2025 Logic System Lab

邏輯系統實驗

Lab5 Report

|  |  |  |
| --- | --- | --- |
|  | 姓名 | 學號 |
| 組員1 | 彭淳漵 | E24131013 |
| 組員2 | 劉宸暟 | E24136487 |

組別 : 8

**[ Summary ]**

|  |  |  |
| --- | --- | --- |
|  | Finished / Unfinished | YouTube Link |
| P1 | Finished |  |
| P2 | Finished |  |
| P3 | Finished |  |
| P4 | Finished |  |

Please read “Lab5 submission” carefully, and answer the following questions based on the requirements specified for each Practice.

**[ Practice 1 ]**

**Q1: Please explain your overall design concept**

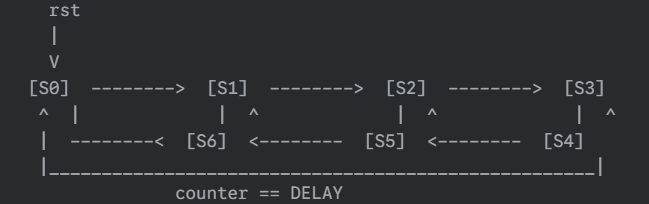
1. How to Determine 0.25 Seconds Have Passed

在這個設計中，0.25 秒的時間間隔是透過一個計數器 (counter) 來實現的。

* 程式碼中定義了一個常數 CYCLE\_025S，其值為 31,250,000。這個數值代表在 125MHz 的時脈頻率下，0.25 秒內時脈訊號的總週期數。
* always 區塊在每個週期開始時執行。
* 在 always 區塊中，如果計數器的值等於 DELAY（ CYCLE\_025S），則表示 0.25 秒已經過去，此時計數器會被重置為 0，並且狀態機 (state) 和 LED 選擇訊號 (led\_sel) 會根據當前狀態進行更新。
* 如果計數器的值還沒達到 DELAY，計數器就會遞增 1。

2. Finite State Machine (FSM)

* Diagram:



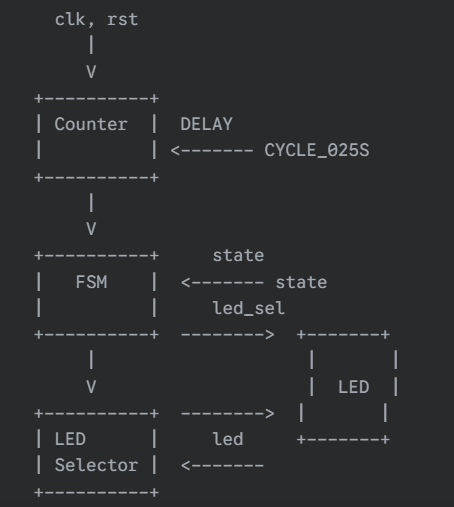
* State Description:
  + S0: 初始狀態，LED 1 亮起。
  + S1: LED 2 亮起。
  + S2: LED 3 亮起。
  + S3: LED 4 亮起。
  + S4: LED 3 亮起 (回程)。
  + S5: LED 2 亮起 (回程)。
  + S6: LED 1 亮起 (回程)。
* Transition Conditions:

所有狀態之間的轉換條件都是當計數器 counter 的值等於 DELAY 時發生。重置訊號 rst 為高電位時，狀態機回到 S0 狀態。

* Output Behavior:
* S0: led\_sel = 4'b0001 (LED 1)
* S1: led\_sel = 4'b0010 (LED 2)
* S2: led\_sel = 4'b0100 (LED 3)
* S3: led\_sel = 4'b1000 (LED 4)
* S4: led\_sel = 4'b0100 (LED 3)
* S5: led\_sel = 4'b0010 (LED 2)
* S6: led\_sel = 4'b0001 (LED 1)

3. Overall Architecture Diagram

* System Architecture Diagram:



