Lab 3 report

組員：110062221李品萱

110062213唐翊雯

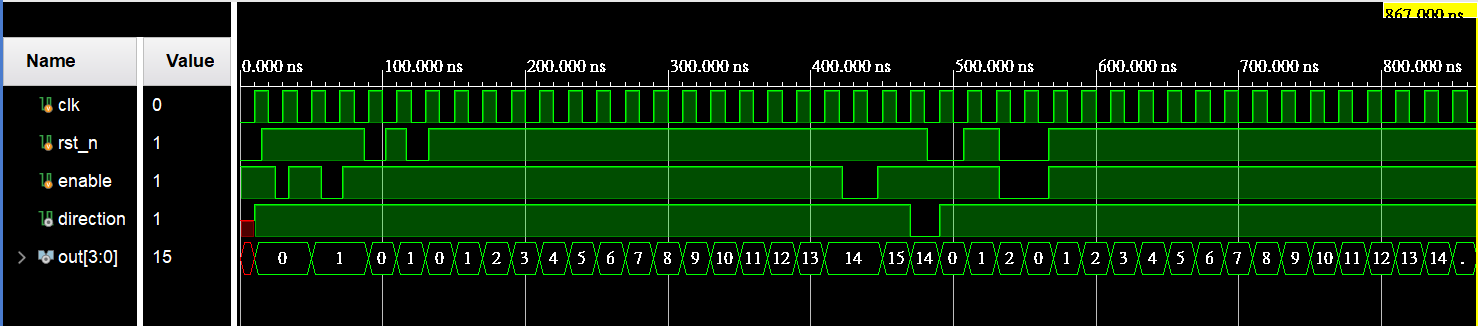
1. **4-bit Ping-Pong Counter**

一張含有 文字 的圖片

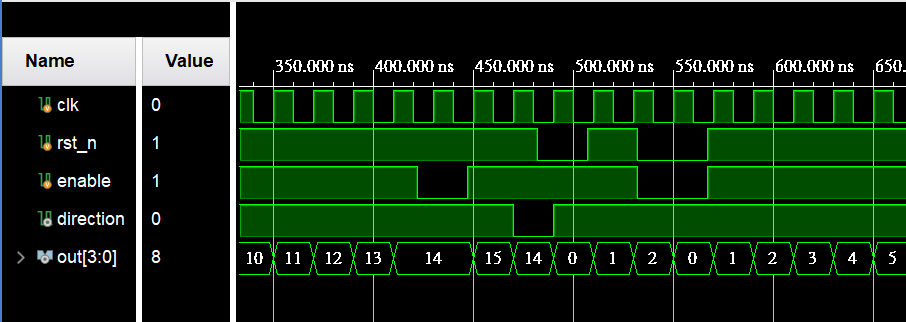
自動產生的描述如下圖，這題我們需要實作一個Ping-Pong Counter。

首先在combinational block的部份我們先用next\_out及next\_direction接下一次clock起來的時候要傳過Flip Flop的值。為了讓這個Counter counting up時能做到Spec中數到15後將direction從1變為0及counting down數到0後將direction從0變為1，我們觀察到只有Counting up的out在0到14的區間內或Counting down的out為0時，會使next\_direction = 1’b1，next\_out = out+1’b1，因此我們用if-else去做判斷，而它synthesize的結果即為Mux。Mux由左而右是priority由低至高，最左邊的Mux如上所述，而接下來這個Mux則會藉由enable判斷是否要hold原值或繼續傳next\_out及next\_direction；rst\_n則會判斷是否要將next\_dirextion設為1及next\_out設為0。在以上的combinational block做完之後，在每次clock起來時這個值就會通過Flip Flop，出來的值即為out與direction。簡單來說，combinational block會負責處理各個條件下next\_direction及next\_out的值，而sequential block則單純負責配合clock處理最後的out及direction。

