

Fall 2020

Final Project

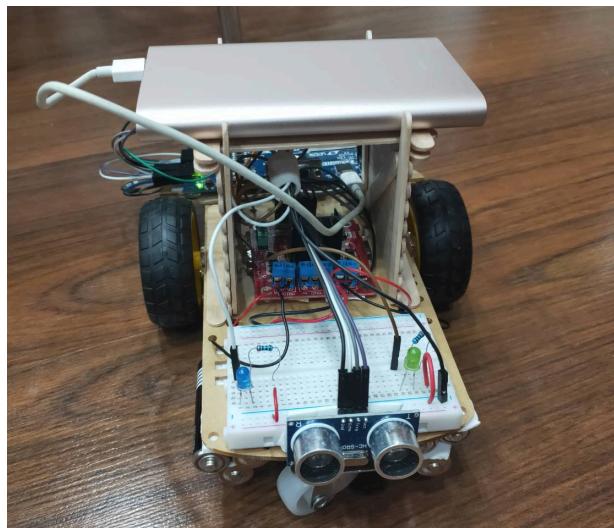
Team24

組長:丁緒慈(108062201)

組員:龐羽涵(107071019)

Project Name

藍牙遙控避障車

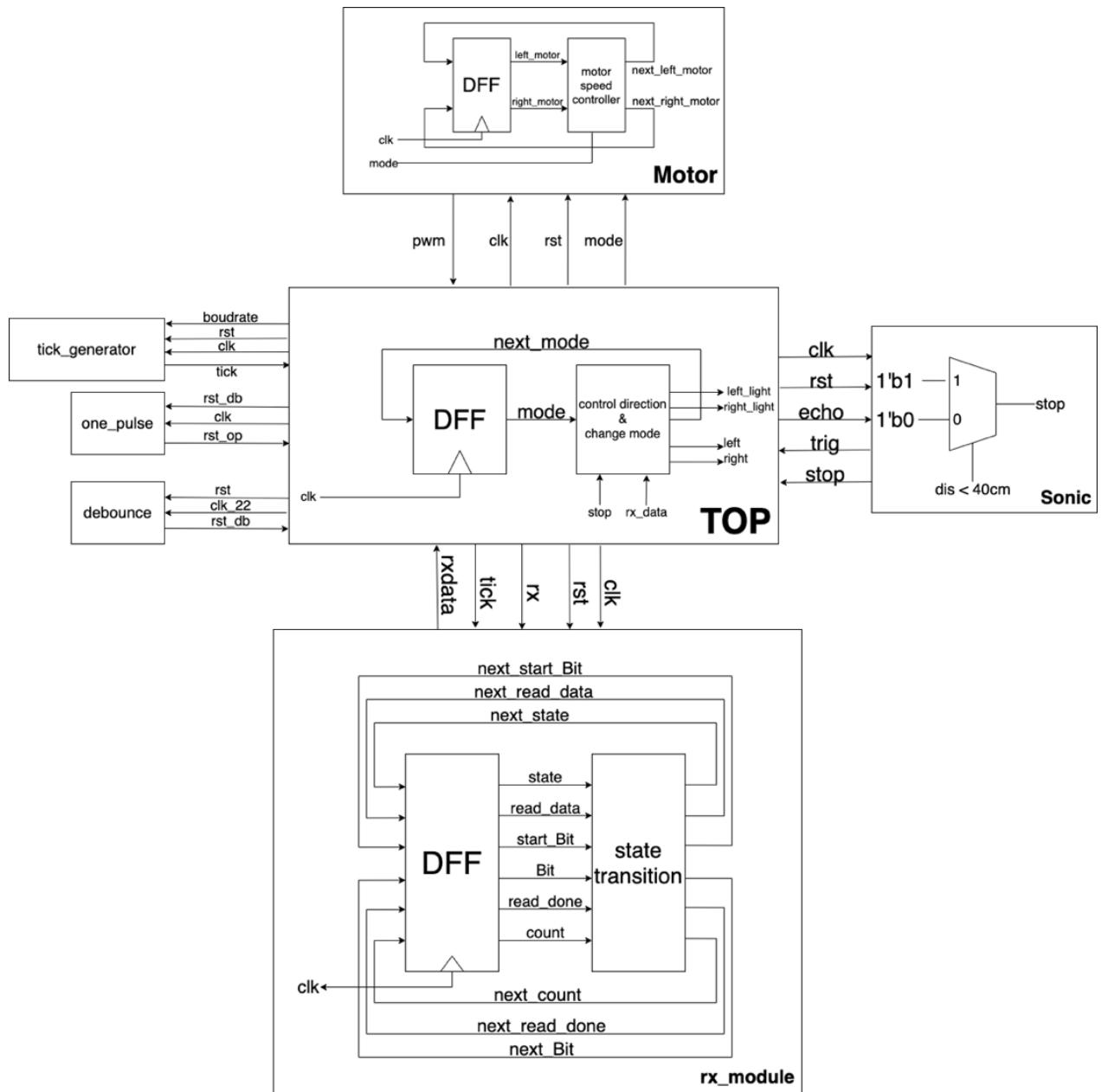


Function Introduction

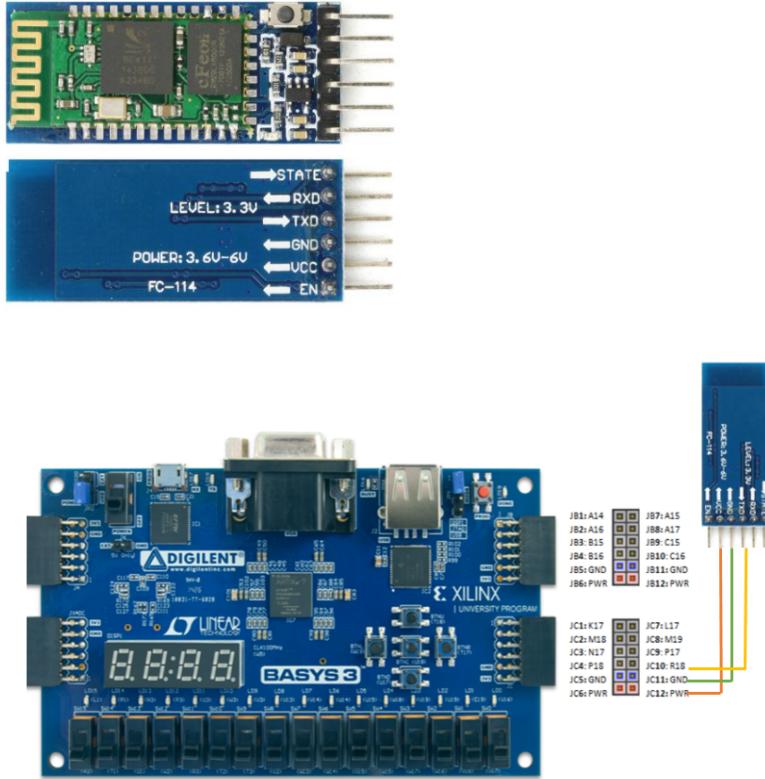
1. 使用者利用手機 APP 連接藍牙模組後，即可遙控避障車，並調整其速度（分成 Fast mode 和 Slow mode）。
2. 當避障車前方有障礙物時，只允許車避障車倒退。
3. 車子轉彎時，亮起相對應的車頭的方向燈。

System Specification

1. Block Diagram:



2. Bluetooth HC-05:



- How UART works:

UART 是一種 Serial communication，傳輸的 data 形式是一連串的 bit，我們可以透過單一個 pin 腳就能接收訊號。當有訊號傳輸時，訊號的第一個 bit 會是低電位，藉此我們可得之訊號要開始傳遞，並開始蒐集後續傳來的 bit。