* Component Functions:
  + Counter: 一個 26 位元的計數器，用於計算時脈週期，以判斷 0.25 秒是否經過。
  + FSM (Finite State Machine): 有限狀態機，控制 LED 的顯示順序。
  + LED Selector: 根據 FSM 的輸出 (led\_sel)，選擇要點亮的 LED。
  + LED: 4 位元的 LED 輸出。
  + clk: 時脈訊號，驅動整個設計。
  + rst: 重置訊號，用於將系統初始化到已知狀態。
* Code Mapping:
  + Counter: 對應到程式碼中的 counter 暫存器。
  + FSM: 對應到程式碼中的 state 暫存器和 always @(posedge clk or posedge rst) 區塊中的 case 敘述。
  + LED Selector: 對應到程式碼中的 led\_sel 暫存器。
  + LED: 對應到程式碼中的 led 輸出埠。
  + clk: 對應到程式碼中的 clk 輸入埠。
  + rst: 對應到程式碼中的 rst 輸入埠。

**Q2: Screenshot the waveform and explain how to determine if it is functioning correctly**

1. Please explain what changes you made to the design in order to facilitate waveform viewing during pre-simulation.

因為實際的0.25秒需要花費太多模擬時間，故運行模擬時只以99個週期模擬0.25秒。

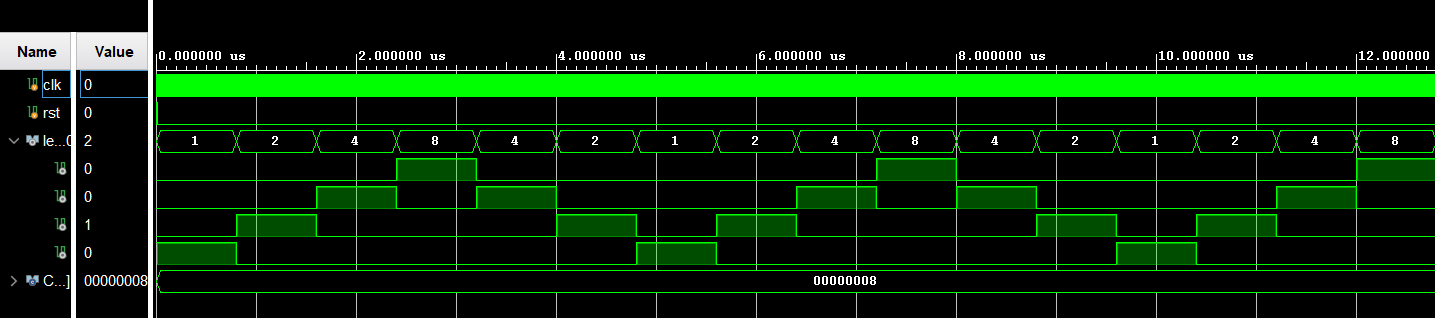
Testbench 中包含了以下關鍵部分，這些部分用於波形檢視：

* 訊號宣告：Testbench 宣告了需要連接到被測試模組 (Marquee) 的訊號 (clk, rst, led)，以及內部用於監測的線 (wire) 。
* 模組實例化：它實例化了 Marquee 模組，並將宣告的訊號連接到模組的埠。
* 時脈生成：initial 和 forever 區塊用於生成時脈訊號 clk。
* 重置生成：initial 區塊產生重置訊號 rst。
* 監測訊號：initial 區塊中的 $monitor 用於在模擬過程中印出訊號的值，這對於除錯很有幫助。更重要的是，$dumpfile 和 $dumpvars 用於生成 VCD 檔案，這些檔案可以被波形檢視器用來顯示訊號隨時間變化的情況。

2. Please write the testbench yourself and run pre-simulation

* testbench內容已附加於檔案中

3. Add the signals you think are necessary to the waveform, then screenshot the waveform and explain how to determine if it is functioning correctly

