Team1_Final_Project_Report

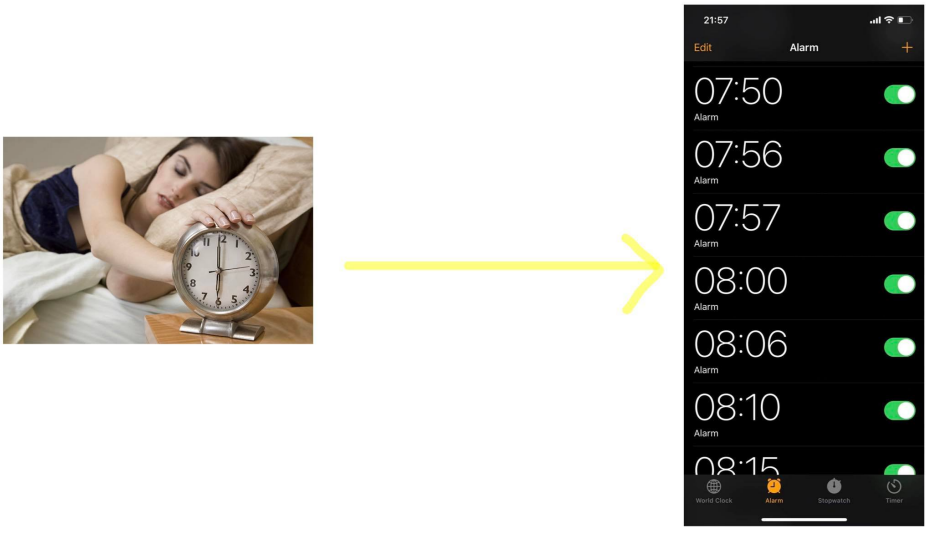
組員

107071016 施泳瑜
109062320 朱季葳

1. Introduction

我們的 Running Alarm 兼具鬧鐘、計時、碼表功能，且在鬧鐘以及倒數計時具備奔跑功能的奔跑型鬧鐘。這個特別設計出的鬧鐘可以幫助使用者不再需要經歷因為把鬧鐘按掉所以睡過頭的窘境，因為鬧鐘一響就會開始亂跑，使用者就要去追鬧鐘，而在追鬧鐘的過程可以順便醒醒腦！

2. Motivation



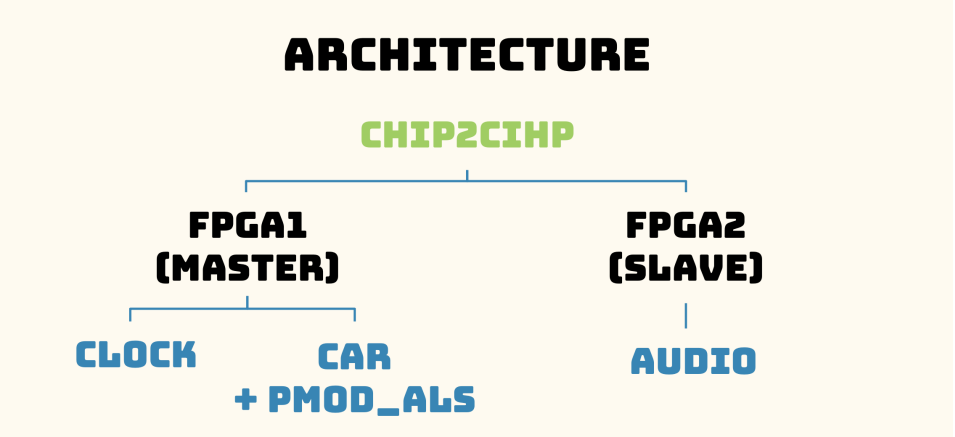
許多人每天早上大概都會面有上圖的情況：為了防止自己把鬧鐘按掉導致自己遲到，所以設置了一整排的鬧鐘，隔個五到十分鐘，甚至一兩分鐘就要響一次，而會發生這樣的窘境就是因為這**不是一個好的鬧鐘**！
所以我們得到了一個結論，鬧鐘除了要能夠在設定的時間響起之外，還要讓使用者不能輕易的就把它按掉！
[小結]
基於以上，我們對於心目中的**好的鬧鐘**規範出下列三點：

