

# Lab 5: Keyboard and Audio Modules

Group 21: 陳克盈 (112062205)、蔡明姍 (112062224)

## Table of Contents

<b>1 Q1: Sliding Windows sequence detactor</b>	<b>2</b>
1.1 State diagram . . . . .	2
1.2 Implementation . . . . .	3
1.3 Simulation . . . . .	4
<b>2 Q2: Traffic light controller</b>	<b>4</b>
2.1 State diagram . . . . .	5
2.2 Implementation . . . . .	5
2.3 Simulation . . . . .	8
<b>3 Q3: Greatest common divisor</b>	<b>8</b>
3.1 Implementation . . . . .	8
3.2 Simulation . . . . .	11
<b>4 FPGA1: Mixed keyboard and audio modules together</b>	<b>12</b>
4.1 Implementation . . . . .	12
<b>5 FPGA2: vending maching</b>	<b>16</b>
5.1 Implementation . . . . .	16
<b>6 Other</b>	<b>20</b>
6.1 What we have learned . . . . .	20
6.2 分工 . . . . .	20

## 1 Q1: Sliding Windows sequence detector

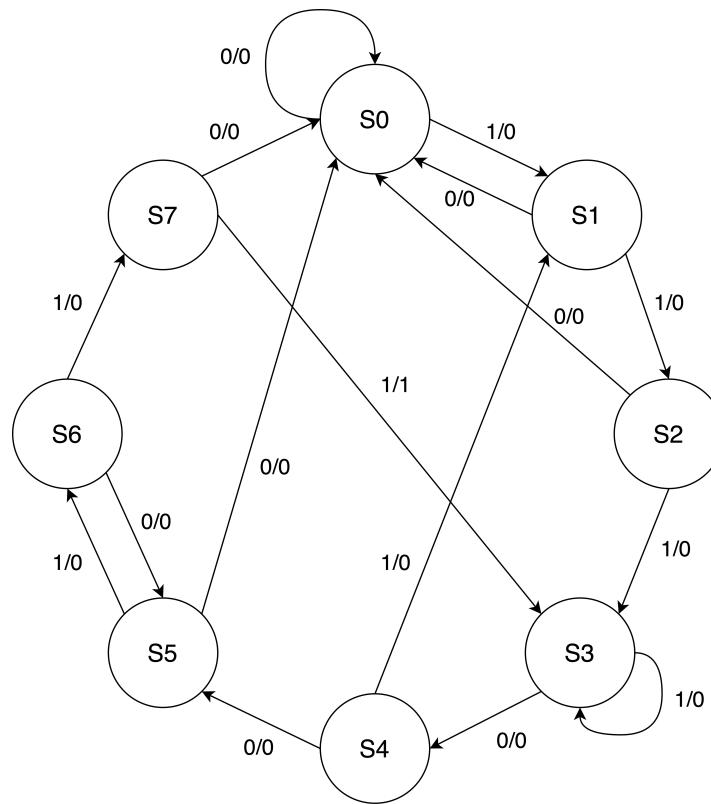
- input clk: clock
- input rst\_n: reset
- input in: input data
- output dec: detect signal

這題需要我們實作一個 Mealy machine，用來偵測輸入的序列是不是滿足  $1110(01) + 11$  這個 regular expression。

### 1.1 State diagram

首先我們需要先畫出狀態圖，狀態的定義如下：

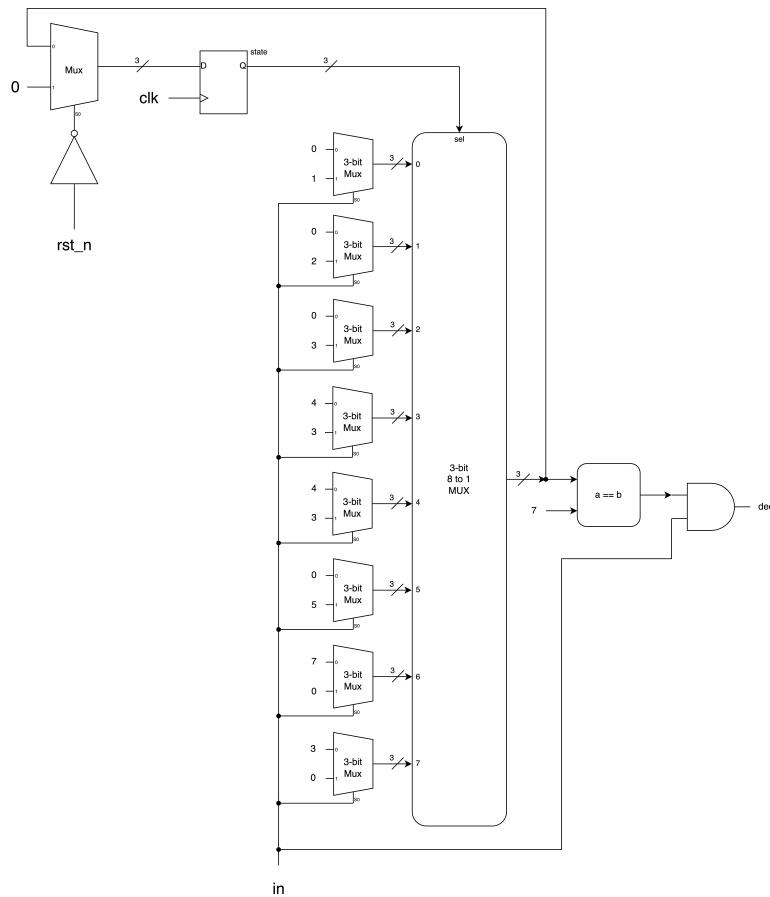
- S0: 初始狀態
- S1: 偵測到 1 的狀態
- S2: 偵測到 11 的狀態
- S3: 偵測到 111 的狀態，因此接收到 1 之後，還是代表前面有 111，因此會繼續保持 S3。
- S4: 偵測到 1110 這個狀態，因此接收到 1 之後，會回到只接收 1 的狀態，也就是 S1
- S5: 偵測到 11100 的狀態，由於下一個狀態 S6 需要輸入 1 才會達到，因此當輸入 0 的時候就必須回到 S0 的狀態
- S6: 偵測到 111001 的狀態，由於 (01) 可以多次出現，因此輸入為 0 的時候，可以回到 S5 重新偵測 (01)。
- S7: 偵測到  $1110(01)+1$  的狀態，如果輸入為 1，就代表符合 regular expression，但因為這個 detector 是 Sliding Window 的，不會因為偵測到符合的 sequence 就直接重設，因此輸入 1 之後會再回到 S3。



**Fig. 1** Q1 State diagram

## 1.2 Implementation

首先是狀態偵測的部分，我們使用 switch case 的語法，針對目前的狀態以及輸入的值，來決定下一個狀態。另外，因為這是一個 Mealy machine，dec 的值是不受 clock 影響的，因此 dec 是額外計算的，只要當前狀態是 S7，且輸入是 1，就會馬上輸出 True。



