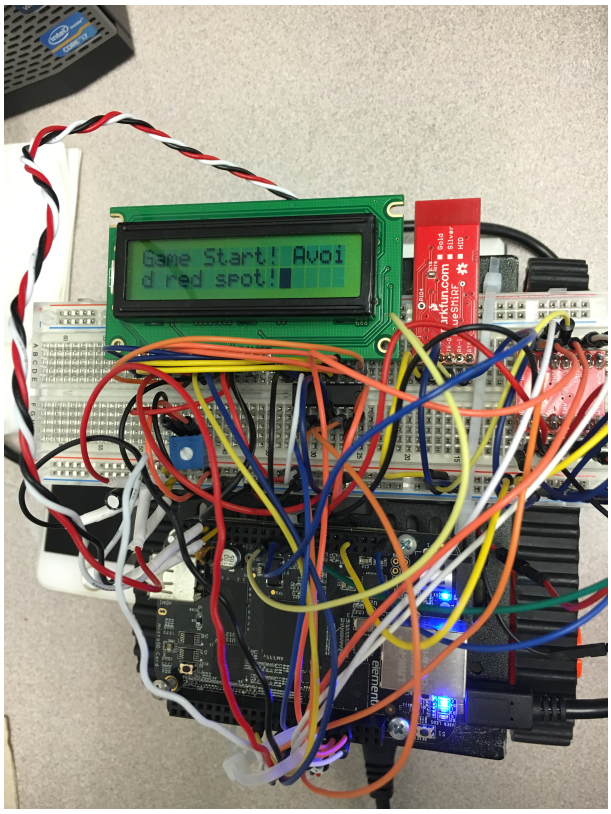

Fig 1. Circuit connection on BBB

Sensor							
	White	AIN0	P9-39				
	Black	GND					
	Red	Sys 5v	P9-7				
Motor Driver							
	PWMA	PWM1A	P9-14		VM	Motor Power	
	AIN2	GPIO67	P8-8		VCC	SYS_5V	
	AIN1	GPIO66	P8-7		GND	GND	
	STBY	SYS_5V			A01	Motor Right Red	
	BIN1	GPIO68	P8-10		A02	Motor Right Black	
	BIN2	GPIO69	P8-9		B02	Motor Left Black	
	PWMB	PWM1B	P8-19		B01	Motor Left Red	
	GND	GND			GND	GND	
Register Pin	Register Nam	LCD Name	LCD Pin	GPIO Functic	BBB pin	74HC595 Pin Number	
	VCC			Sys 5v	P9_7	16	
	QA	DB0	7			15	
	SER			GPIO_26	P8_14	14	
	OE	GND		DGND	P9_1	13	
	RCLK		RCLK STCP	GPIO_65	P8_18	12	shuchu
	SRCLK		SRCLK SHC	GPIO_27	P8_17	11	shuru
	MR			Sys 5v	P9_7	10	
	QH					9	
	GND			DGND		8	
15	QA	DB0	7			15	
1	QB	DB1	8			1	
2	QC	DB2	9			2	
3	QD	DB3	10			3	
4	QE	DB4	11			4	
5	QF	DB5	12			5	
6	QG	DB6	13			6	
7	QH	DB7	14			7	
8							
			15				
			16				
		VSS	1	DGND	P9_1		
		Vcc	2	Sys 5v	P9_7		
		Vee	3				
		RS	4	GPIO_46	P8_16		
		RW	5	DGND	P9_1		
		E	6	GPIO_61	P8_26		
blueooth							
VCC	vcc						
GND	gnd						
TX01		UART1_RXI	P9-26				
RX1		UART1_TXI	P9-24				

Fig 2. Wiring

II. Game

Game Description

In order to implement what we have explored about the Beagle Bone Black and its characteristics, we wrote a simple maze game in which the tank controlled by Bluetooth connection walks on the floor that mimics an imaginary maze.

Functions:

- **`void playGame();`**
  Start the game with an 8*8 matrix with 0 as clear rode, 1 as bombs and 2 as the tank. The tank starts at position (7, 0), facing north as default. Positions of the 15 bombs are randomly generated every time when the game is played. When tank gets to (7, 7), player wins; if the tank reaches any bomb, player loses.

  This function generates the original maze and put tank's position at (7, 0).