- 具備基本計時功能
- 有額外會飛會跑的追逐功能
- 將聲音弄得越 noisy 越符合期待

以下也將會詳述我們是透過哪些元素達成以上理想。

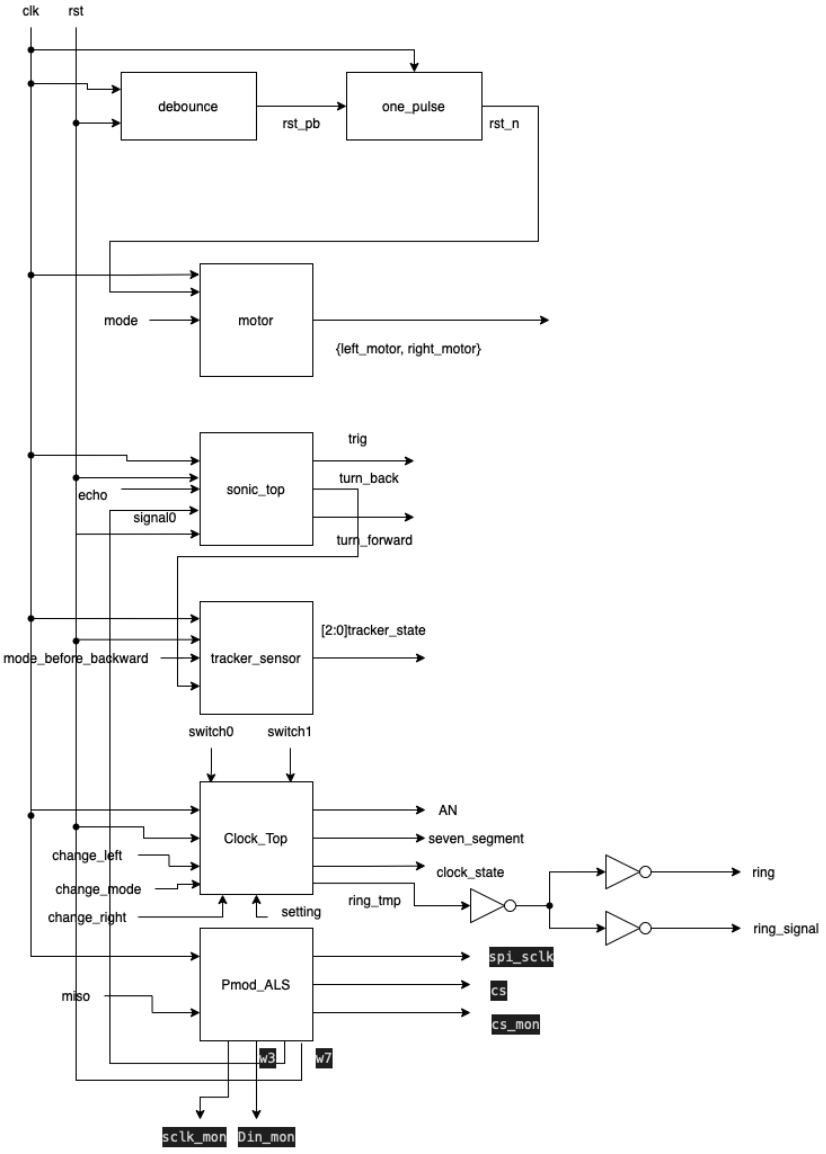
3. System specification

chip-to-chip architecture



Master chip

CAR

- block diagram
  - 

- 我們在 car_top module 裡面 instantiate clock_top module 以及 PmodALS module 去得到感光訊號以及 clock 會輸出的 AN, seven_segment, clock_state, ring_tmp 去進行相對應的輸出，而在 ring_tmp 這個訊號的處理上，我們將此訊號做 Fan-out 再得到兩個訊號，分別是 ring 以及 ring_signal，其中的 ring_signal 是用來傳遞到 slave chip 的訊號，ring 則是用來在 master chip 上做 LED 燈顯示的訊號。