**Fig. 2** Q1 Circuit

### 1.3 Simulation

我們重現了題目上的波形圖，可以發現當輸入的序列符合 regular expression 的時候，dec 會變成 1。



**Fig. 3** Q1 Waveform match



**Fig. 4** Q1 Waveform mismatch

## 2 Q2: Traffic light controller

- input clk: clock

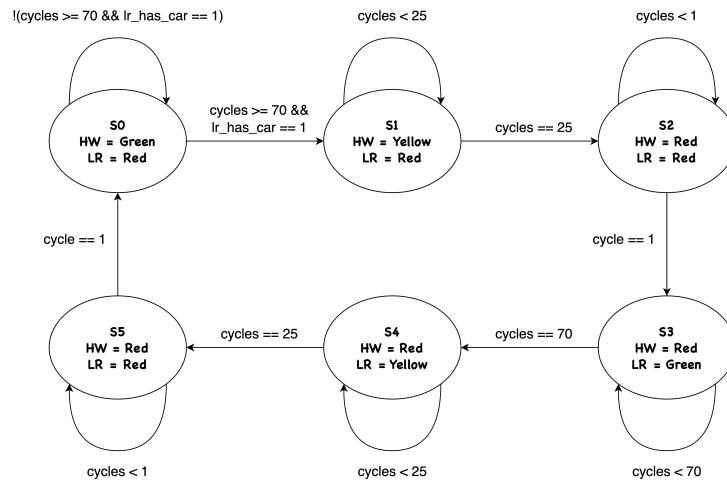
- input rst\_n: reset
- input lr\_has\_car: Local road has car
- output hw\_light: Highway light
- output lr\_light: Local road light

這題需要我們實作一個紅綠燈的控制器，控制一個由 Highway 和 Local road 兩條道路所組成的交叉路口的紅綠燈。

由於 Highway 的優先度最高，因此這個流程分為六個狀態（三種顏色在程式碼中以二進位表示，Green: 100, Yellow: 010, Red: 001）：

- (1) HW = Green, LR = Red: 如果已經保持這個狀態  $\geq 70$  個 cycle 以上，且 Local road 有車，那就進到下一個狀態。
- (2) HW = Yellow, LR = Red: 黃燈階段，保持 25 個 cycle。
- (3) HW = Red, LR = Red: 兩邊都紅燈，保持一個 cycle
- (4) HW = Red, LR = Green: Local road 綠燈，保持 70 個 cycle
- (5) HW = Red, LR = Yellow: Local road 黃燈，保持 25 個 cycle
- (6) HW = Red, LR = Red: 兩邊都紅燈，保持一個 cycle 後回到第一個狀態。