- Implement rx

- rx_module

透過這個 module，我們可以讀取手機傳來的訊號(rx)，這個訊號會提供給馬達的 module 判斷如何移動車子。每次

手機傳送的訊號長度為 1 byte，第一個 bit 是低電位的 start bit，我們透過偵測 start bit，就可以知道甚麼時候要接收資料。緊接著，會有 8 個 bit 的資料 data，最後是一個 bit 的高電位 stop bit 代表資料已傳輸完畢。

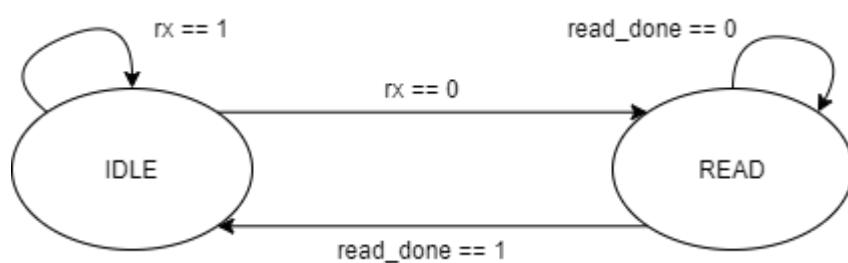


在 rx_module 中，有兩個 state，分別是 IDLE 和 READ。當 $rx == 0$ 時，也就是 start_bit 為低電位時，進入 READ 這個 state 讀取資料。當 $read_done == 1$ 時，也就是資料以讀取完畢，則回到 IDLE state 等待下一筆資料。

```
always @(*) begin
    case (state)
        IDLE : begin
            if (rx == 1'b0) next_state = READ;
            else next_state = IDLE;
        end

        READ : begin
            if (read_done) next_state = IDLE;
            else next_state = READ;
        end

        default : next_state = IDLE;
    endcase
end
```



在 Read 這個 state 時，為了確保資料的穩定性，我們要在 1 個 bit 的中間讀取資料，因此透過 tick 來得知目前傳送的進度，一個 bit 有 16 個 tick，當 count 數到 8 時，就是 bit 的正中間。當 start_bit==1 代表的是 start_bit 已經出現過，可以開始讀取資料，所以當位於 bit 的正中間且可開始讀取資料，我們就將 start_bit 和 count 歸零，代表這一次的資料已經開始讀取。

```
if (tick) next_count = count + 1;
else next_count = count;
next_start_bit = start_bit;
next_bit = bit;
next_read_done = read_done;
next_read_data = read_data;
```

```
if ((count == 4'b1000) && start_bit) begin
    next_count = 0;
    next_start_bit = 0;
    next_bit = bit;
    next_read_done = read_done;
    next_read_data = read_data;
end
```