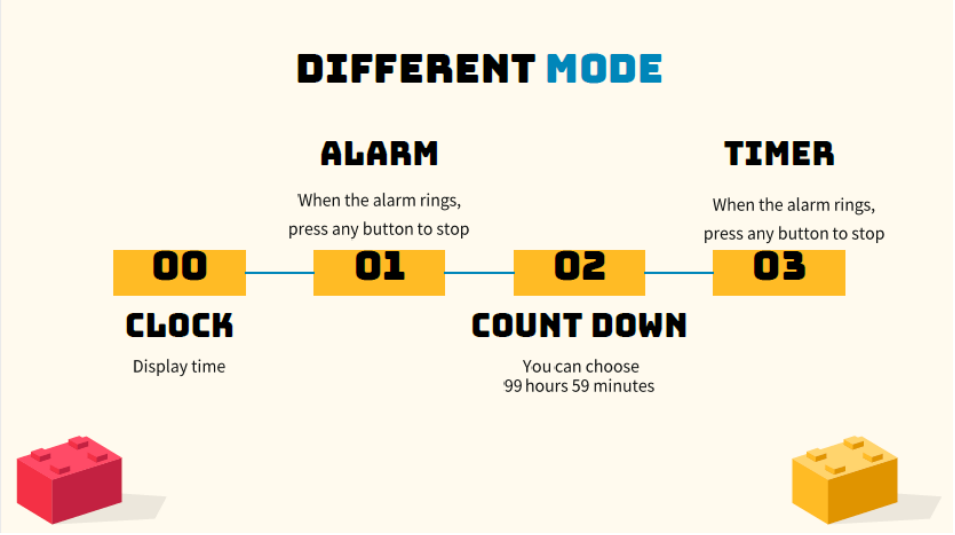
    - 而在其餘的部分，我們藉由改動原本在 Lab6 car 中實作的 code 來完成。

        - 加入了 turn_back、mode_before_backward：在前方有障礙物時，升起turn_back (於 sonic_top module 內進行實作) ，並記錄當時車子轉的方向 (mode_before_backward)，再把 turn_back 以及 mode_before_backward 傳入 tracker_sensor module，並用 LFSR (LFSR 取末兩碼的方式代表前進 (10,11)、左轉(00)、右轉(01)) 去選出和 mode_before_backward 相異的方向 ➥ 可確保方向不重複
        - 加入 turn_forward：利用 Pmod_ALS 所輸出的 w3, w7 輸入到 sonic_top module，並於 { w3, w7 } == 2'b00 (代表處於暗的環境) 維持八秒後升起 turn_forward ➥ 可確保鬧鐘確實在暗處
        - 而如果上述所說的兩者 turn_back 以及 turn_forward 同時升起的話，就讓車子停下來 (STOP)。會這麼做是因為若前方又障礙物無法前進，又考慮到鬧鐘車車跑到暗處，如：床底下、桌底下...的地方，為了不要讓使用者找不到鬧鐘而造成無法關掉它的結果，所以就設定讓鬧鐘車車停下來 ➥ 可確保鬧鐘不會東撞西撞 or 跑到找不到的地方。
        - 若只有 turn_back 升起：代表前方有障礙物，就應該要換方向，而我們在讓車子倒退一秒之後，就會透過前方所說的 LFSR 機制換上新的 tracker_state。
        - 若只有 turn_forward 升起：讓車車往前走 (CENTER) 直到 turn_forward 訊號降下來為止。

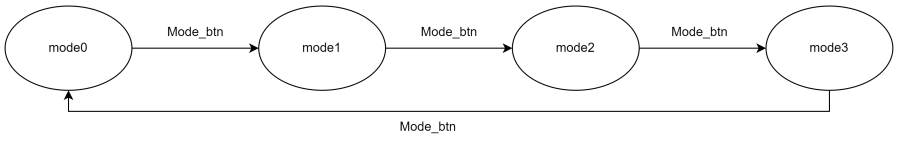
PMOD ALS (LIGHT SENSOR)

- 這個元件是透過 SPI protocol 與我們的 fpga 板互動，在成功連結元件之後，完成將光訊號從 analog data 轉為 8 位元 digital data 的結果。
- 我們將接收到的亮度 > 8'b00001111 設為「弱光模式」，亮起 fpga 板上的設定的 LED，並將輸出 w3 設為 1；而在收到的亮度 > 8'b11111111 設為「強光模式」，亮起 fpga 板上的設定的 LED，並將輸出 w7 設為 1，依照以上可精準確認現在環境光的強度。
- 將 w3、w7 輸出至 car_top module，作為判斷是否進到暗處的車子加裝感測器，若為 yes 則讓車子做出相對應的**倒車、轉向**動作。