## 2.1 State diagram

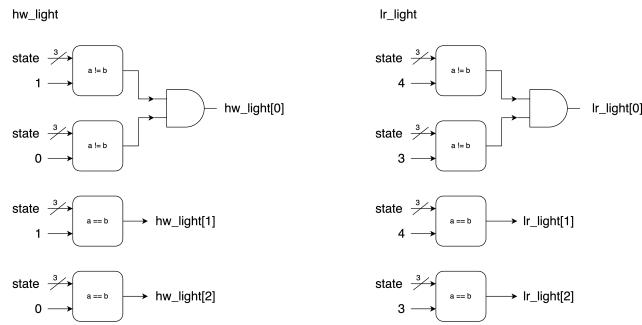


**Fig. 5** Q2 State diagram

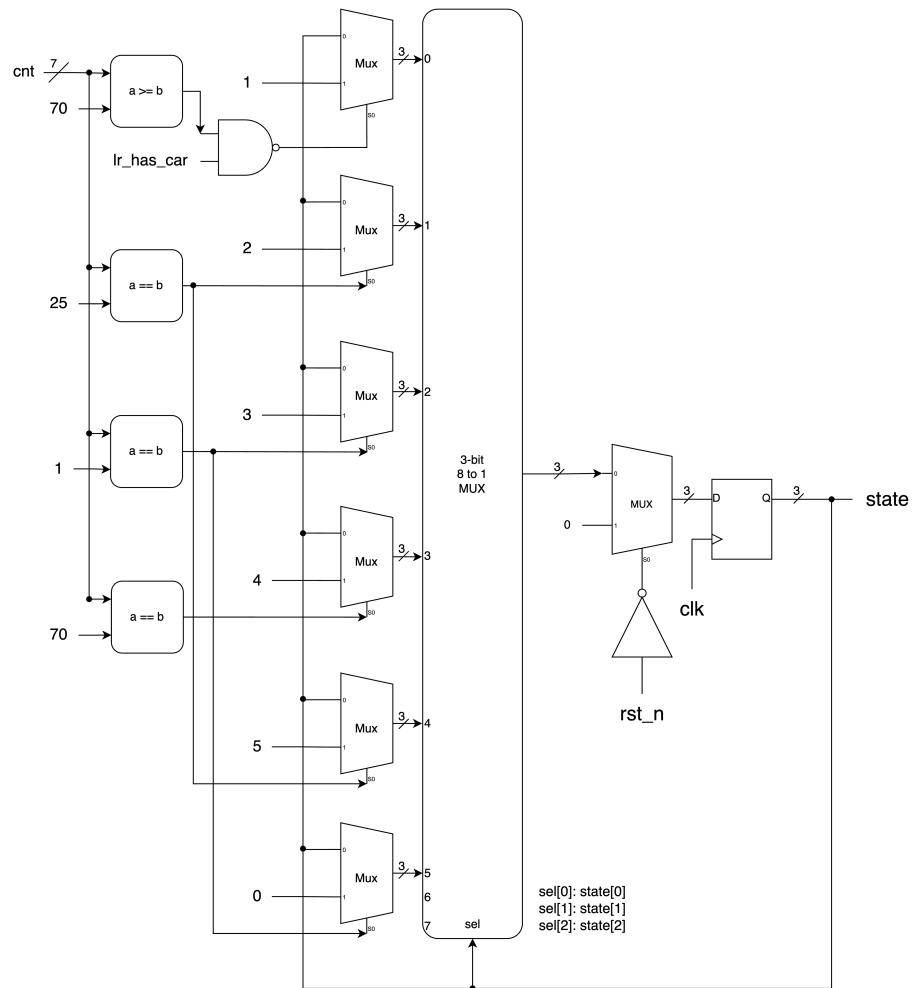
## 2.2 Implementation

跟前一題的實作方法相似，使用 switch case 由目前經過的 clock cycle 以及狀態來決定下一個狀態是什麼。

首先是紅綠燈的判定：

**Fig. 6** Q2 Light Circuit

接著是狀態的判定：

**Fig. 7** Q2 Circuit

然後是計數器的判定：

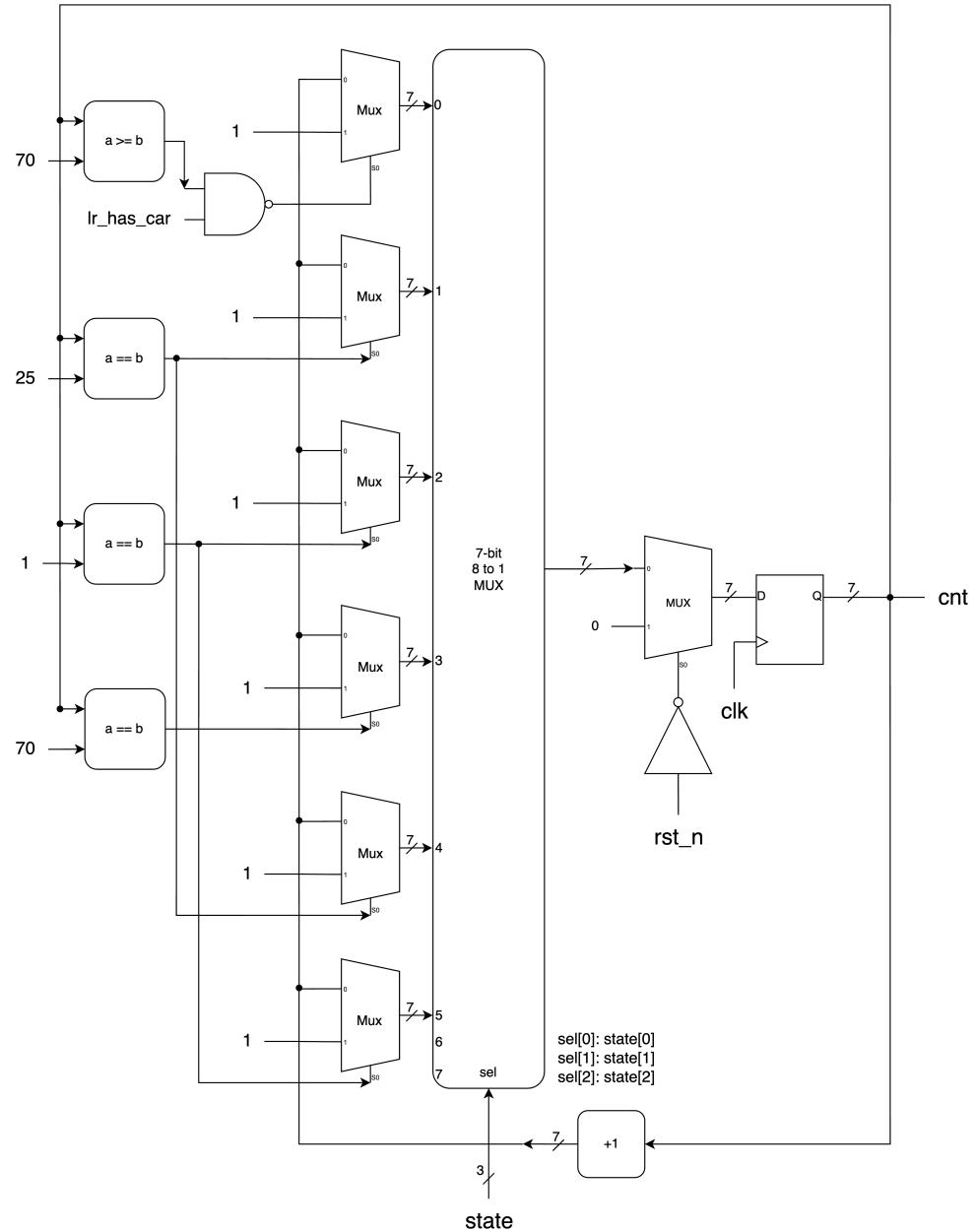


Fig. 8 Q2 Counter Circuit

### 2.3 Simulation

我們重現了題目上的波形圖：



**Fig. 9** Q2 Waveform

### 3 Q3: Greatest common divisor

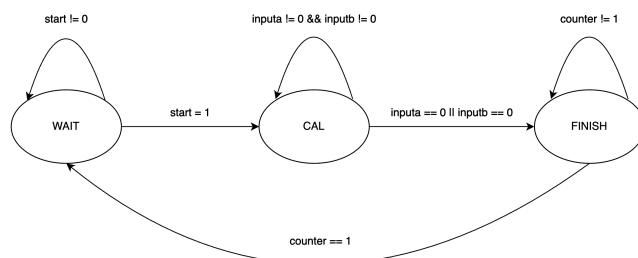
- input clk: clock
- input rst\_n: reset
- input start: start signal
- input [15:0] a, b: input numbers

這題我們需要實作一個利用輾轉相除法計算 GCD 的模組，整體分為三個狀態：

- (1) WAIT: 等待 start signal，當收到 start signal 的時候，就 fetch 輸入的 a, b，並進入到下一個狀態
- (2) CAL: 利用輾轉相除法計算 GCD
- (3) FINISH: 計算完畢，將 gcd 輸出，在兩個 clock cycle 後回到 WAIT 狀態