  `int carRow, carCol`: the coordinates that keep track of the position of the tank.
  `int direction`: the direction the tank is facing.
  0: north 1:east 2: south 3:west.
  `int randRow, randCol`: the coordinates the represent the position of the bombs.

```
void playGame() {
 if (count == 0){
    //map: Matrix (8 * 8) Start: (7, 0)
    carRow = 7;
    carCol = 0;
    direction = 0;
    int row;
    for (row = 0; row < 8; row++) {
       int col;
       for (col = 0; col <8; col++) {
          map[row][col] = 0;
       }
     }

    map[carRow][carCol] = 2;
    unsigned char start[] = "Game Start! Avoid the bombs!";
    info(start);
    sleep(1);

   //Set the position to be 1 (aka bomb)
   int i;
   for (i = 0; i < 15; i++) {
   //start from bottom left corner (7, 0)
```

```
        int randRow = 7;
        int randCol = 0;
        while (((randRow > 5) && (randCol < 2 )) || ((randRow >
5 && randCol > 5))) {
           randRow = rand() % 8;
           randCol = rand() % 8;
        }
           map[randRow][randCol] = 1;
        }
        showMap();
     }
     count++;
}
```

- **`void showMap();`**
  This function prints the map on the terminal. This function is called every time the position of the tank changes.

```
void showMap(){
 printf("\n");
 int h;
    int j;
    for ( h = 0; h < 8; h++ ) {
       printf("\n");
     for ( j = 0; j < 8; j++ ) {
        printf("%d ",map[h][j] );
     }
   }
}
```

- **`void move(int);`**
  This function reads from player's input and determines the motion of the tank.
  0: halt
  1: forward
  2: backward
  3: left
  4: right

```
void move(int signal) {
```

```
    if(signal == 1 ){
       moveForwards(1);
    }else if(signal == 2){
       moveBackwards(1);
    }else if(signal == 3){
       turnLeft(1);
    }else if(signal == 4){
       turnRight(1);
    }else if(signal == 0){
      stop();
    }
 }
```

- **`void moveForwards()`**
  Moves the tank forward and the tank keeps going for 0.5s. Also changes the tank's position on the map. Then determines the status of the game: win/lose/out of board.

  The new position of the tank depends on its current direction. If it's facing north, then moving forward will cause the tank's row number -1 and column number unchanged. If the tank is facing east, moving forward will cause the column position number +1 and the row number unchanged.

  The tank is supposed to be within the boundary of the 8*8 map, i.e. 0 <= `carRow`, `carCol` <= 7. The function checks if the tank is now out of boundary. If it is, the tank does not move and the position is reset to the previous one.

  If the new position on the map reads a 1, that means the tank just touched a bomb and the player loses. Function `gameOver()` is called for specific rules about losing the game.

  If the new position is (7, 7) (which is always set as 0), the player wins and function `win()` is called.

  The value at the previous position on the map is set to be 0 and the one on the new position is set to be 2.

  If the tank's new position is legal (not out of board, no bomb, no win), `controlLeft()` and `controlRight()` are called to control the wheels of the

tank. Distance is read from the front sensor 100 times per second, if the distance is greater than 3000, it technically means that there is some barrier in front of the tank, the LCD board will say “blocked” and tank will not move. Otherwise tank moves and LCD tells the current position of the tank.

```
//Move the tank forward in the current direction for 0.5s.
void moveForwards() {
    if(direction == 0 ){
       carRow--;
    } else if (direction == 1){
       carCol++;
    } else if (direction == 2){
       carRow++;
    } else {
       carCol--;
 }

 unsigned char table[100];
 if((carRow > 7) || (carRow < 0) || (carCol > 7) || (carCol
< 0)){  //stay
    sprintf(table, "%s",   "Out of board");
    if(direction == 0 ){
       carRow++;
    } else if (direction == 1){
       carCol--;
    } else if (direction == 2){
       carRow--;
    } else {
       carCol++;
    }
 } else if(map[carRow][carCol] == 1) {
    sprintf(table, "%s",   "!!!Boom!!!");
    info(table); //didnt work
    sleep(1);
    gameOver();
 } else if((carRow == 7) && (carCol == 7)){
    win();
 } else {  //move
    if(direction == 0 ){
```

```
            map[carRow+1][carCol] = 0;
        } else if (direction == 1){
            map[carRow][carCol-1] = 0;
        } else if (direction == 2){
            map[carRow-1][carCol] = 0;
        } else {
            map[carRow][carCol+1] = 0;
        }

        int n = 0;
        while (n < 100) {
            if (sensorDistance(path3) < 3000){
                sprintf(table, "%s","Position: ");
                char row[15];
                sprintf(row, "%d", carRow);
                strcat(table, row);
                strcat(table, ", ");
                char col[15];
                sprintf(col, "%d", carCol);
                strcat(table, col);
                controlLeft(0, 1);
                controlRight(1, 0);
                usleep(10000);
            } else {
            sprintf(table, "%s",   "Blocked!!");
        }
            n++;
        }
        map[carRow][carCol] = 2;
    }
    info(table);
    showMap();
    usleep(500000);
    stop();
}
```