CLOCK

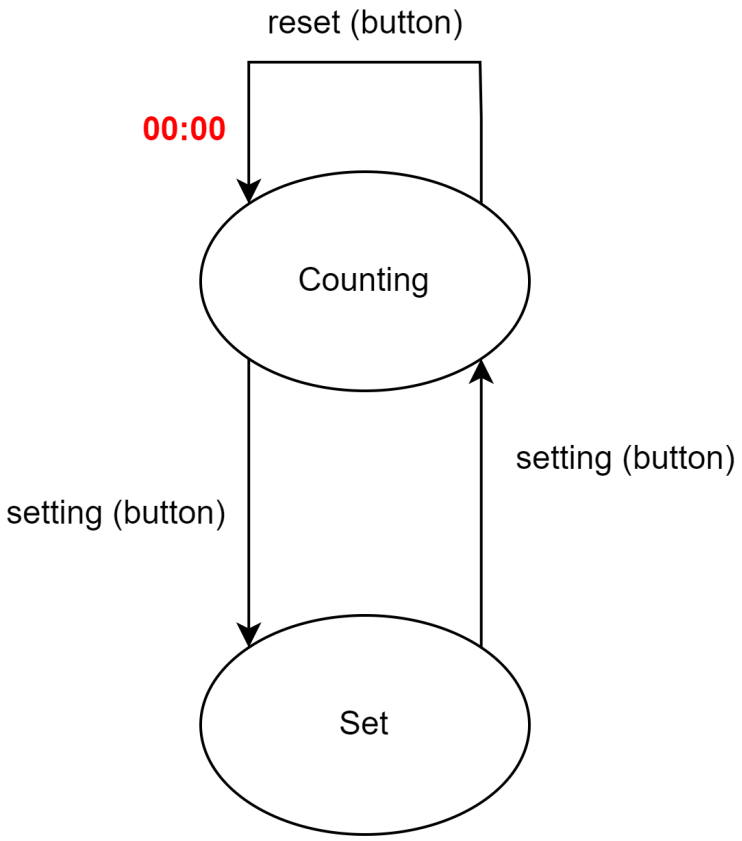


- 改變 mode 的 state diagram

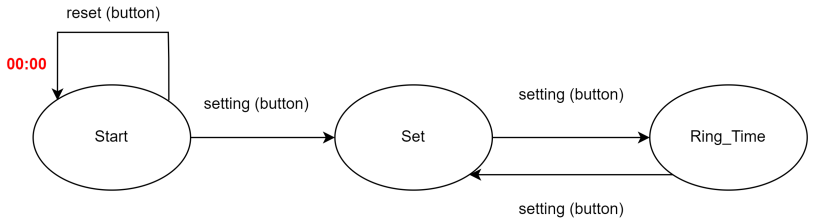


- state diagram for different mode

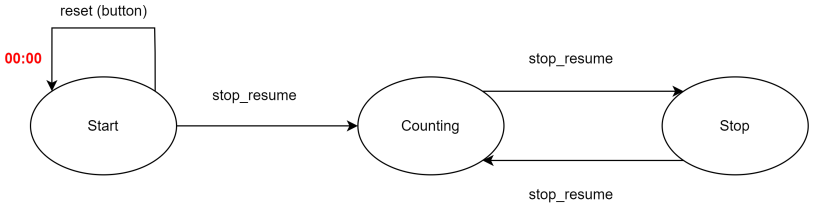
○ mode0 : clock



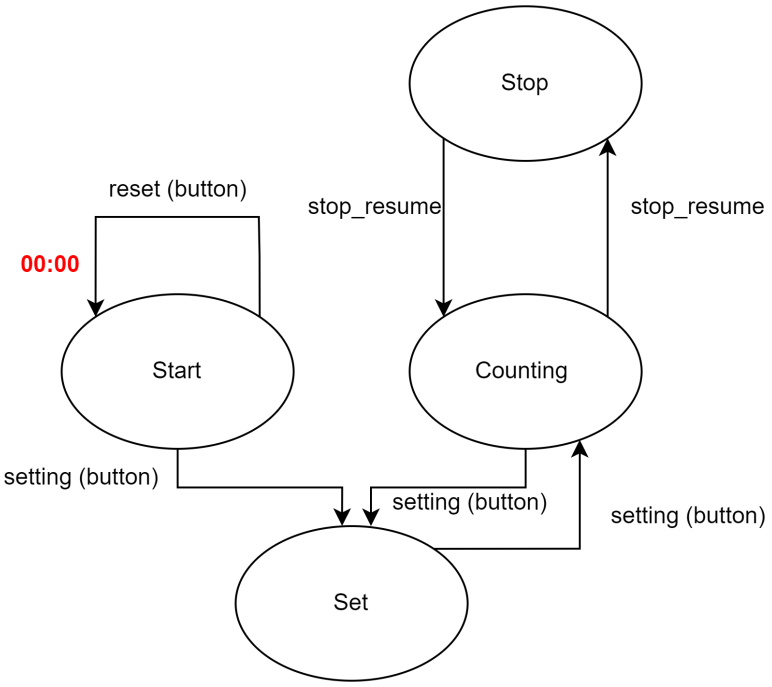
○ mode1 : alarm



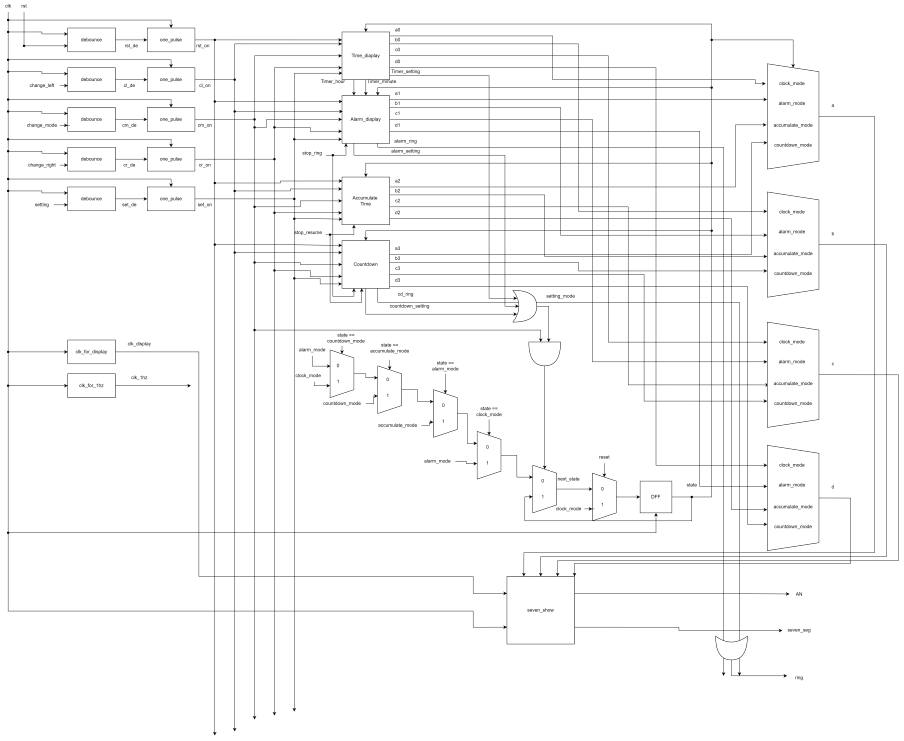
○ mode2 : timer



- mode3 : countdown



- block diagram for clock_top module



- switch
  - switch0–>與上一個 cycle 時的 switch0 狀態比較
    - 不同：升起 stop_ring
    - 相同：維持原狀態
  - switch1–>與上一個cycle時switch1的狀態比較
    - 不同：升起 stop_resume_count

        - 相同：維持原狀態

- button：進行模式切換、設定
    - BTNU：Reset–> wire rst
    - BTNC：Change Mode–> wire change_mode
    - BTNL：Change left display–> wire change_left
    - BTNR：Change right display–> wire change_rignt
    - BTND：每個 state 裡做 setting–> wire setting