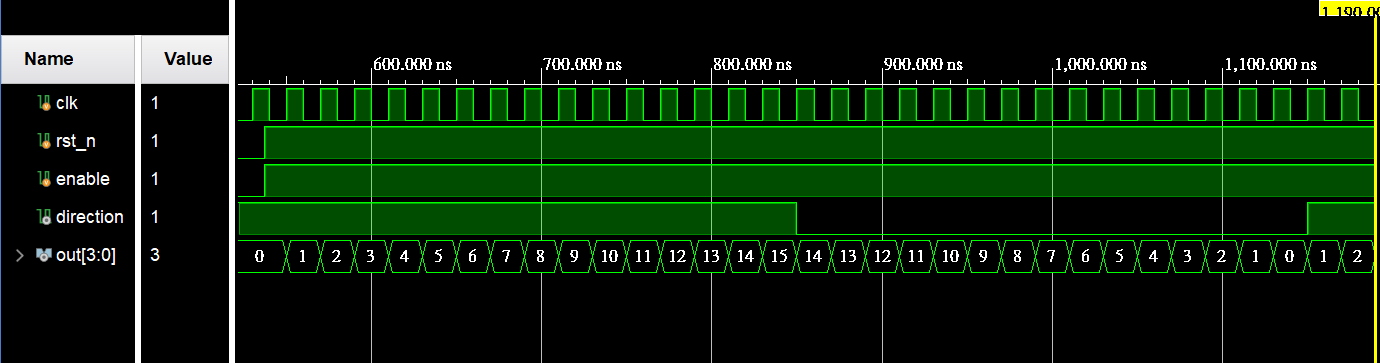
在下面的波型圖中我們可以看到，在clk起來時才會trigger到rst\_n為0，在此之前都是Unknow的狀態，並將我們的counter設定初始值，在out為1時會造成hold value的情形是因為我們的(rst\_n, enable) = (1, 0)的結果。



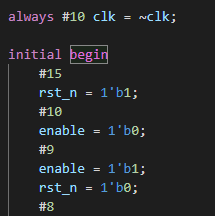
接下來我們在Counter運算過程中進行測試(rst\_n, enable)為(1, 0), (0, 1), (0, 0)的情況，如下圖。在out為14後經過一個clock， enable為0會被trigger到，此時如預期的會使counter hold value，而後由於已經數到15，direction會變為0並開始往下數；接下來我們讓rst\_n為0，clock起來時Counter也確實使out回到0，direction回到1；至於(0, 0)的情況則是由於rst\_n已經是0，且它的優先權是最高的，因此便會如waveform顯示的直接reset。



下圖我們測試從0數到15再數回0的狀況，確認我們設計的ping-pong counter可以正確地執行count up, count down及轉換direction的動作。



testbench的設計我們每10個單位時間（配合timescale即為10ns）會將clock反向，藉此產生clock的週期變化，其他input主要則是使用延遲一段時間去raise不同信號確認以上所述的各種情況，部分如下圖。



1. **First-In First Out (FIFO) Queue**
2. **Multi-Bank Memory**
3. **Round-Robin FIFO Arbiter**
4. **4-bit Parameterized Ping-Pong Counter**

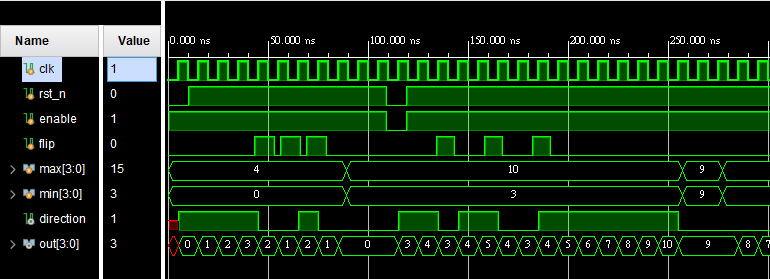
這一題與第一題非常相似，小差別在於多了flip、max、min等control signal，如下圖，block diagram的hold如下所示。

**一張含有 文字 的圖片

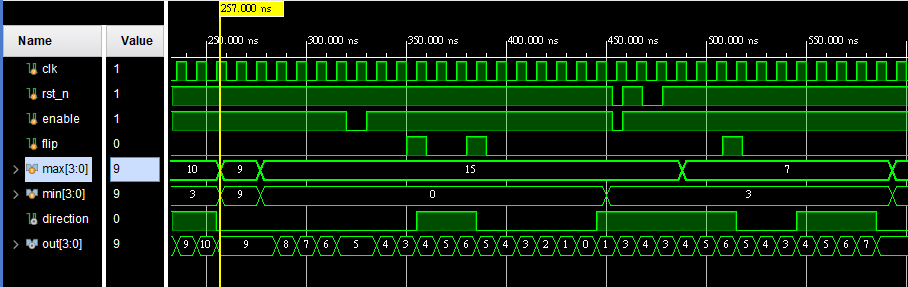
自動產生的描述**

與第一題相同的Mux這邊就不贅述，我們可以先看以flip為reset signal的Mux，flip為1時會將direction反向，out數的方向也會與原本相反；而hold我們將這題新增的條件直接以assign的方式接起來去判斷是否要維持當前的值。經過途中這些Mux之後我們就會得到最後的值，等clock起來便會將值送出。