* 為了驗證設計的功能，我們需要在波形檢視器中觀察以下訊號：
  + clk：時脈訊號，用於確認時脈是否正常工作。
  + rst：重置訊號，用於確認重置是否有效。
  + state：狀態機的當前狀態，用於追蹤狀態轉換。
  + counter：計數器的值，用於驗證 0.25 秒的時間間隔。
  + led\_sel：LED 選擇訊號，用於確認哪個 LED 應該被點亮。
  + led：LED 的輸出值，用於檢視最終的 LED 顯示模式。
* 波形截圖與功能驗證:

**[ Practice 2 ]**

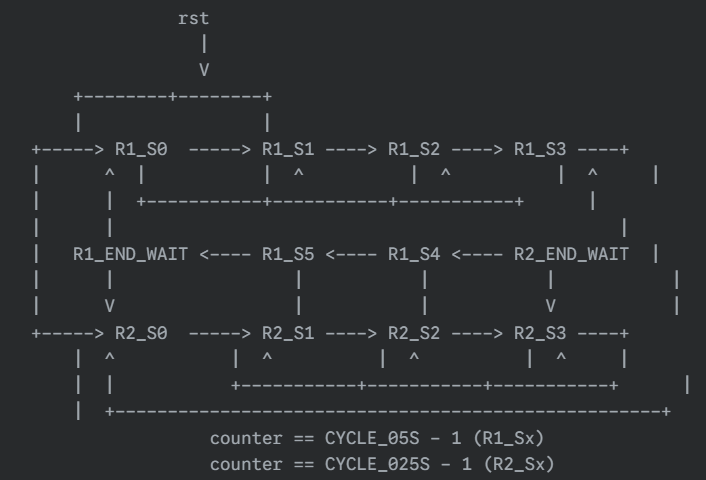
**Q1: Please explain your overall design concept**

1. How to Switch Between Two Speeds?

這個設計使用 round\_type 暫存器來控制兩個速度之間的切換，並使用兩個不同的預設計數值 (CYCLE\_05S 和 CYCLE\_025S) 來實現速度切換 。

* round\_type 是一個 1 位元的暫存器，用於指示當前處於哪個速度模式。當 round\_type 為 0 時，系統處於第一輪模式（較慢速度），當 round\_type 為 1 時，系統處於第二輪模式（較快速度） 。
* CYCLE\_05S 和 CYCLE\_025S 這兩個 define 分別定義了較慢速度和較快速度的計數器最大值。CYCLE\_05S 的值是 62,500,000，CYCLE\_025S 的值是 31,250,000 。由於 CYCLE\_05S 的值是 CYCLE\_025S 的兩倍，因此第一輪的速度是第二輪速度的一半。
* 在 always @(posedge clk or posedge rst) 區塊中，case 敘述根據 current\_state 的值來決定何時更新狀態和 LED 輸出。在每個狀態中，都有一個 if 條件判斷計數器 counter 的值是否達到預設的最大值 (CYCLE\_05S - 1 或 CYCLE\_025S - 1)。如果達到，計數器重置，狀態更新，LED 輸出也更新。
* 當狀態機從第一輪的 R1\_END\_WAIT 轉換到第二輪的 R2\_S0 時，round\_type 從 0 變為 1，從而切換到較快速度 。當狀態機從第二輪的 R2\_END\_WAIT 轉換回第一輪的 R1\_S0 時，round\_type 從 1 變回 0，從而切換回較慢速度。

2. Finite State Machine (FSM)

* FSM Diagram:
* State Description:

第一輪狀態 (R1\_\*): 這些狀態控制 LED 以較慢的速度循環點亮。

* + R1\_S0: 第一輪的起始狀態，點亮 LED[0]
  + R1\_S1: 點亮 LED[1]
  + R1\_S2: 點亮 LED[2]
  + R1\_S3: 點亮 LED[3]
  + R1\_S4: 點亮 LED[2]（回程）
  + R1\_S5: 點亮 LED[1]（回程）
  + R1\_END\_WAIT: 第一輪的結束等待狀態，準備切換到第二輪