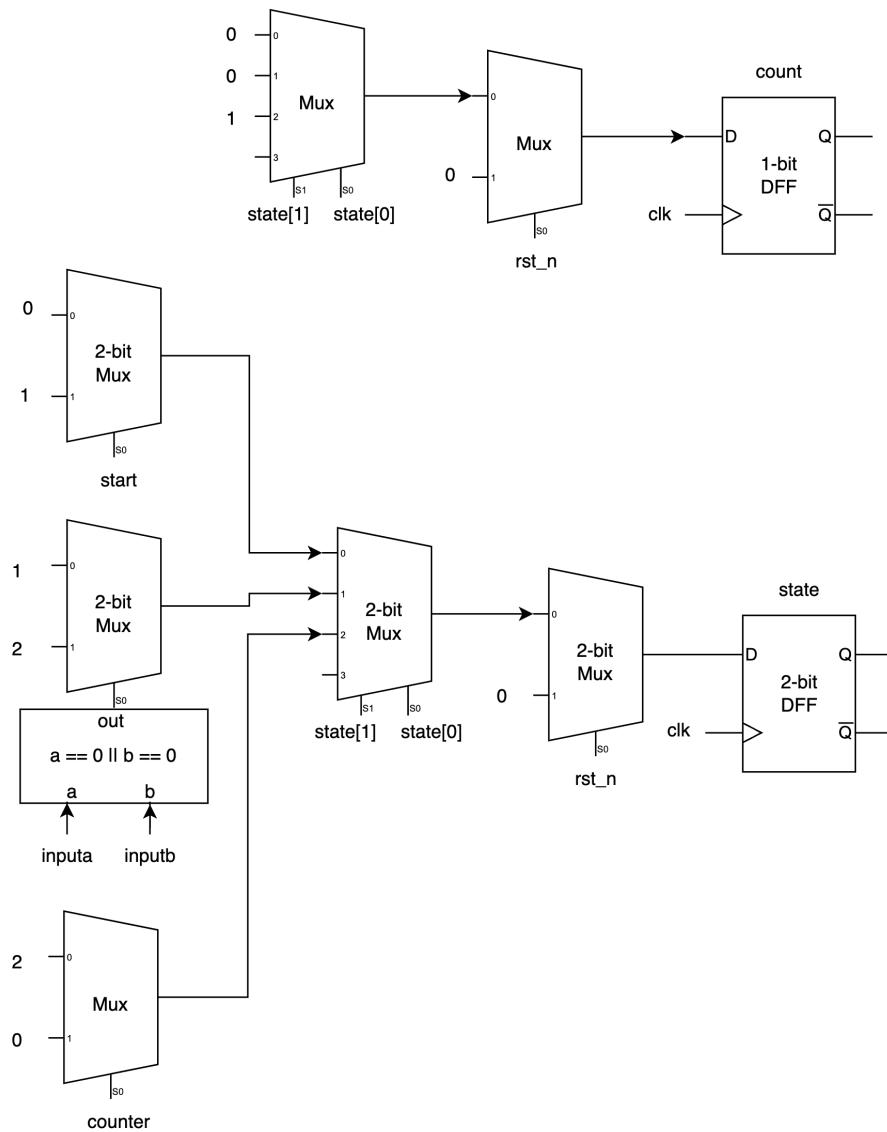
#### 3.1 Implementation

首先是狀態圖：



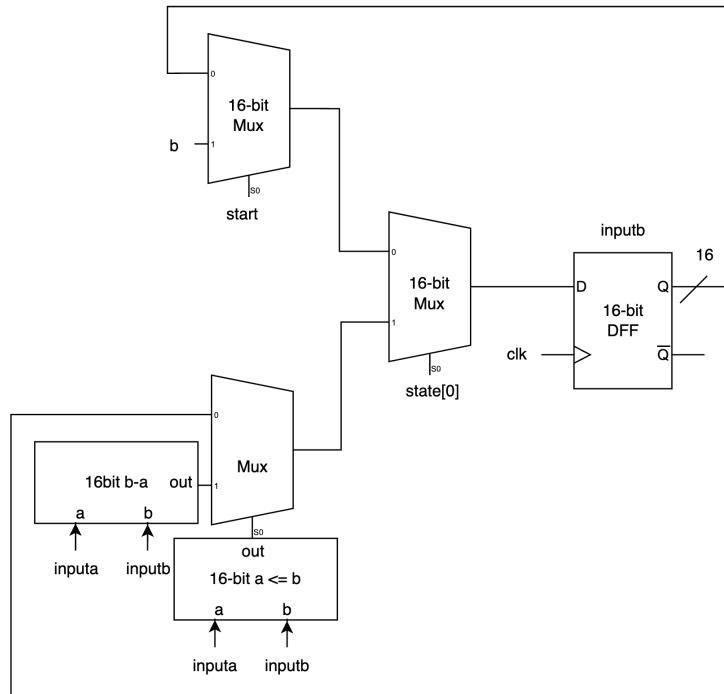
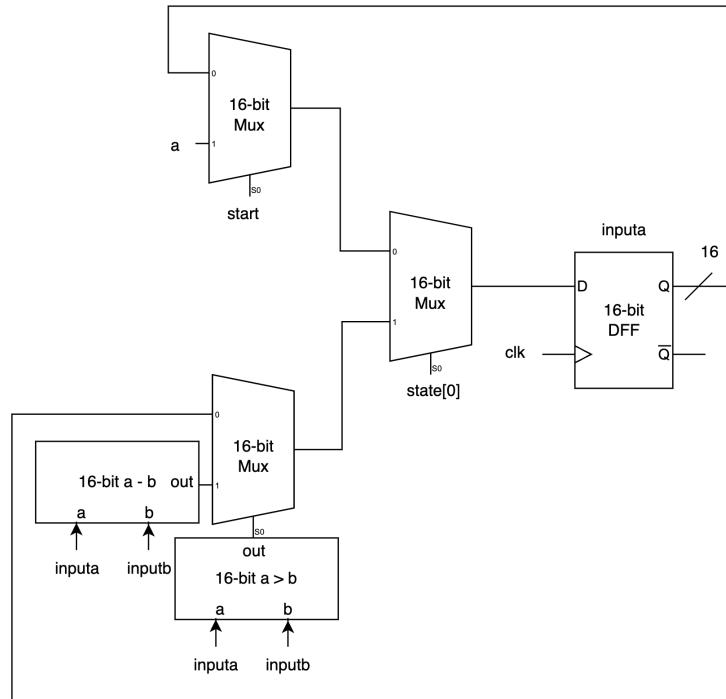
**Fig. 10** Q3 State diagram