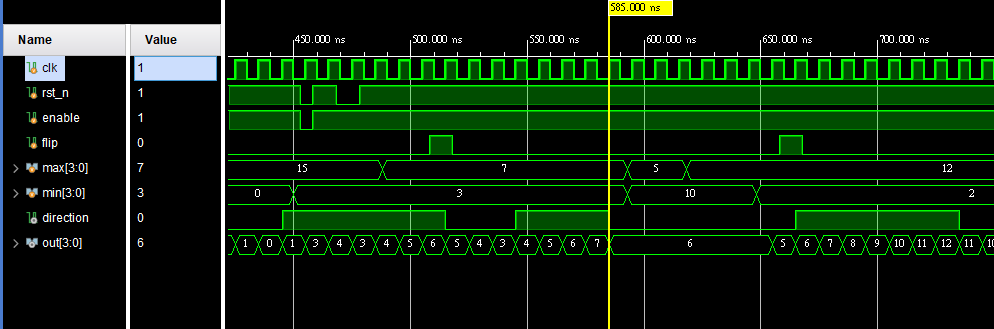
下面的波型圖中，我們先將(max, min)設為(4, 0)，做出與spec相同的waveform初步確認我們的設計應該是正確的，這邊我們也設計了連續flip的狀況，能看到direction及out都有做出相對的改變。接下來我們將(max, min)設為(3, 10)，剛trigger到新的值的時候，由於out<min，因此out會維持它原先的值一直到rst\_n為0時，direction為1，out被設為min，它才符合繼續count的條件。



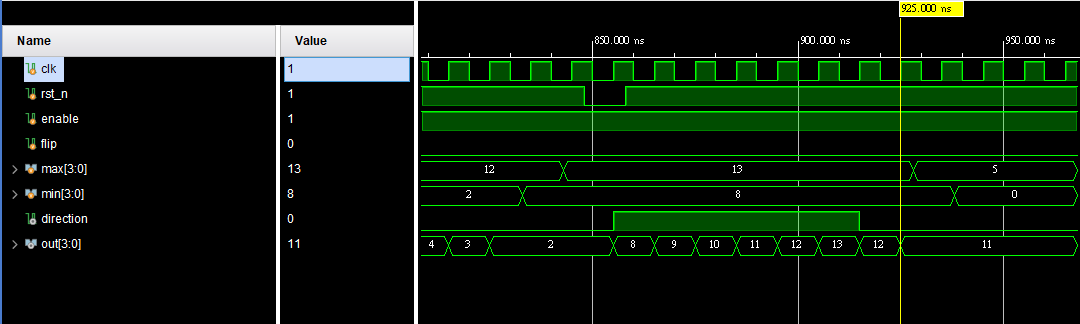
接下來我們測min==max==output的情況，counter此時會維持其原值(out and direction)，之後我們也繼續測flip及enable，皆符合預期。而由於前面我們都是同時改動max及min，這邊我們試了先改動min的case，也都有出現正常的結果。