- **`void moveBackwards();`**
  Similar to `moveForwards();` Moves the tank backwards and uses back sensor to detect distance from barrier.

```
// Move backwards in current direction for 0.5 second.
void moveBackwards() {
    if(direction == 0 ){
       carRow++;
    } else if (direction == 1){
       carCol--;
    } else if (direction == 2){
       carRow--;
    } else {
       carCol++;
}

 //printf("Backward: %i\n", direction);
 unsigned char table[100];
 if((carRow > 7) || (carRow < 0) || (carCol > 7) || (carCol
< 0)){
    sprintf(table, "%s",   "Out of board");
    if(direction == 0 ){
       carRow--;
    } else if (direction == 1){
       carCol++;
    } else if (direction == 2){
       carRow++;
    } else {
       carCol--;
    }
 } else if(map[carRow][carCol] == 1) {
    sprintf(table, "%s",   "!!!Boom!!!");
    info(table);
    //sleep(2);
    gameOver();
 } else if((carRow == 7) && (carCol == 7)){
    win();
 } else {
    if(direction == 0 ){
       map[carRow-1][carCol] = 0;
    } else if (direction == 1){
       map[carRow][carCol+1] = 0;
```

```
        } else if (direction == 2){
            map[carRow+1][carCol] = 0;
        } else {
            map[carRow][carCol-1] = 0;
        }

        int n = 0;
        while(n < 100) {
            if (sensorDistance(path4) < 3000){
                sprintf(table, "%s","Position: ");
                char row[15];
                sprintf(row, "%d", carRow);
                strcat(table, row);
                strcat(table, ", ");
                char col[15];
                sprintf(col, "%d", carCol);
                strcat(table, col);
                controlLeft(1, 0);
                controlRight(0, 1);
                usleep(10000);
            } else {
                sprintf(table, "%s",   "Blocked!!");
            }
                n++;
        }
        map[carRow][carCol] = 2;
    }
    info(table);
    showMap();
    usleep(500000);
    stop();
}
```

- **`void turnLeft();`**
  Drives the left motor forward and right motor backward for 1 second, which ends up turning the tank left for 90 degrees. Also updates the direction of the tank. If the tank was facing north (0), the new direction would be east (3).

```
// Turn left (90 degrees)
```

```
void turnLeft() {
    if(direction == 0) {
       direction == 3;
    } else{
       direction--;
    }
    controlLeft(0, 1);
    controlRight(0, 1);
    sleep(1);
}
```

- **`void turnLeft();`**
  Drives the right motor forward and left motor backward for 1 second, which ends up turning the tank right for 90 degrees. Also updates the direction of the tank. If the tank was facing north (0), the new direction would be west (1).

```
// Turn right (90 degrees)
void turnRight() {
    if (direction == 3) {
       direction = 0;
    } else {
       direction++;
    }
    controlLeft(1, 0);
    controlRight(1, 0);
    sleep(1);
 }
```

- **`void stop();`**
  Stops both motors and the tank stops.

```
// stop
void stop() {
    controlLeft(0, 0);
    controlRight(0, 0);
    usleep(1000);
}
```

- **`void controlLeft(int, int);`**
  According to the H-SW Control table (Fig 4. H-SW Function in appendix), IN1 high and IN2 low will force the wheel roll clockwise. Here, the value that transfers to the pin AIN1 and AIN2 control the motor that drives the left wheel.

```
// LEFT wheel
void controlLeft(int IN1, int IN2) {
   fprintf(value[0], "%d", IN1); //AIN1 -0
   fflush(value[0]);
   fprintf(value[1], "%d", IN2); //AIN2 -1
   fflush(value[1]);
}
```

- **`void controlRight(int, int);`**
  According to the H-SW Control table (Fig 4. H-SW Function in appendix), IN1 high and IN2 low will force the wheel roll clockwise. Here, the value that transfers to the pin BIN1 and BIN2 control the motor that drives the left wheel.

```
// RIGHT wheel
void controlRight(int IN1, int IN2) {
    fprintf(value[2], "%d", IN1); //BIN1 -0
    fflush(value[2]);
    fprintf(value[3], "%d", IN2); //BIN2 -1
    fflush(value[3]);
}
```