第二輪狀態 (R2\_\*): 這些狀態控制 LED 以較快的速度循環點亮。

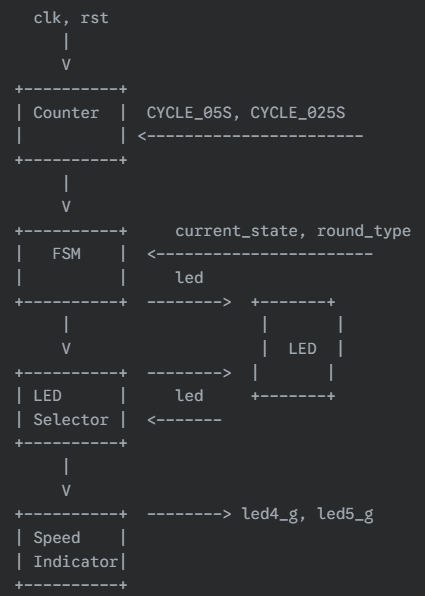
* + R2\_S0: 第二輪的起始狀態，點亮 LED[0]
  + R2\_S1: 點亮 LED[1]
  + R2\_S2: 點亮 LED[2]
  + R2\_S3: 點亮 LED[3]
  + R2\_S4: 點亮 LED[2]（回程）
  + R2\_S5: 點亮 LED[1]（回程）
  + R2\_END\_WAIT: 第二輪的結束等待狀態，並準備切換回第一輪
* Transition Conditions:

所有狀態之間的轉換都是由計數器達到一個預設值觸發的。在第一輪狀態中，這個值是 CYCLE\_05S - 1，在第二輪狀態中，這個值是 CYCLE\_025S – 1。rst 為高電位時，狀態機回到 R1\_S0 狀態。

* Output Behavior:
  + 每個狀態都會控制 led 輸出，以點亮特定的 LED。
  + round\_type 在 R1\_END\_WAIT 狀態轉換到 R2\_S0 時從 0 變為 1，在 R2\_END\_WAIT 狀態轉換回 R1\_S0 時從 1 變為 0。
  + led4\_g和led5\_g的值取決於round\_type。當round\_type為0時，led4\_g為1，led5\_g為0。為1時，led4\_g為0，led5\_g為1。

3. Overall Architecture Diagram

* System Architecture Diagram:



* Component Functions:
  + Counter: 29 位元的計數器，用於計數時脈週期，以判斷何時進行狀態轉換。
  + FSM: 有限狀態機，控制LED的顯示順序和速度切換。
  + LED Selector: 根據 FSM 的輸出 (led)，選擇要點亮的 LED。
  + LED: 4 位元的 LED 輸出。
  + Speed Indicator: 根據 round\_type 的值，控制 led4\_g 和 led5\_g 的輸出，指示當前速度模式。
  + clk: 時脈訊號，驅動整個設計。
  + rst: 重置訊號，用於將系統初始化到已知狀態。
* Code Mapping:
  + Counter: 對應到程式碼中的 counter 暫存器。
  + FSM: 對應到程式碼中的 current\_state 暫存器和 always @(posedge clk or posedge rst) 區塊中的 case 敘述。
  + LED Selector: 對應到程式碼中 led 輸出埠的控制邏輯。
  + LED: 對應到程式碼中的 led 輸出埠。
  + Speed Indicator: 對應到 always @(posedge clk or posedge rst) 區塊中控制 led4\_g 和 led5\_g 的 if-else 敘述。
  + clk: 對應到程式碼中的 clk 輸入埠。
  + rst: 對應到程式碼中的 rst 輸入埠。