接下來我們測max<min，此時counter會維持其當前值。



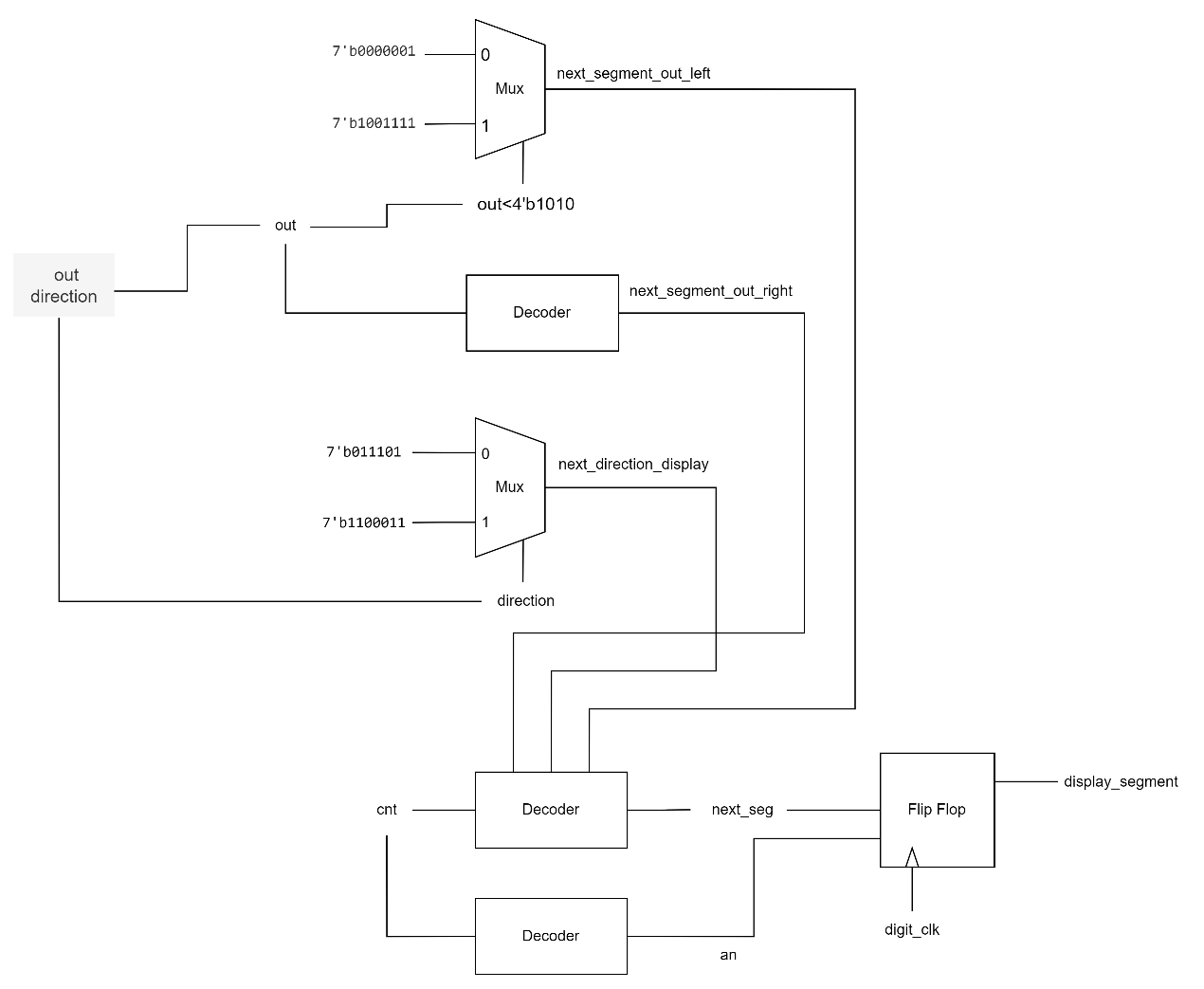
下圖中我們測out>max，此時counter亦會維持當前值。



這個testbench我們的目的是測試其是否符合spec的要求，因此除了檢查flip及enable是否能正常運作之外，我們主要對特殊狀況都進行測試，稍嫌冗長但至少是完整的。

1. **4-bit Parameterized Ping-Pong Counter on FPGA**

如下圖，這題我們將上一題實作的呈現於FPGA上，由於上一題已說明過counter的out及direction，這邊我們針對display的部分進行說明。首先，我們用Mux及Decoder為out對應到它要在seven segment顯示的值，Mux處理的是十位顯示的值及direction，Decoder則處理個位數的值。而後，如同上課提過的，seven segment一次只能顯示一個digit，因此我們用cnt決定它顯示的先後順序，而此處我們用的clock是除頻過的digit\_clock，在每個positive edge digit\_clock，我們的display\_segment就會收到next\_seg的值，即為它要顯示的值。



由於板子給的clock變化很快，為了方便肉眼觀察，我們需要為clock做除頻，我們的作法是將除頻過的clock與原本的enable做and再送進負責計算out及direction的module。除頻的部分上課時教授給的建議值是1/2^17clk，但我們在操作時發現如果要同時滿足觀察跟reset及flip bottom的話，1/2^24clk會較為剛好，如下圖。我們運用的是在basic lab做過的Clock Divider加以衍伸，用cnt數我們要在第幾個clock時將對應的divided\_clk做更動。

一張含有 文字, 螢幕, 螢幕擷取畫面, 銀色 的圖片

自動產生的描述

另外，我們還需要對bottom做debounce及one pulse，這邊上課時都有講解過，所以可以直接實做出來。One pulse Circuit首先會將debounce過的結果做延遲一個clock及反向，而後將其與debounce的結果and起來送入DFF，即會得到結果。這麼做的原因在於bottom通常可能被按著很長時間，但我們只想要按下去時產生一個clk的變化。在design時需要注意的是， onepulse module用的clock必須是除頻過後的clock，因為我們這邊onepulse出來的結果必須跟控制display的clock一樣寬，否則可能會trigger不到這個信號，這也是為什麼我們需要配合bottom去調整除頻的數值。Debounce的部分則是為了去除push bottom產生的glitches，因此我們用四個DFF過濾掉它，即可完成debounce。

這題我們沿用上一題的module並稍作修改，如：在push reset bottom時它會給出1，但我們要的是reset為0時reset，因此我們先做not再送進上一題的module。對於這題我們分不同module處理debounce、one pulse、clock divider及FPGA display，在display的部分則寫好7’b對應亮哪個Pin再將其接好，即可display 4-bit Parameterized Ping-Pong Counter。

1. **Summary**
2. **Contributions**

* **Code:**
* **Report:**