- **`int sensorDistance (char *);`**
  Reads the distance from the sensor.

```
int sensorDistance(char * path){
    sensor= fopen(path, "r");
    char buff[255];
    fgets(buff, 255, (FILE*) sensor);
    int tmp;
    char *ptr;
    tmp = strtol(buff, &ptr, 0);
    return tmp;
}
```

- **`void gameOver();`**
  Function is called when the tank touches a bomb. LCD board shows “Bye” and a new game is initiated.

```
void gameOver(){
    unsigned char end[] = "Bye";
    info (end);
    sleep(1);
    unsigned char re[] = "Restarting ...";
    info (re);
    sleep(2);
    playGame();
}
```

- **`void win();`**
  Function is called when the tank reaches (7,7). The LCD board shows “You Win” and restarts the game.

```
void win(){
    unsigned char end[] = "You Win!";
    info (end);
    sleep(1);
    unsigned char re[] = "Restarting ...";
    info (re);
    sleep(2);
    playGame();
}
```

III. Auto Boot Beagle Bone Black

• Bluetooth.py

This code is used to let the user transfer the command to the Beagle Bone Black car and control it remotely.

Create directory `/dev/ttyO1` as UART port for serial writing and reading and set the baud rate to 115200:

```
UART.setup("UART1")
```

```
ser = serial.Serial(port = "/dev/ttyO1", baudrate=115200)
```

The entire code:

```
import Adafruit_BBIO.UART as UART
import serial

UART.setup("UART1")

ser = serial.Serial(port = "/dev/ttyO1", baudrate=115200)
ser.close()
ser.open()
// test the port open or not
if ser.isOpen():
     print "Serial is open!"
     ser.write("Hello World!")
ser.close()
```

- **install.py**

  This code run the C code when call this python file without typing commands in terminal window.

```
import os
import time
def getch():
     os.system("bash -c \"./test\"")
getch()
```

- **p1.service**

  This service file is created in the directory `/lib/systemd/system/` in Beagle Bone Black to auto-run the bluetooth.py when reboot the Beagle Bone Black.

```
[Unit]
Description=My Service
After=syslog.target network.target
[Service]
```

```
WorkingDirectory=/usr/bin/
ExecStart=/usr/bin/python bluetooth.py
SyslongIdentifier=blue
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

- **p2.service**

  This service file is created in the directory `/lib/systemd/system/` in Beagle Bone Black to auto-run the install.py when reboot the Beagle Bone Black.

```
[Unit]
Description=My Game Service
After=syslog.target network.target p1.service
[Service]
WorkingDirectory=/usr/bin/
ExecStart=/usr/bin/python install.py
SyslongIdentifier=install
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

IV. Appendix

- **Beagle Bone Black Pinout**

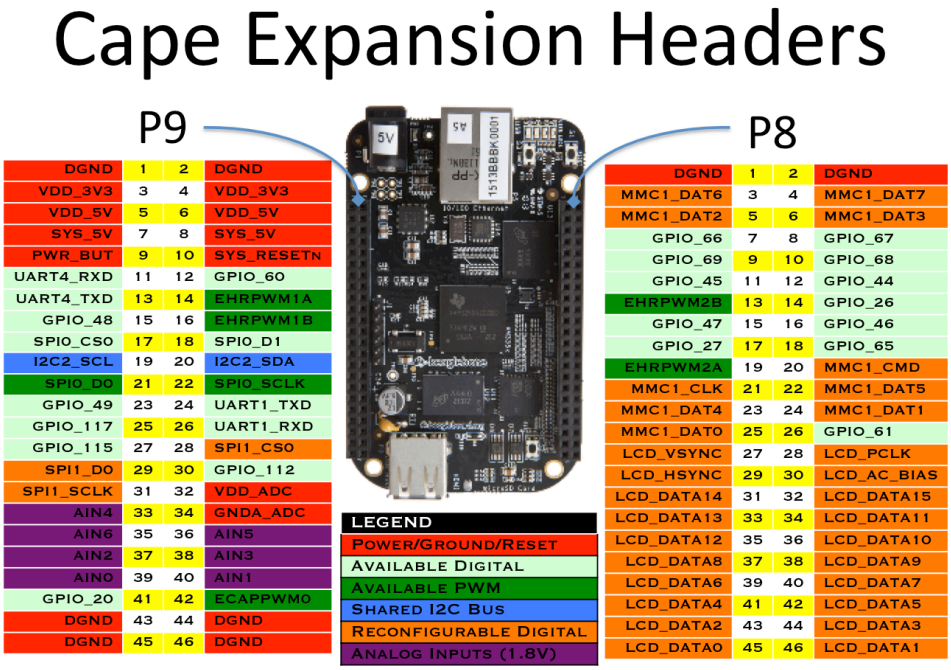

Fig 3. Beagle Bone Black Pinout

- **HS-W Control Function**

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Fig 4. H-SW Control Table

- **Snapshot during the game**

  0: represent nothing is at that position
  1: represents the blocks that user should avoid
  2: represent the car