**Q2: Screenshot the waveform and explain how to determine if it is functioning correctly**

1. Please explain what changes you made to the design in order to facilitate waveform viewing during pre-simulation.

因為實際的0.5秒 (0.25秒) 需要花費太多模擬時間，故運行時只以99 (49) 個週期模擬0.5秒(0.25秒)。

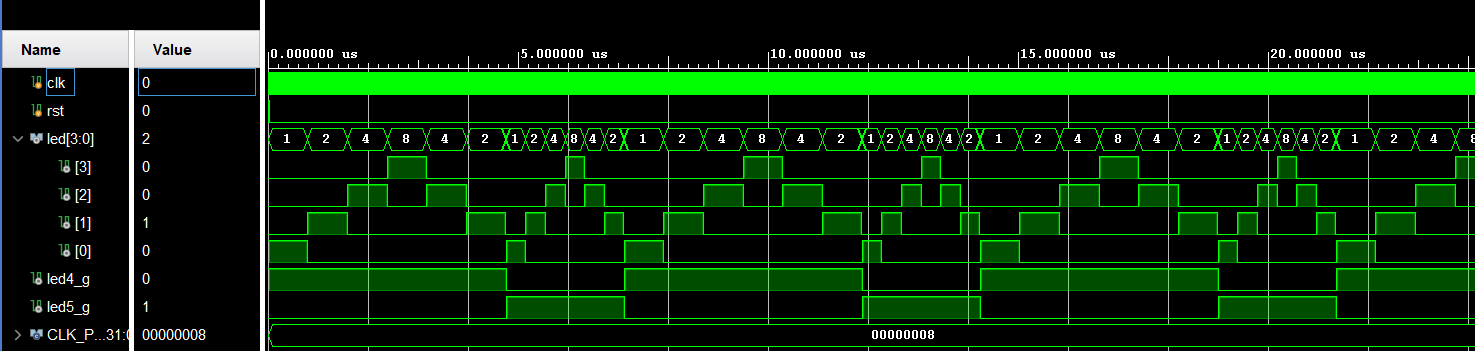
Testbench 中包含了以下關鍵部分：

* 訊號宣告：Testbench 宣告了需要連接到被測試模組 (Marquee) 的訊號 (clk, rst, led, led4\_g, led5\_g)。
* 模組實例化：實例化 Marquee 模組，並將宣告的訊號連接到模組的埠.
* 時脈生成：initial 和 forever 區塊用於生成時脈訊號 clk。
* 重置生成：initial 區塊產生重置訊號 rst。
* 監測訊號：initial 區塊中的 $monitor 用於在模擬過程中印出訊號的值，這對於除錯很有幫助。

2. Please write the testbench yourself and run pre-simulation

* testbench內容已附加於檔案中

3. Add the signals you think are necessary to the waveform, then screenshot the waveform and explain how to determine if it is functioning correctly

* 為了驗證設計的功能，我們需要在波形檢視器中觀察以下訊號：
  + clk：時脈訊號
  + rst：重置訊號
  + current\_state：狀態機的當前狀態
  + counter：計數器的值
  + round\_type：速度模式指示器
  + led：LED 的輸出值
  + led4\_g：速度指示 LED 4
  + led5\_g：速度指示 LED 5
* 波形截圖與功能驗證

**[ Practice 3 ]**

**Q1: Please explain how you implement “Pulse\_Generator” & “Mechanical Counter”**

1. How do you generate a single pulse regardless of the duration of the input signal ?

Pulse\_Generator 模組 (Pulse\_GEN.txt) 的目的是將一個輸入訊號 (in) 轉換為一個單時脈週期的脈衝 (out)。 它使用一個暫存器 executed\_once 來確保無論輸入訊號 in 的持續時間如何，只產生一個脈衝。

