**Lab8\_1**

**Design Specification**

* For a speaker:

Input:

clk, rst\_n

Output:

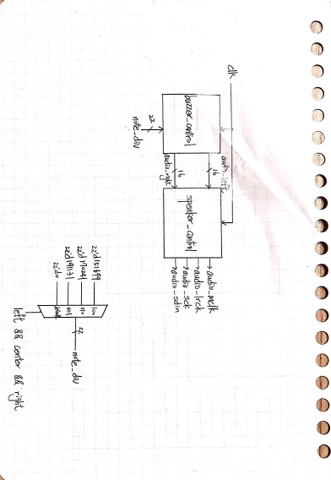
audio\_mclk,

audio\_lrck

audio\_sck,

audio\_sdin

* Draw the block diagram of the design.



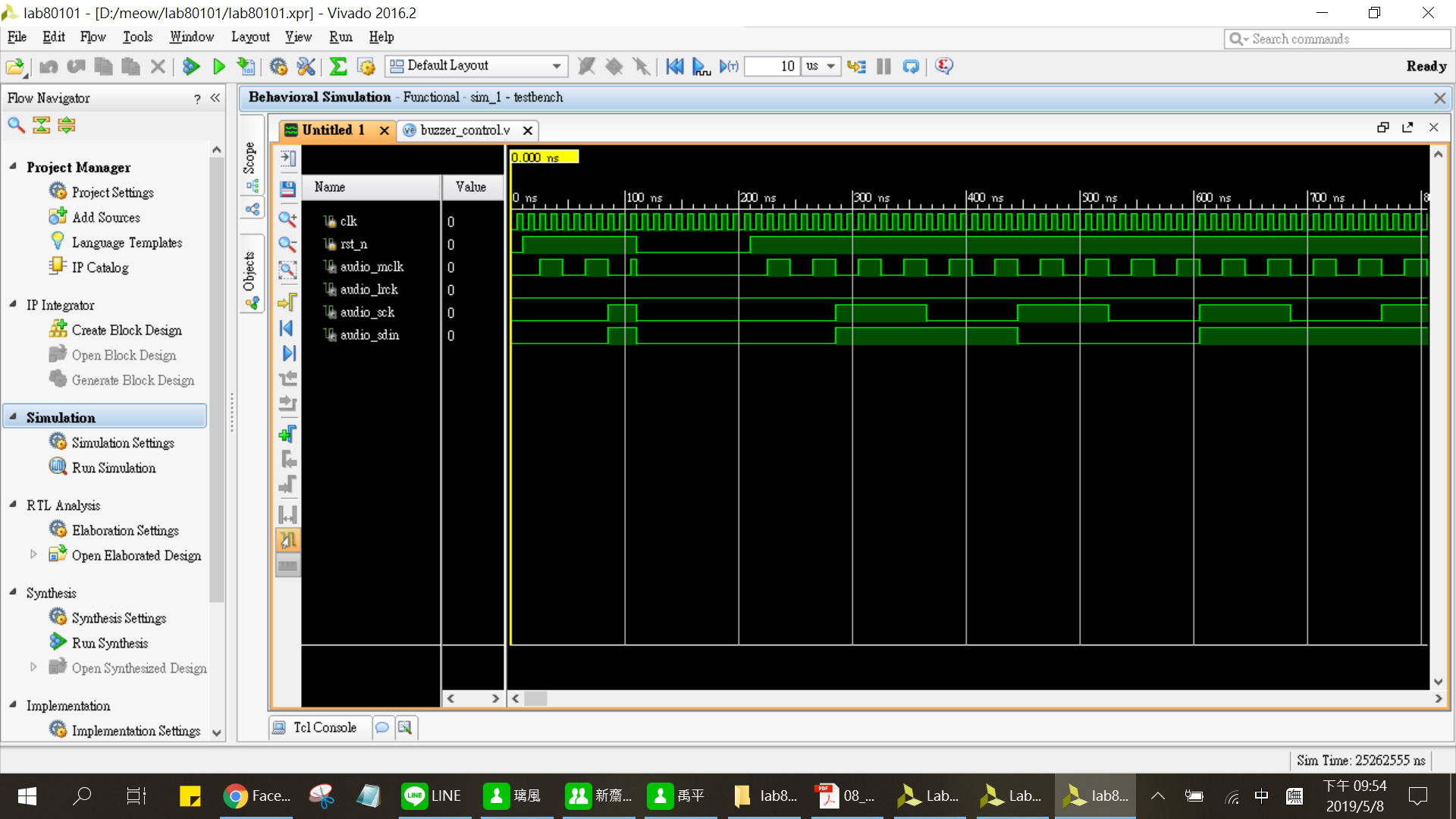
**Design Implementation**

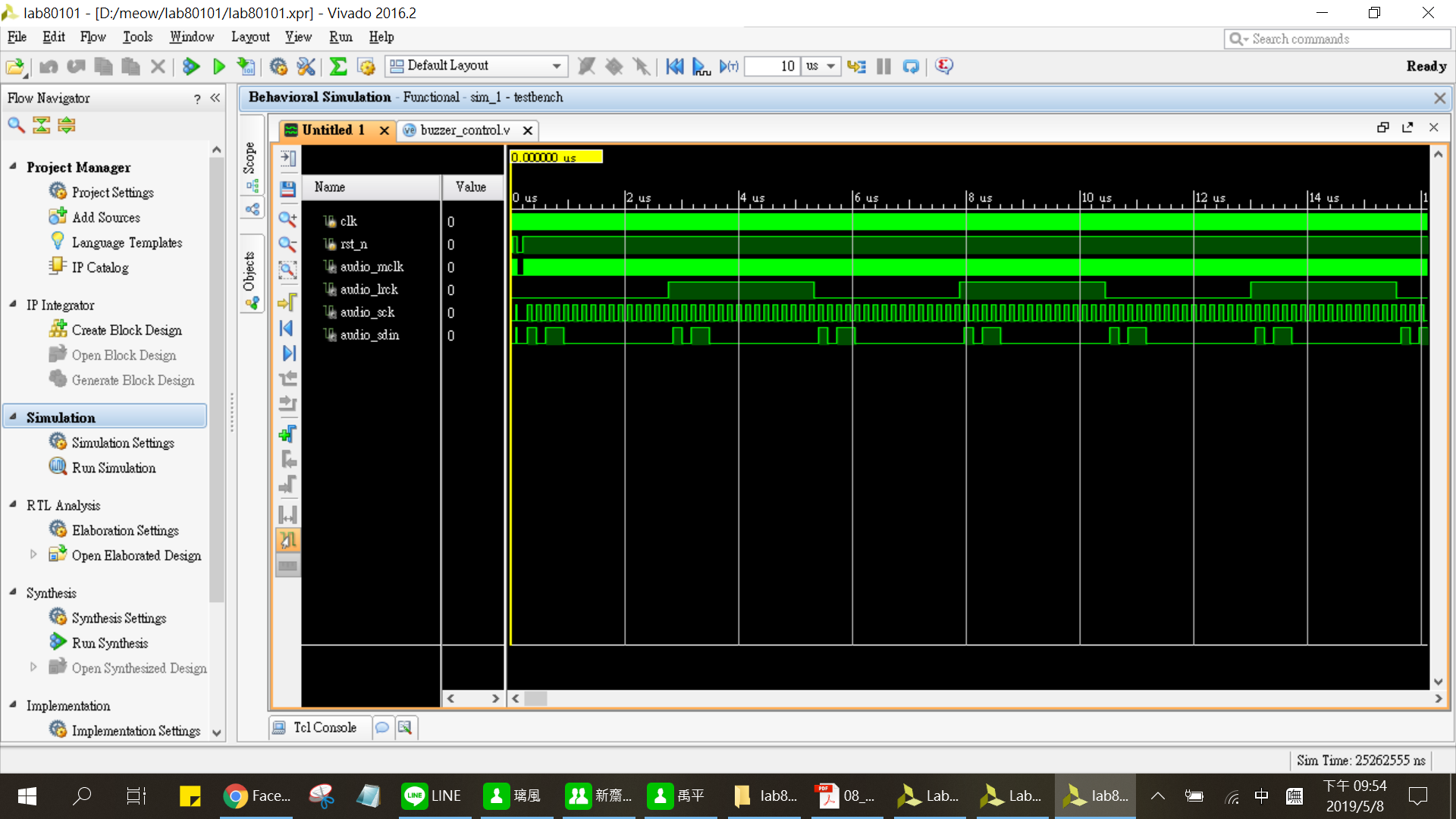
* 本題由buzzer\_control與speaker\_control組成。
* buzzer\_control