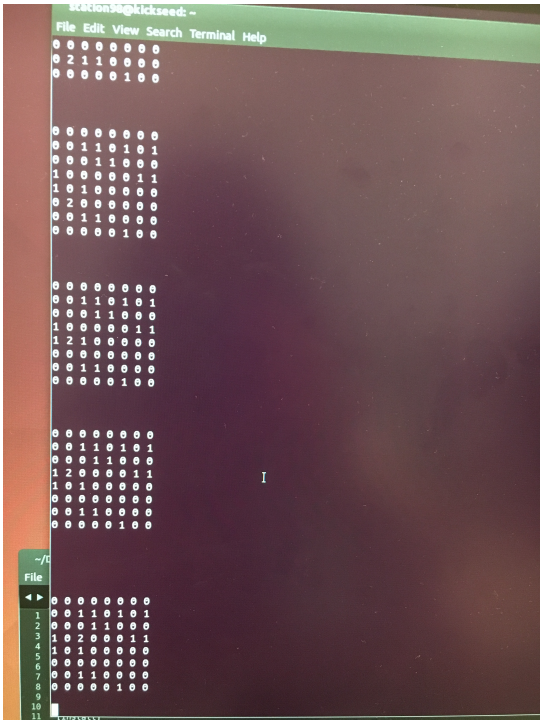

Fig 5. snapshot of the game

- **Create service file**

```
//instruction for service
cd /lib/systemd/system/
systemctl --system daemon-reload

// enable the service
systemctl enable p1.service
systemctl enable p1.service

// start the service
```

```
systemctl start p2.service
systemctl start p2.service

// check the status of the service
systemctl status p1.service
systemctl status p1.service
```

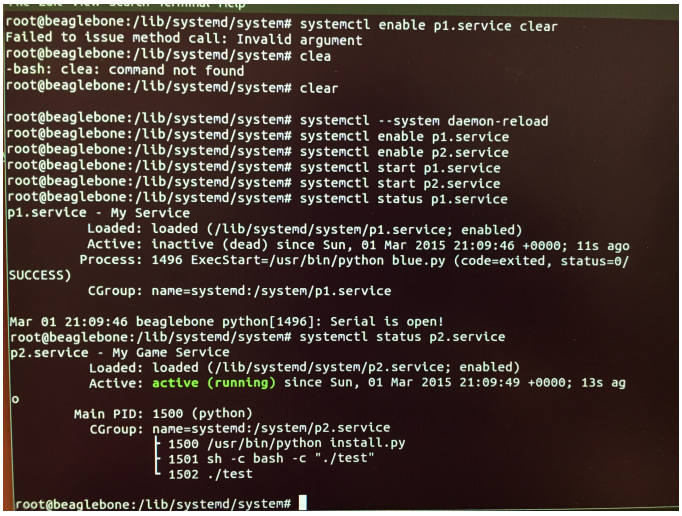

Fig 6. Status of the service file

- **Minicom**

We use minicom to simulate the process of transferring data from android device to Beagle Bone Black through Bluetooth device. The figure 7 below is when we testing the Bluetooth using minicom.

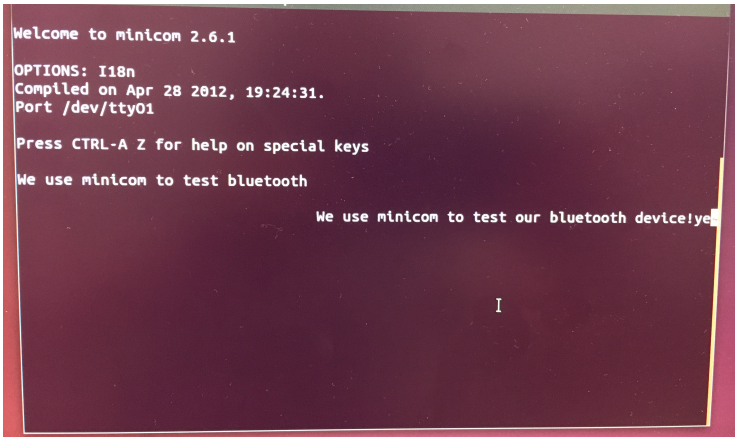

Fig 7. Test the Bluetooth with Minicom