接著是計數器與狀態的判定：



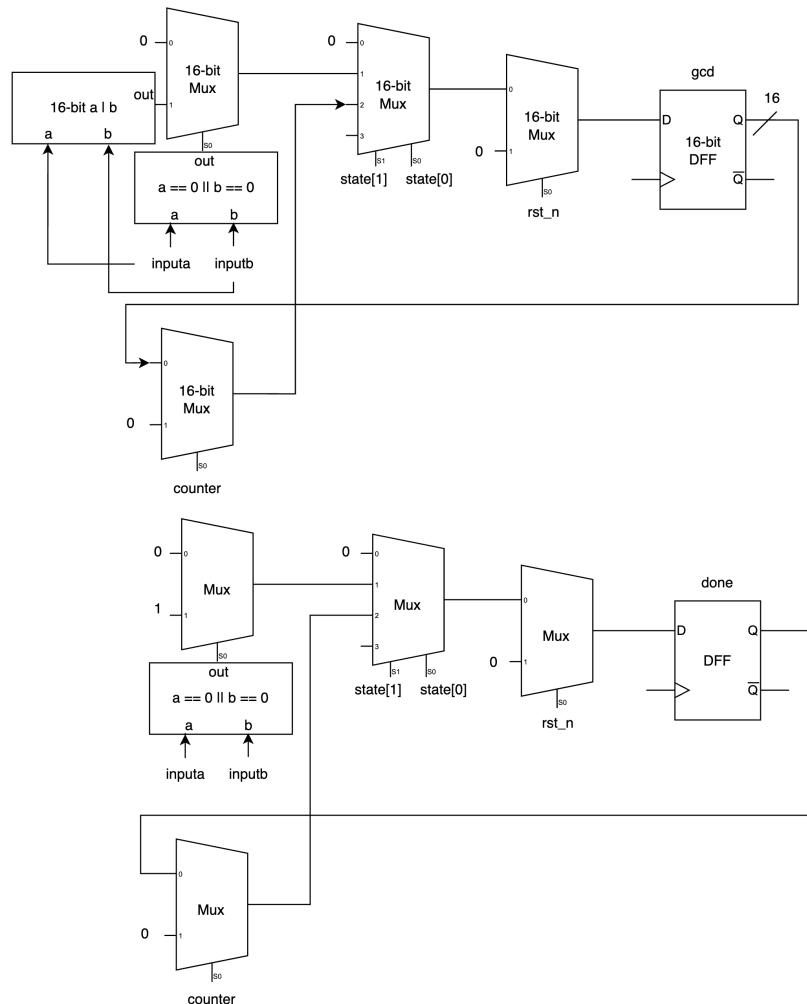
**Fig. 11** Q3 Counter Circuit

接著是輾轉相除法的實作：



**Fig. 12** Q3 Euclidean Circuit

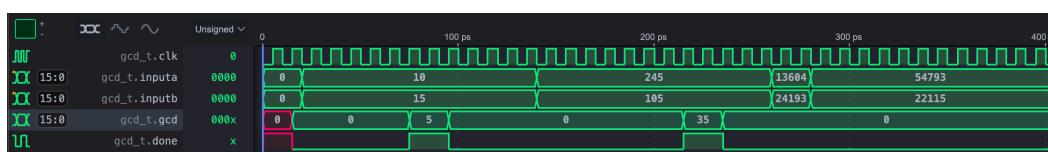
最後是 gcd 與 done 的輸出：



**Fig. 13** Q3 Output

### 3.2 Simulation

我們使用 random 來產生 a, b 的值，並觀察 gcd 的輸出是否正確。



**Fig. 14** Q3 Waveform

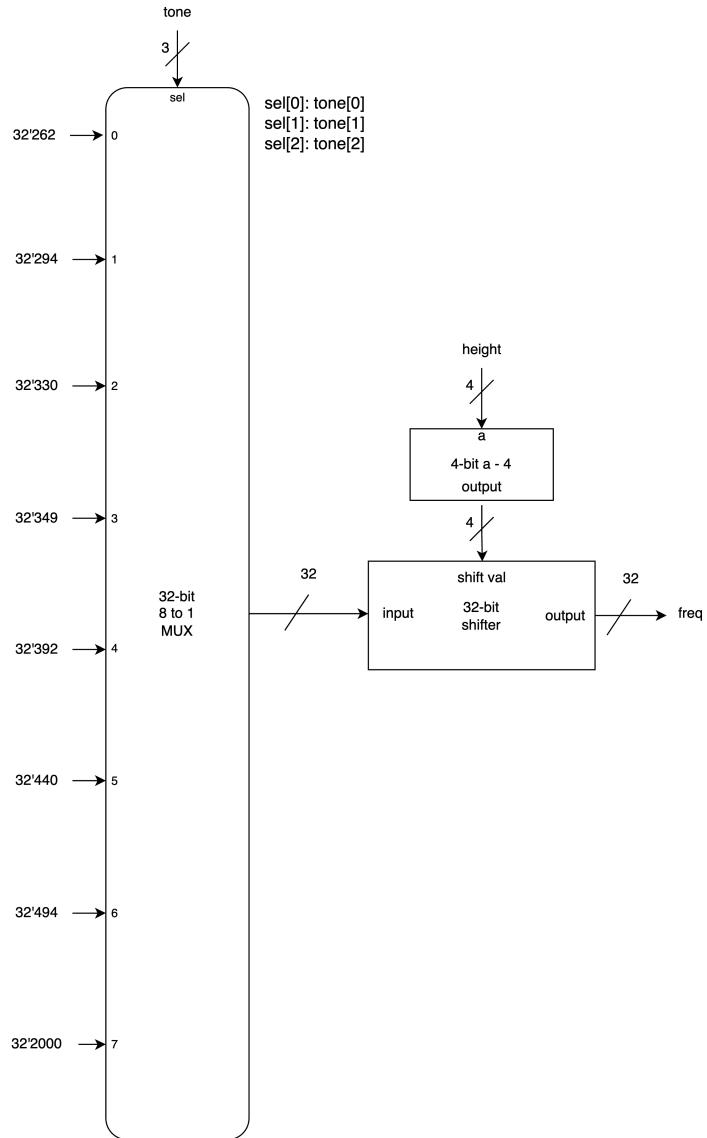
## 4 FPGA1: Mixed keyboard and audio modules together

這題要在 FPGA 上實作一個播放 C4~C8 音階的功能，當按下 w 鍵時，音調會往上播放、按下 s 鍵時，音調會往下播放、按下 r 鍵時，會在 1 秒與 0.5 秒之間互相切換播放速度。

### 4.1 Implementation

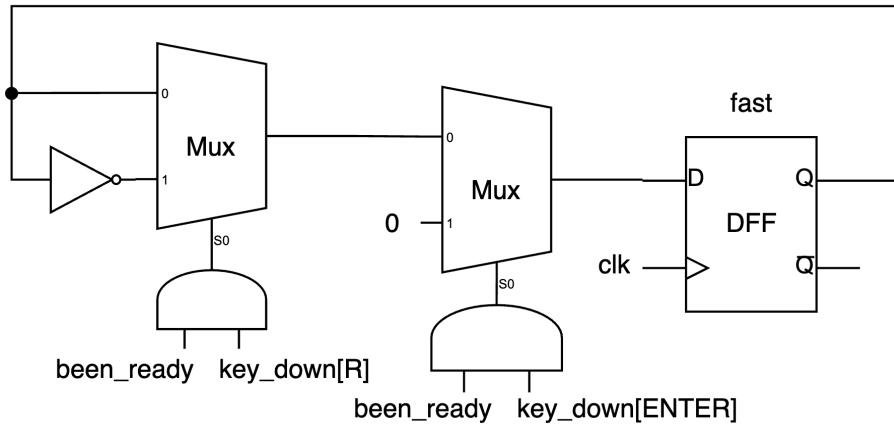
首先是 Decoder 的部分，這邊會接收兩個輸入：tone, height，分別代表音調和音高，如 C4 就會被分成 C, 4，並輸出相對應的音調頻率。

以 C 這個音為例，C4 的時候頻率就是 262 Hz，C5 的時候就是  $262 \times 2^1$ ，C6 則是  $262 \times 2^2$ ，以此類推，因此我直接使用 left shift 音高減掉四位元，就能夠得到相對應的頻率。



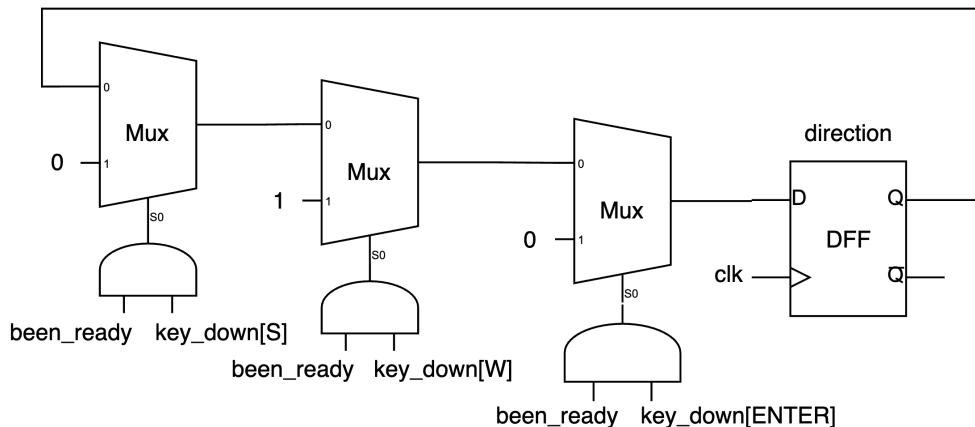
**Fig. 15** FPGA1 Decoder

接著是音階控制的部分，首先是速度， $\text{fast} = 0$  時代表每一秒更新， $\text{fast} = 1$  時代表每半秒更新。當按下鍵盤 R 時， $\text{fast}$  值就會做一次 not 反轉，下圖是這部分的電路圖：