此module的功能為控制蜂鳴器的聲音頻率與振幅。由於第一題中的數值為給訂好的，因此其詳細功能將在第二題中說明。

* speaker\_control

此module的功能為將buzzer\_control輸出的各16bits的聲音頻率分為一次輸出1bit。其中的audio\_mclk, audio\_lrck, audio\_sck, 皆為用除頻器所得的結果；audio\_lrck是控制audio\_right與audio\_left的clock，audio\_sck則是控制audio\_sdin的。在這邊為了將audio\_right與audio\_left的值給audio\_sck，並一次輸出1 bit，我利用了flip flop來完成此部分。而由於audio\_right與audio\_left為各16 bits的數值，因此audio\_sck的頻率為audio\_lrck的32倍。





由圖可看出，audio\_mclk的頻率為clk的1/4倍；audio\_sck的頻率為audio\_lrck的32倍。

**Lab8\_2**

**Design Specification**

* For a speaker:

Input: clk,

rst\_n,

left,

center,

right,

up,

down

output:

audio\_mclk,

audio\_lrck,

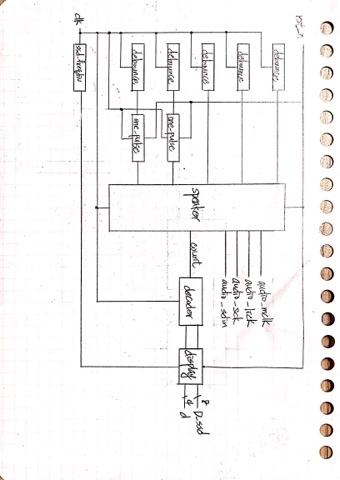
audio\_sck,

audio\_sdin,

[7:0]D\_ssd,

[3:0]d

* Draw the block diagram of the design.



**Design Implementation**

* 第二題為第一題的實作，在第一題中buzzer\_control的頻率與振幅是給定值，在第二題中的音頻則分別是do、re、mi，分別對應到左中右按紐；振幅(音量)則是由上下按鈕控制。在這裡我做了一個MUX來選擇輸出的頻率為哪個音。振幅則是將其做成16段的調節器，每當up = 1，down = 0時音量則加1，反之則減1。由於若只將音量的數值加一並沒有明顯區別，因此將其乘上1280(任一較大的變數)，才可以聽出其音量的改變。另外，為了讓它可以像鋼琴一樣，因此沒有在控制頻率的按鈕加one\_pulse。
* 音頻

|  |  |  |  |
| --- | --- | --- | --- |
| left | center | right | note\_div |
| 1 | 0 | 0 | 22'd151699 |
| 0 | 1 | 0 | 22'd170241 |
| 0 | 0 | 1 | 22'd191131 |
| default | | | 0 |

* I/O pin

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I/O | D\_ssd[7] | D\_ssd[6] | D\_ssd[5] | D\_ssd[4] | D\_ssd[3] | D\_ssd[2] | D\_ssd[1] | D\_ssd[0] |
| VOC | W7 | W6 | U8 | V8 | U5 | V5 | U7 | V7 |
| I/O | d[3] | d[2] | d[1] | d[0] | audio\_mclk | audio\_lrck | audio\_sck | audio\_sdin |
| VOC | W4 | V4 | U4 | U2 | A14 | A16 | B15 | B16 |
| I/O | rst\_n | clk | left | center | right | up | down |  |
| VOC | V17 | W5 | T17 | U18 | W19 | T18 | U17 |  |

**Discussion**

　　在speaker\_control中，我原本想用ring counter讓audio\_sdin可以一次輸出一個bit，但卻發現這樣會出問題，程式也會變得很冗長。所以我改用一般的flip flop，並設一個變數去控制輸出的值為audio\_left與audio\_right合起來的第幾個bit。

　　在控制do、re、mi時，原本的控制是當一個按鈕按下後便會發出聲音，沒有排除其他可能(例如兩個按鈕一起按)，所以會同時聽到兩個以上的音。為避免這種狀況，我把MUX改為只有一個按鈕按下時才會有聲音。

**Conclusion**

這次的實驗運用到了新的概念，一開始會有點不瞭解為什麼buzzer\_control的結構，但經由第二題自己改過參數後就會比較了解他的原理了。