接下來是開始讀取 data，由於剛剛我們已經在 start_bit 的正中間，再數 16 個 tick，就會是下一個 bit 的正中間，我們要在那個時候，將獨到的 data 記錄下來。直到 bit==8 且 stop_bit 為高電位時，我們讀完所有的資料，將 read_done 設為 1。

```

if ((count == 4'b1111) && (bit < 8) && tick) begin
    next_count = 0;
    next_start_bit = 0;
    next_bit = bit + 1;
    next_read_done = read_done;
    next_read_data = {rx, read_data[7 : 1]};
end

if ((count == 4'b1111) && (bit == 8) && rx) begin
    next_count = 0;
    next_start_bit = 1;
    next_bit = 0;
    next_read_done = 1;
    next_read_data = read_data;
end

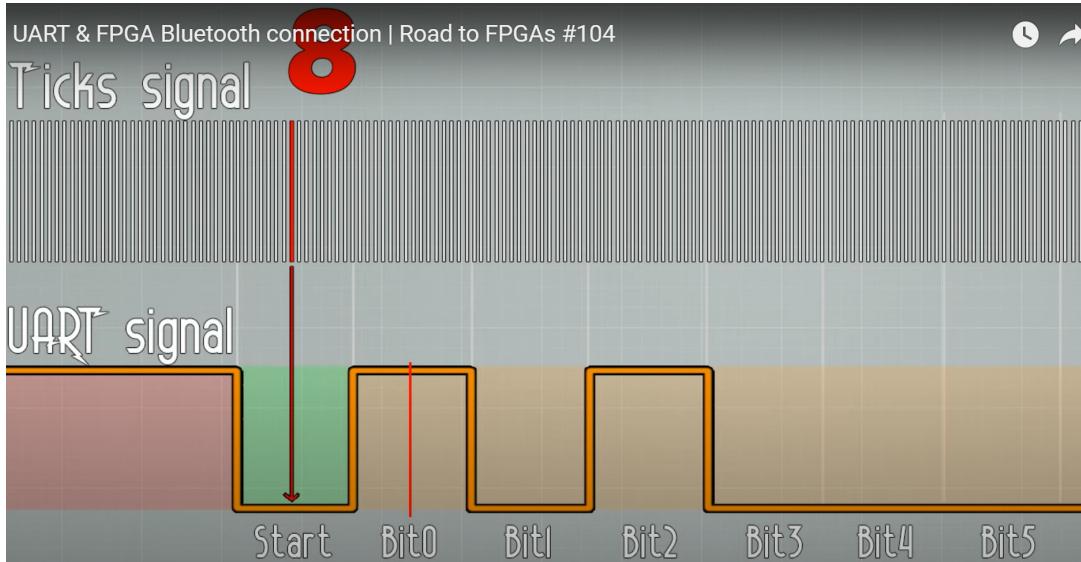
```

最後，由於藍芽傳輸時，是以最低有效位的方式傳輸，因此，我們需要將 `read_data` 的資料反過來放，也就是將 `rx` 放在 `read_data` 的最高位，每次要放下一個 `rx` 時，右移 `read_data`，最後 `read_data` 蔊集完 8 個 `bit`，再以 `rx_data` 作為 `output`。

```
next_read_data = {rx, read_data[7 : 1]};
```

- `tick_generator`:

由於 HC_05 的鮑率為 9600，而我們希望在 `bit` 的中間時讀取資料，避免讀到邊界值造成錯誤，因此我們透過這個 `module`，產生一個比藍芽傳輸鮑率快 16 倍的 `tick`，也就是說一個 `bit` 的時間長度相當於 16 個 `tick`，這樣我們就能在第 8 個 `tick` 的時候，知道我們在 `bit` 的中間。



因為 9600 的 16 倍是 153600，而 fpga 的 clock 是 100MHz，所以經過 651 個 clock cycle，就可以產生一個 tick。