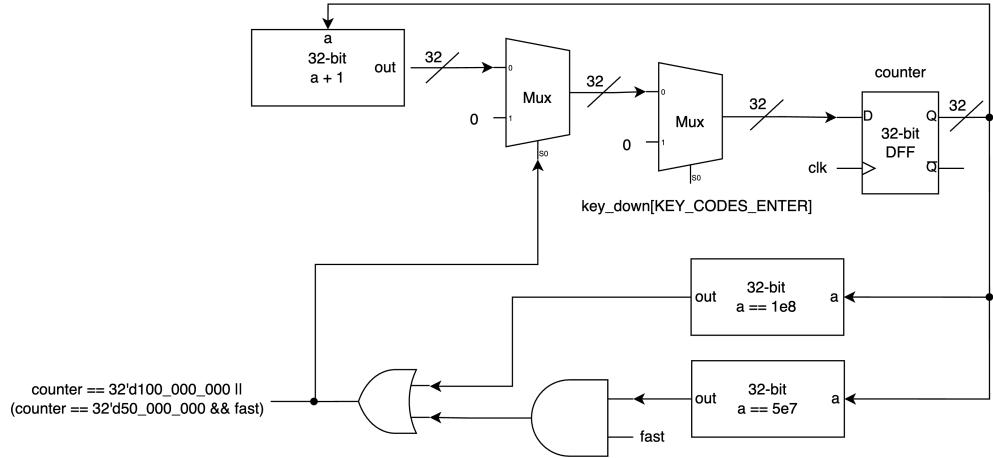
**Fig. 16** FPGA1 Fast

接著是 direction，與 fast 的實作方法類似， $\text{direction} = 0$  代表下行，反之則是上行。實作的部分是改成按下 S 時將 direction 設為 0，按下 W 時將 direction 設為 1。



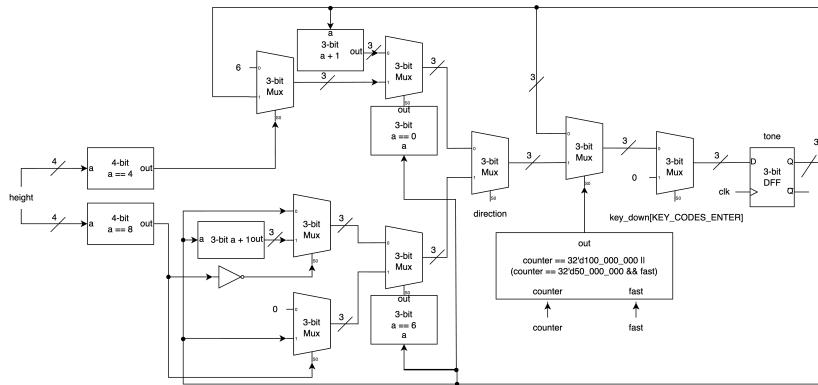
**Fig. 17** FPGA1 Direction

接下來是 counter，每一個 clock cycle 就會加一，並且根據 fast 的值，輸出一個訊號代表音調是否要更新。



**Fig. 18** FPGA1 Counter

有了以上三個 register 後，就可以來實作音調的控制了。首先是 tone 的部分，當音調不在 C4 或 C8 時，每次更新就會根據 direction，將 tone 在  $0 \sim 6$  的區間加上一或減去一。如果判定會超出範圍就會保持原樣。