- 詳細說明
    - clock_top
        - **fundamental**：我們在 clock 中將相對應的 button 訊號、目前的 state ([2-1:0] state) 以及 clk 接入 Time_display, Alarm_display, Accumulate_time, countdown module
        - Alarm_display module 還需要接入 Time_display module 所輸出的 Timer_minute, Timer_hour 來與設定的鬧鐘時間比較，如果相同的話就把 ring 升起 (clock_top module 中，將 alarm_ring 接到 Alarm display module 屬於 ring 的那個 port)
        - Alarm_display, countdown module 都會有觸發鈴響的功能，所以把 stop_ring 分別接入這兩個 module 的 stop_ring port
        - Accumulate_time, countdown module 需要有暫停碼表、暫停倒數計時的功能，所以把 stop_resume_count 接入這兩個 module 的 stop_resume port
        - **for setting mode**：各個 mode 一但進入 setting mode, 就要等他們解除 setting mode 才能切換到其他 mode，所以 Timer_display, Alarm_display, countdown module 都會輸出代表他們目前是否在 setting mode 的訊號 (Timer_setting, Alarm_setting, Countdown_setting 分別接到各個 module 的 setting sort)，並只有在各個 mode 解除 setting mode 並按下 BTNC 時才能夠切換到下一個mode 去

        - Time_display, Alarm_display, Accumulate_time, countdown module 都會個別輸出他們的 ai, bi, ci, di array 訊號 (代表四個七算顯示器上面，各個 digit 所對應到的 seven segment 訊號)，並依據現在所屬的 state 去將相對應的 array assign 給 [7-1:0] a, b, c, d，再把 a, b, c, d 以及 clk, clk_display (clk for seven segment display) 接入 seven_show module，再將 [3:0]AN, [6:0] seven_seg 接到 output port，其中
            - i = 0–> timer
            - i = 1 -->alarm
            - i = 2 --> accumulate time
            - i = 3 --> countdown)
    - Timer_display (時, 分)
        - 設定目前時間
        - 依據設定的時間來依據經過的秒數增加時 / 分：過了 3600 秒就增加一小時，過了 60 秒就增加一分鐘
    - Alarm_display (時, 分)
        - 設定鬧鐘時間
        - 對比目前時間以及鬧鐘時間，兩者相同時把 ring 升起
        - 當 ring 升起時，stop ring 的 switch 被切換，就把 ring 降下來
    - Accumulate_Time (分,秒)
        - 當 stop_resume 的 switch 被切換時，便**開始**算時間或者**停止**算時間
        - 為了防止使用者忘記把碼表關掉，讓他無止盡的算時間，於是設置一個上限–> 當時間到達 99 分 59 秒時，就將碼表維持在 99:59
        - 每經過 60 秒就加一分鐘
    - Countdown （時,分)
        - 設定倒數計時時間
        - 當時間數盡，到達 00:00 時，把 ring 升起

- 當 ring 升起時，stop ring 的 switch 被切換，就把 ring 降下來
- 當 stop_resume 的 switch 被切換時，**開始**倒數或者**停止**倒數