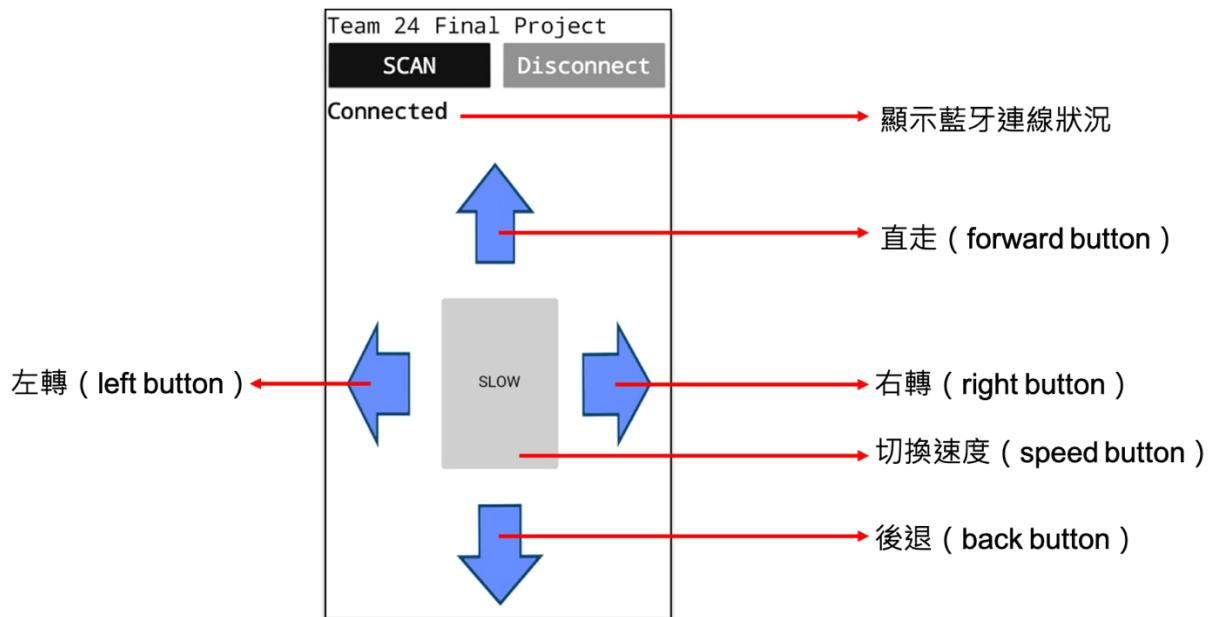
```
module tick_generator(clk, rst, baudrate, tick); // according to the book
    input clk, rst;
    input [16 - 1 : 0] baudrate;
    output tick;

    reg [16 - 1 : 0] count;
    wire [16 - 1 : 0] next_count;

    always @(posedge clk) begin
        if (rst)
            count <= 1;
        else
            count <= next_count;
    end

    assign next_count = (count == baudrate) ? 1 : (count + 1);
    assign tick = (count == baudrate);
endmodule
```

3. App for Bluetooth:



● 藍牙連線

```
when ListPicker1 .BeforePicking
do set ListPicker1 . Elements to BluetoothClient1 . AddressesAndNames

when ListPicker1 .AfterPicking
do if call BluetoothClient1 .Connect
address ListPicker1 . Selection
then set Label1 . Text to " Connected "

when disconnect .Click
do call BluetoothClient1 .Disconnect
set Label1 . Text to " Disconnected "

when Clock1 .Timer
do if BluetoothClient1 . IsConnected and call BluetoothClient1 . BytesAvailableToReceive > 0
then
```