**Fig. 19** FPGA1 Tone

接著是 height，每次更新時會檢測 tone 是否有 overflow 或是 underflow，並根據情況加減一。如果判定會超出範圍也會保持原諒。

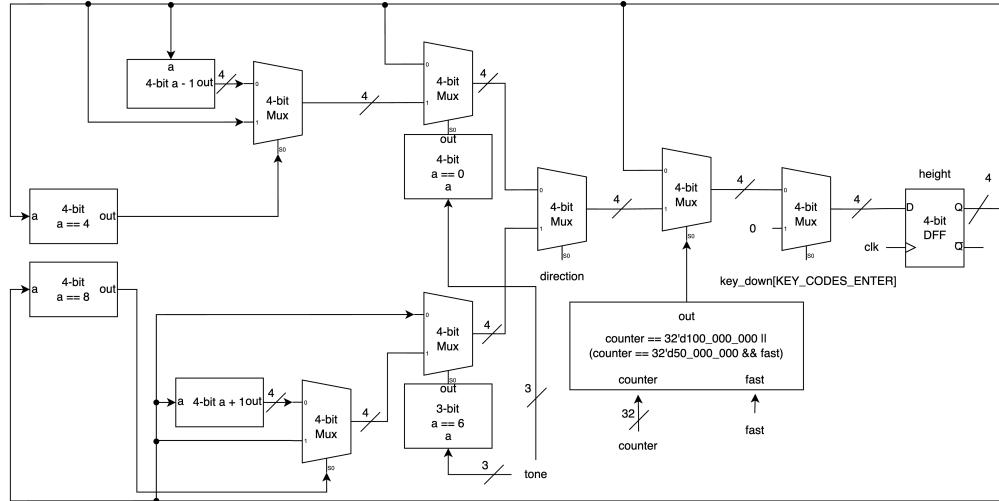


Fig. 20 FPGA1 Height

最後，只要將 tone, height 輸入到 Decoder 中，得到對應的頻率後輸入至音訊控制模組，就完成了這題的音階控制。下圖展現的是相關參數的連接方式：

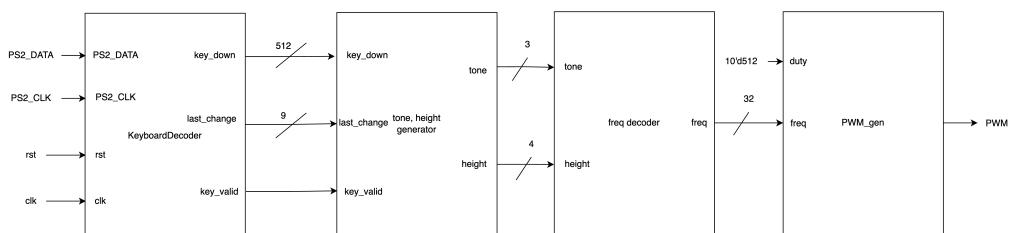


Fig. 21 FPGA1 Connect

## 5 FPGA2: vending machine

這題要在 FPGA 上實作一個自動販賣機，有三種商品：

- Coffee: 80 元
- Coke: 30 元
- Oolong: 25 元
- Tea: 20 元

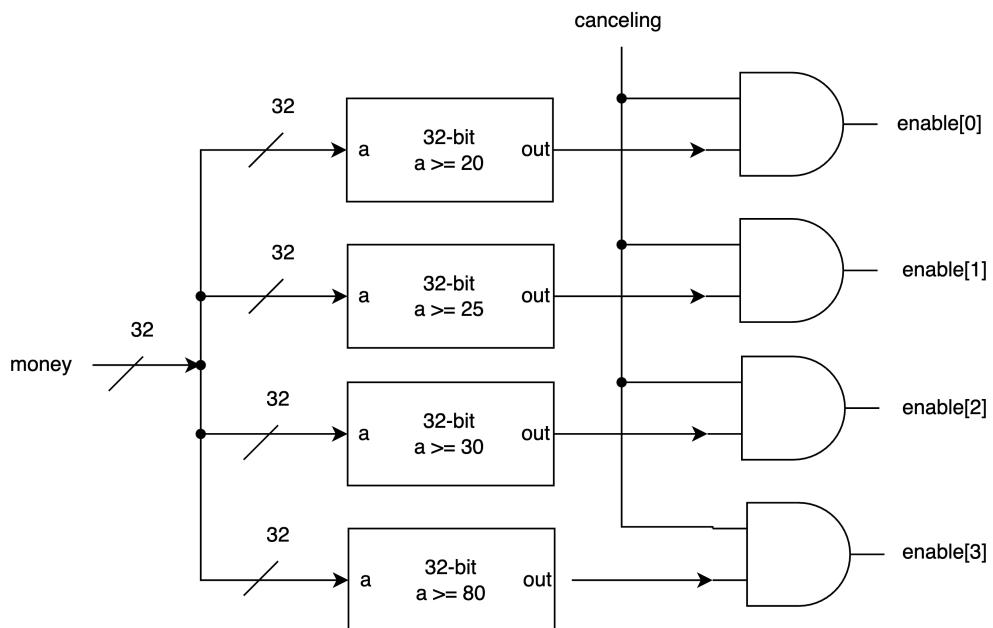
並且有這幾種面值的硬幣，分別對應到板子上的左中右按鈕：

- Left: 5 元
- Center: 10 元
- Right: 50 元

並且將板子上的 Top Button 作為 reset 訊號、將 Bottom Button 作為取消訊號。

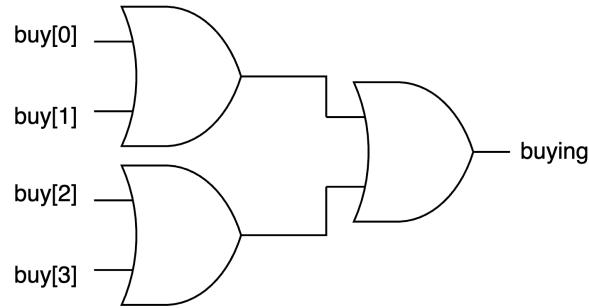
### 5.1 Implementation

enable[3:0]: 代表每個商品是否可以購買，透過判斷是否有足夠的錢，以及目前不在 cancel 狀態中來判定：



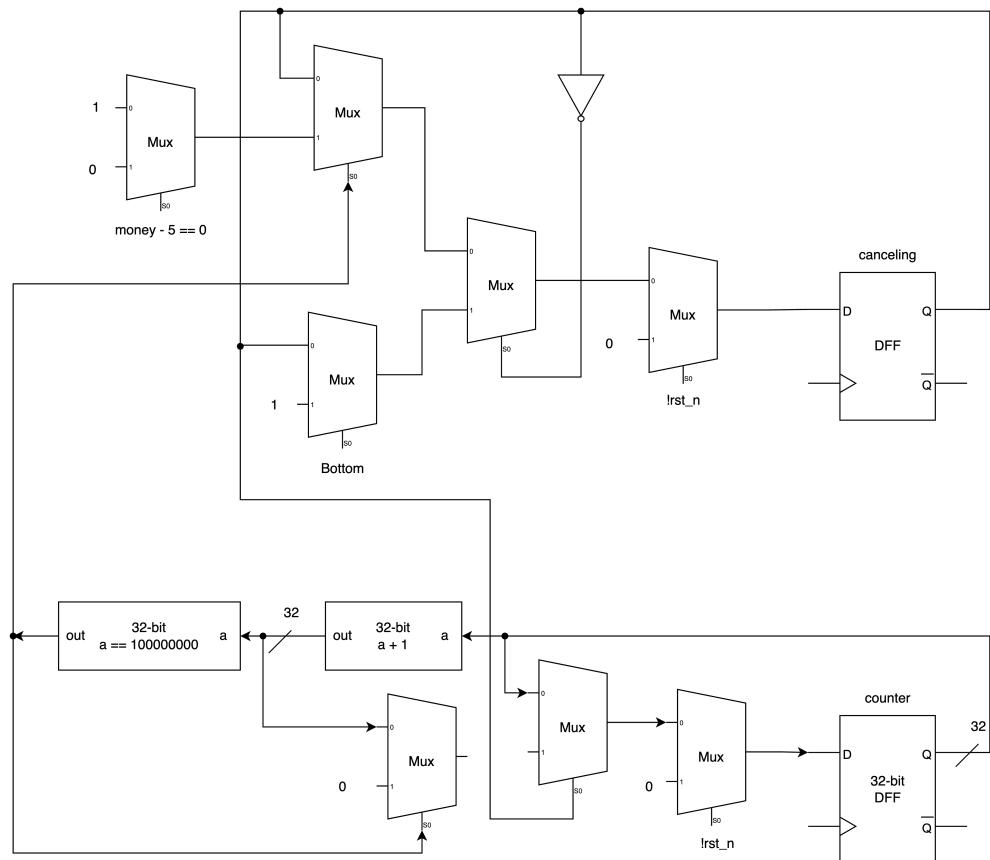
**Fig. 22** FPGA2 Enable signal

buying: 透過將四個 buy 訊號做 or 運算，來判定是否正在購買：