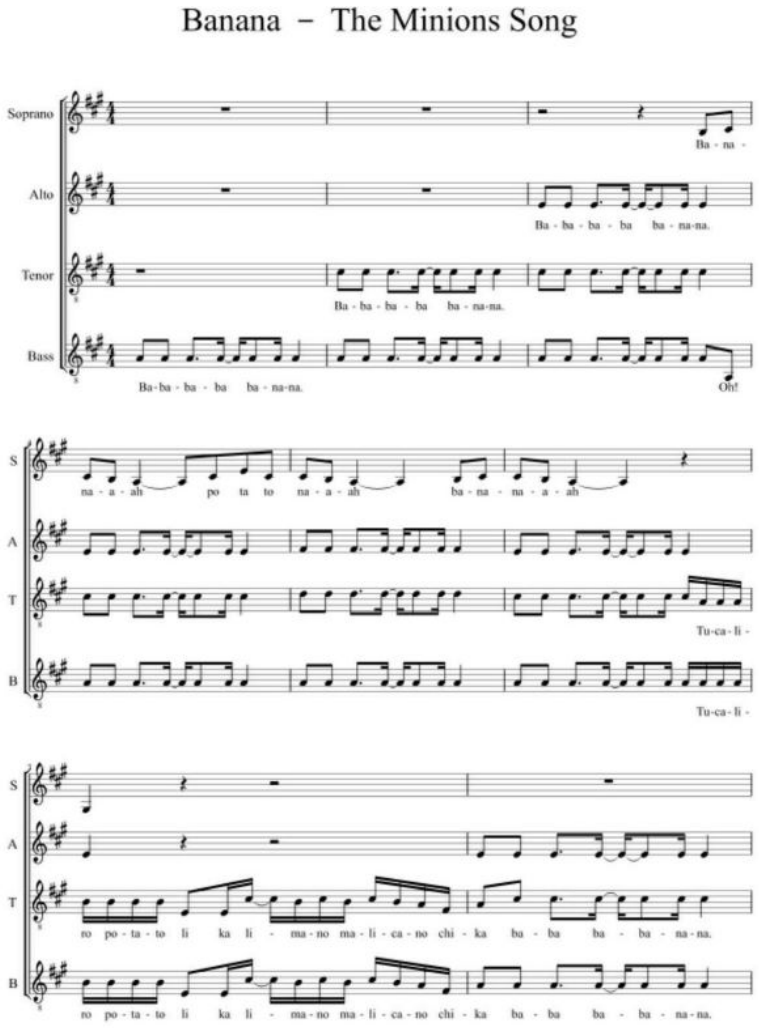
Slave chip

AUDIO

[鈴響譜設計]

- 目標歌曲

- 依以上歌曲根據五線譜改寫 Lab5 Music 的 Music module 去刻出 spectrum



4. Experimental results

- verilog code
  - 一開始我們先將要做的部分分成3個part：clock, PmodALS, Music 來分別實作
  - 接著將 clock 以及 Music 結合，確認鈴響的時候會有聲音
  - 再來將 Lab6 所實作的 car 以及 PmodALS 結合，**用環境光的亮暗程度輔助車子奔跑路徑**，並修改原本的 running rule，改為只會往前、後、左、右四個方向移動，並用 LFSR 輔助進行重找方向的工作
  - 最後將 car + PmodALS, clock + Music 結合到一塊版子上，但是發現會出現一些硬體上面的問題，所以最後改為 master chip 負責 car_PmodALS+clock, slave chip 負責 Music
- 硬體組裝
  - 由於同時接上車子與聲音模組會發生電壓不夠的問題，因此採用分兩塊板子實作 (應用Lab6 chip2chip)

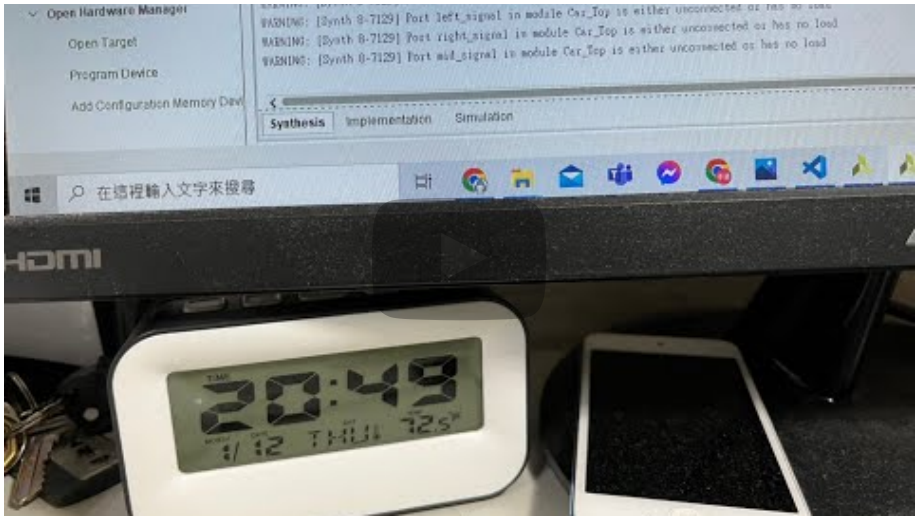
- 同電源：由於兩個不同的行充可能會有不同的傳輸 protocol 導致訊號不穩，所以我們最後將兩塊板子的電源都使用同一個行充來供應。
- 最後在測試的時候可變電阻燒壞了，但是當時不確定是可變電阻壞了還是喇叭，所以添購了新的可變電阻跟喇叭並接到聲音模組