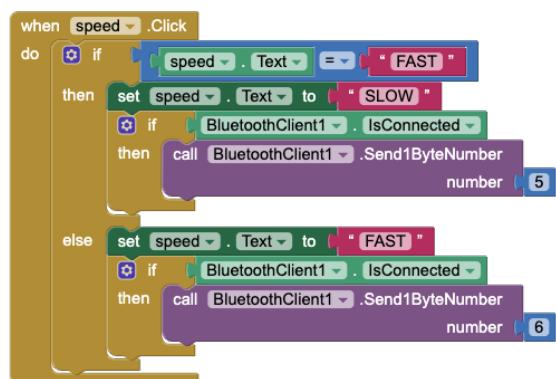
● 方向控制

方向	停止	直走	後退	右轉	左轉
傳送數字	0	1	2	3	4

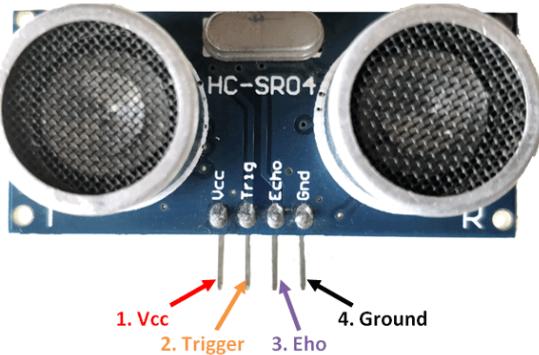


● 速度切換

速度	FAST	SLOW
傳送數字	5	6



4. Ultrasound (HC SR-04):



利用超聲波計算與障礙物間得距離，可由以下公式求得：

$$\text{距離(毫米)} = \frac{\text{時間(微秒)} \times \text{速率(毫米/微秒)}}{2}$$

若將聲速速率=0.344 毫米/微秒代入，約可得：

$$\text{距離(毫米)} = \frac{\text{時間(微秒)} \times 0.177}{2}$$

在PosCounter 中使用的 clock 是 1微秒，因此

distance_register 代表的是 sensor 幾微秒後收到反射信號，

PosCounter output 的 $\text{distance_count} = \text{distance_register} \times$

$100 / 58$ 與上面的公式對照後，可知 distance_count 必須乘以

0.1，單位毫米才是與障礙物之間的真實距離。我們設定安全距離為

60公分，因此當 $\text{distance_count} \times 0.1 \leq 600$ 時，車子必 須停

止， $\text{output stop} = 1'b1$ 。但是小數在合成的時候會出錯，因此等

式兩邊同乘 10，得 到 $\text{distance_count} \leq 6000$ 時 output stop

$= 1'b1$ 。

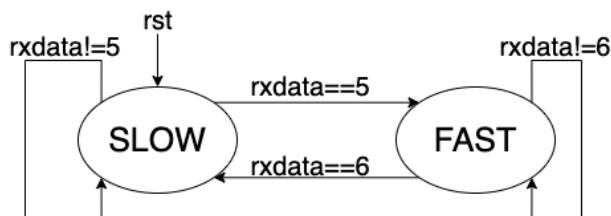
5. Motor:

- 速度控制

reset 時將速度設為 SLOW。

```
always @(posedge clk) begin
    if(rst_op)
        mode <= SLOW;
    else
        mode <= next_mode;
end
```

若接收到藍牙訊號 rxdata==8'd5，設定 next_mode 為 FAST；若接收到藍牙訊號 rxdata==8'd6，則設定 next_mode 為 SLOW；如果 rxdata 不是 5 和 6，則維持原來的速度模式。



```
case (rxdata)
    8'd5: next_mode = FAST;
    8'd6: next_mode = SLOW;
    default: next_mode = mode;
endcase
```

將 mode 作為 motor module 的 input，motor module 會根據 mode output 不同的 pwm。

```
motor A(
    .clk(clk),
    .rst(rst_op),
    .mode(mode),
    .pwm({left_motor, right_motor})
);
```

在 motor module 中，duty_cycle 的參數設定如下（將左右輪分開是因為考慮到兩車輪運轉時結構上的誤差，需要微調時較容易）：

```
parameter FAST_right = 10'd1000;
parameter SLOW_right = 10'd850;
parameter FAST_left = 10'd1000;
parameter SLOW_left = 10'd850;
```

接著根據不同的 mode，assign 不同的 duty_cycle。

```
always @(*) begin
    case (mode)
        FAST: begin
            next_left_motor = FAST_left;
            next_right_motor = FAST_right;
        end
        SLOW: begin
            next_left_motor = SLOW_left;
            next_right_motor = SLOW_right;
        end
        default: begin
            next_left_motor = SLOW_left;
            next_right_motor = SLOW_right;
        end
    endcase
end
```

將 duty_cycle 作為 motor_pwm 的 input，motor_pwm 會 output 相符合的 pwm signal。

```
motor_pwm m0(clk, rst, left_motor, left_pwm);
motor_pwm m1(clk, rst, right_motor, right_pwm);
```

- 方向控制

藉由左輪向前轉動、右輪停止達到右轉；藉由右輪向前轉動、左輪停止達到左轉，詳細的 signal 如下表所示：

	left[1]	left[0]	right[1]	right[0]
直走	1	0	1	0
後退	0	1	0	1
右轉	1	0	0	0
左轉	0	0	1	0
停止	0	0	0	0