**Fig. 23** FPGA2 Buying signal

有關取消訊號的部分：



**Fig. 24** FPGA2 Cancel signal

將以上訊號接到 state machine 中：

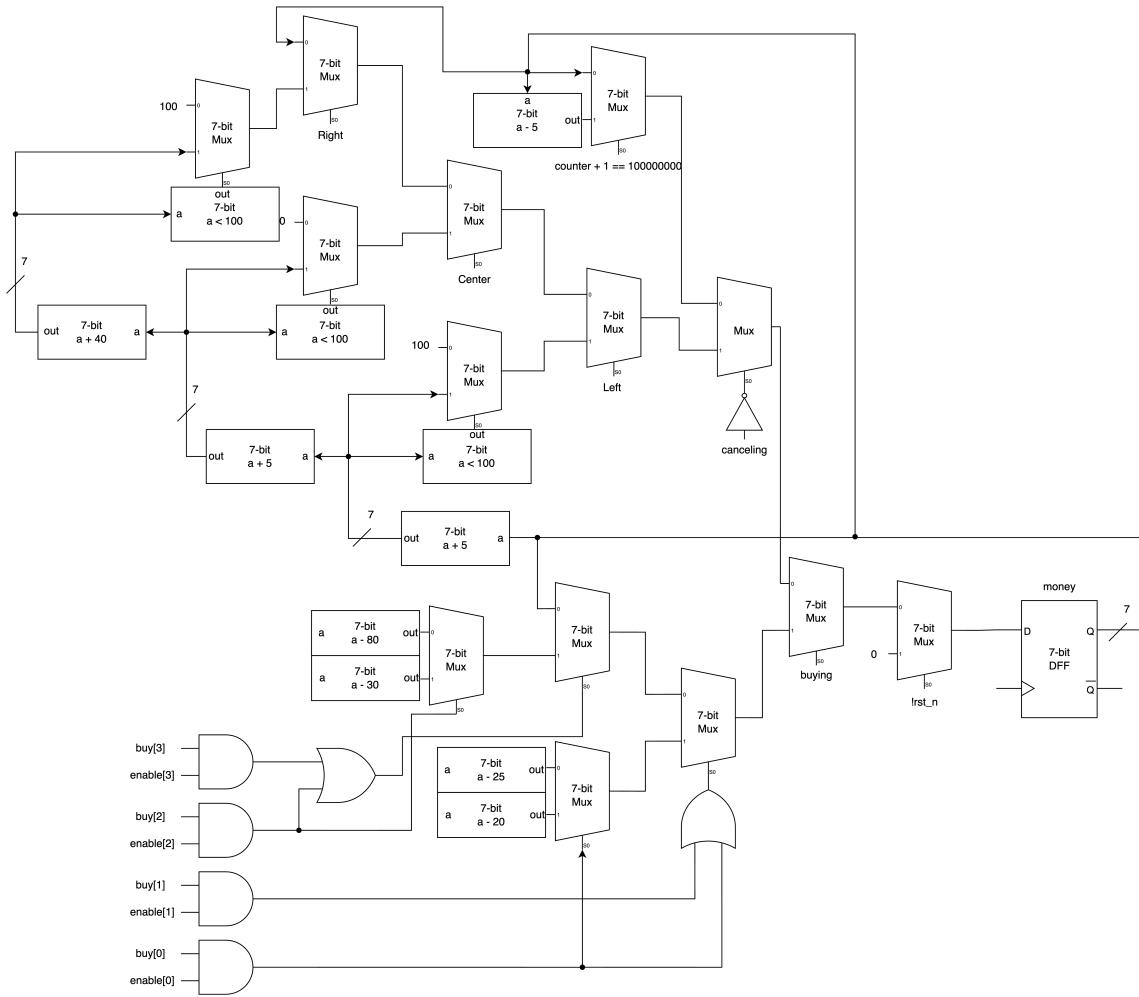
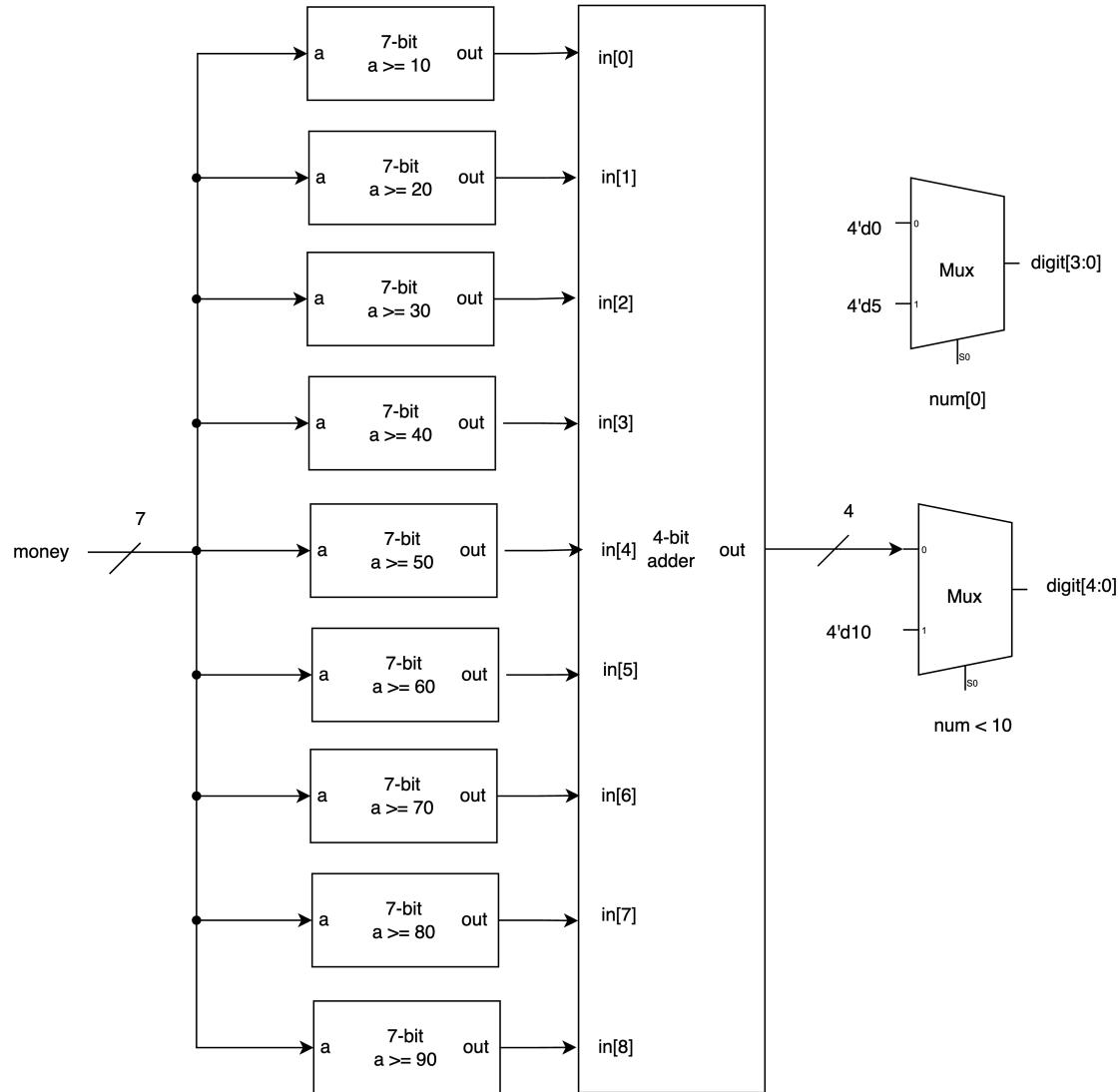


Fig. 25 FPGA2 State machine

接著，將 VendingMachine 的輸出接到 NumberDevide 中，計算出三個數字分別要顯示什麼：



**Fig. 26** FPGA2 Digit Devider

最後再將這些訊號接到七段顯示器的輸出控制模組中，就完成了這題的實作。

## 6 Other

### 6.1 What we have learned

- 這次實作了更多 Finite State Machine，透過多次實作讓我們學習到並建構出一個撰寫 FSM 的 Pattern。使得這次的實作過程比上次都還要順利不少。
- 在研究 Booth Multiplier 的過程中，我了解到，其實這個演算法就只是將我們平常加速乘法運算的手法，只是轉成二進位的形式而已。雖然網路上的程式碼沒有理解得很清楚，但因為已經了解概念了，所以還是可以自己寫出來。
- 了解到如何在 FPGA 上連接鍵盤，並稍微了解了一下 FPGA 上要怎麼操作 USB 的相關協定，這將對我們 Final Project 的實現方式有很大的幫助。
- 實際使用了一些小的音訊裝置組成一個蜂鳴器，比起直接接喇叭，這樣的方式讓我們更認識了喇叭裡面的工作原理，也發現到了，當可變電阻接電過長，有可能因為過熱而導致 FPGA 直接關機。

### 6.2 分工

- 陳克盈：Q3, Q4, FPGA1, FPGA2
- 蔡明妍：Q1, Q2