5. Conclusion

在這次的 final project 中，我們實現了心目中理想的鬧鐘，兼具了所歸納出 good clock 的三項功能：基本顯示時間、動態移動、吵鬧的鈴聲，藉此可以完善我們所認為現在鬧鐘不足的部分，在邏輯設計實驗課程中學以致用，成功解決生活中的困難點。在作法上，扎實結合了上課所學的 chip2chip、car、LFSR、speacker…內容，並衍生到自己所撰寫出的具備各 state 的鬧鐘，也結合了外接元件、刻劃特定歌曲的譜、相關的硬體探索...，也確實在巨人肩膀 (這堂課的老師、所有助教) 上成功完成了一項新的任務！
做這項 final project 的時候，再次切身體會這領域的困難點，我們常不知道到底是軟體寫壞了 or 硬體零件失效的問題，而走了許多冤枉路，曾在最後幾天買零件卻發現零件根本沒壞、debug de 半天才發現 .v 檔的 module 忘了加入 output 值、將類比訊號當作數位處理...。過程雖然繁雜，但現在回首，和隊友披荊斬將的經驗是真不錯，這一個月是相當百感交集且富足的一段回憶，大家辛苦了。

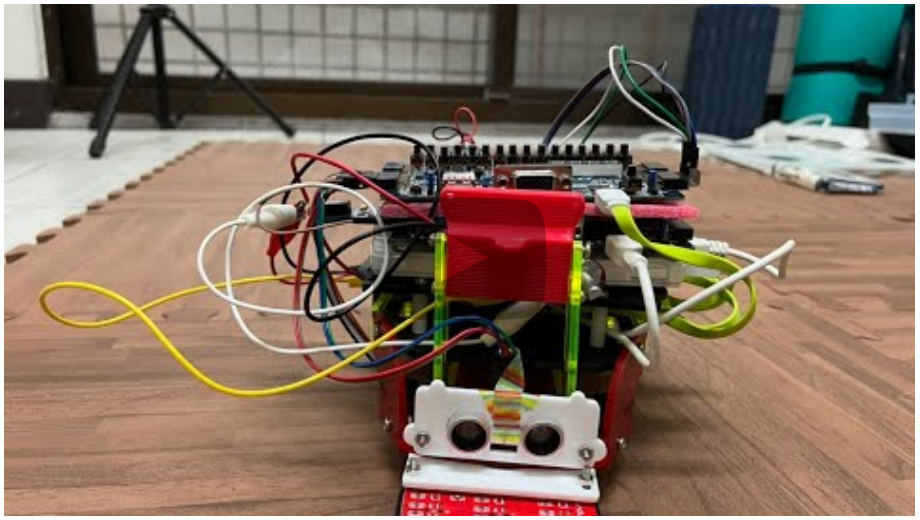
6. Demo video

- 鬧鐘 mode1~4 的設置方法



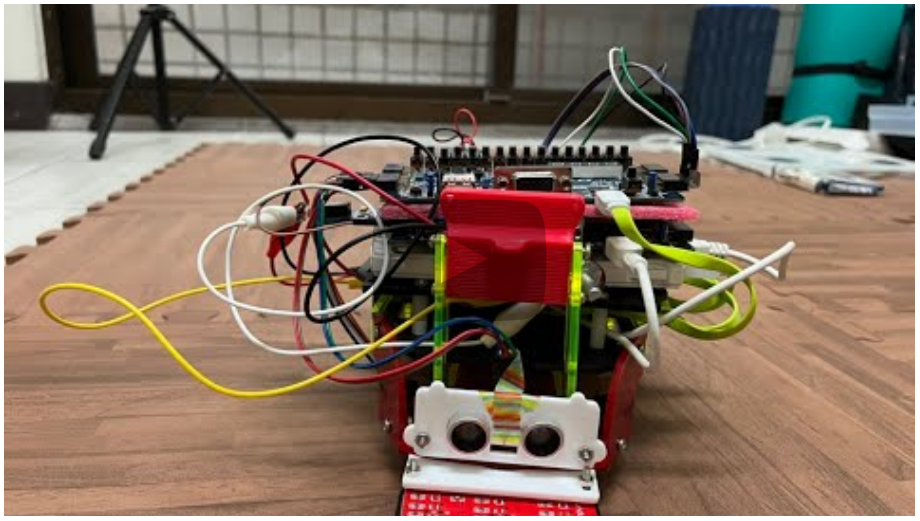
Link (https://reurl.cc/zr91RN)

- 鬧鐘模式鈴響



Link (https://reurl.cc/rZW9mr)

- 倒數計時鈴響



Link (https://reurl.cc/85mvEy)

7. Reference

Pmod_ALS (https://reurl.cc/Z1xgdp)