然而避障車不完全受使用者的控制：若 stop_front==1 (即車子前方有障礙物)，只允許車子後退

```
always @(*) begin
    if (stop_front) begin
        case (rxdata)
            8'd2: {left, right} = {2'b01, 2'b01}; // backward
            default: {left, right} = {2'b00, 2'b00}; // stop
    endcase

```

當避障車前方沒有障礙物時，才能自由的前後左右移動。

```
end else begin
    case (rxdata)
        8'd1: {left, right} = {2'b10, 2'b10}; // forward
        8'd2: {left, right} = {2'b01, 2'b01}; // backward
        8'd3: {left, right} = {2'b10, 2'b00}; // right
        8'd4: {left, right} = {2'b00, 2'b10}; // left
        default: {left, right} = {2'b00, 2'b00}; // stop
    endcase
end

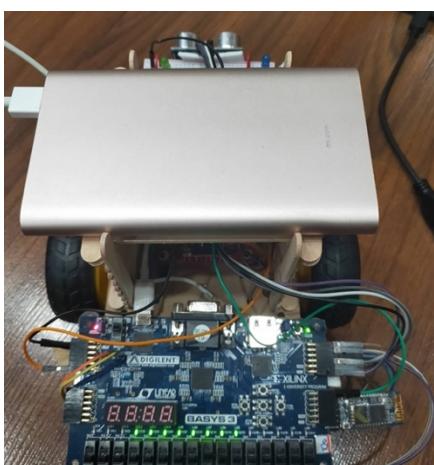
```

Conclusion

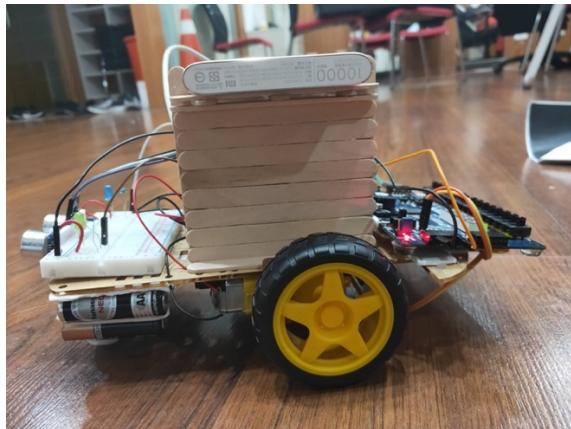
1. 學習心得：

在這次的 final project 製作過程中，光是購買材料就跑了許多趟，也經歷過買錯模組，怎麼測試都毫無反應的挫折。處理藍牙的部分，則是學習到了很多，像是一開始的時候，不知道鮑率該設多少，上網查了許多資料，才知道 HC-05 的傳輸鮑率是 9600，並且如果鮑率設錯的話，根本無法傳輸資料。此外藍芽的接線也是一大難題，我們原本將藍芽模組的 rx 腳接 fpga 的 rx 腳位，結果一直收不到資料，直到後來，我們才知道原來 tx 要接 rx 才對。

另外，我們原本想使用馬達驅動板的 5V 作為 FPGA 的電力來源，但發現當車輪轉動時，5V 的供電並不穩定，導致藍牙斷線，因此最後使用行動電源作為 FPGA 得電力來源，使藍牙通訊更加穩定。



最後在實際遙控避障車時，發現因為 FPGA 黏貼在避障車後，導致重心太後面，避障車直走時車頭會翹起，很不穩定，因此我們在避障車車頭底下黏貼電池，以增加重量，維持平衡。



2. 成員貢獻:

龐宇涵	藍牙模組、LED 燈
丁緒慈	藍牙 APP、馬達、超聲波

Reference

1. Bluetooth:

http://www.electrooobs.com/eng_circuitos_tut26.php

2. APP Inventor

<https://roboindia.com/tutorials/sending-receiving-with-hc05-mit-app-inventor/>