* 初始狀態： 模組包含兩個暫存器，executed\_once 和 pulse。 當重置訊號 (rst) 為高電位時，executed\_once 被初始化為 1'b0，pulse 被初始化為 0。
* 產生脈衝： 在時脈的上升緣，如果輸入 in 為高電位 (1'b1) 且 executed\_once 為 0，則 pulse 被設定為 1，並且 executed\_once 被設定為 1。 這樣就產生了一個單時脈週期的脈衝。
* 抑制後續脈衝： 如果 in 為高電位且 executed\_once 不為 0，則 pulse 會被設定回 0。 這確保了即使輸入保持在高電位，也只會產生一個脈衝。
* 輸出： 輸出訊號 out 被賦予 pulse 的值。

總結來說，executed\_once 暫存器就像一個旗標，它確保了只有在輸入訊號第一次變為高電位時，pulse 才會產生一個時脈週期的脈衝。之後即使輸入保持高電位，它也會阻止 pulse 再次被設定為高電位，從而產生單一脈衝。

2. How to implement the saturation feature ?

Mechanical Counter 模組 (MC.txt) 實作了飽和功能，以防止計數器超出其範圍。

* 計數器範圍： 計數器 counter 是一個 4 位元的暫存器，這意味著它可以計數的範圍是從 0 到 15。
* 飽和邏輯：
  + add 訊號被賦予一個條件，即只有當 up\_pulse 為高電位，且計數器小於或等於 15 時，add 才會為高電位。 這可以防止計數器向上計數超過其最大值 15。
  + minus 訊號被賦予一個條件，即只有當 down\_pulse 為高電位，且計數器大於或等於 0 時，minus 才會為高電位。 這可以防止計數器向下計數低於其最小值 0。
* 計數器更新：
  + 在 always 區塊中，計數器會根據 add 和 minus 訊號進行更新。
  + 如果 rst 為高電位，計數器重置為 0。
  + 如果 add 為高電位，計數器遞增 1。
  + 如果 minus 為高電位，計數器遞減 1。
  + 如果 add 和 minus 都為低電位，計數器保持其當前值。
* 通過這種方式，add 和 minus 訊號的條件邏輯確保了計數器不會超出其 0 到 15 的範圍，從而實作了飽和功能。

**Q2: Screenshot the waveform and explain how to determine if it is functioning correctly**

1. Please explain what changes you made to the design in order to facilitate waveform viewing during pre-simulation.

為了方便在預模擬期間查看波形，通常不需要對設計程式碼進行重大更改。然而，可以採取一些除錯和可見性的最佳實踐：

* 加入重要的內部訊號到頂層模組： 為了觀察子模組的行為，將子模組中重要的內部訊號連接到頂層模組 (MC\_TOP.txt) 並將它們作為頂層模組的輸出，這樣可以在模擬波形中輕鬆地觀察它們。
* 使用有意義的訊號名稱： 確保所有訊號都具有描述性名稱，以提高可讀性。 這在分析波形時非常有幫助。
* 註解程式碼： 在程式碼中新增註解，以解釋特定區塊的功能。 這有助於理解程式碼在做什麼，並使除錯更容易。

在本範例中，不需要修改原始程式碼 (BTN\_DEB.txt, MC.txt, MC\_TOP.txt, Pulse\_GEN.txt) 就能進行模擬和觀察波形。 提供的程式碼已經模組化，並且頂層模組 (MC\_TOP.txt) 連接了所有必要的訊號。

2. Please run pre-simulation (You may write your own testbench or use the one provided by the TA)

* testbench內容參考助教提供版本

3. Add the signals you think are necessary to the waveform, then screenshot the waveform and explain how to determine if it is functioning correctly

為了完整驗證設計，以下訊號應該新增到模擬波形中：

* clk：時脈訊號，用於參考時序。
* rst：重置訊號，用於驗證初始狀態。
* btn：2 位元的輸入訊號，代表按鈕。
* btn0\_deb：按鈕 0 的去雜訊後訊號。
* btn1\_deb：按鈕 1 的去雜訊後訊號。
* up\_pulse：由 Pulse\_GEN 模組產生的向上計數脈衝。
* down\_pulse：由 Pulse\_GEN 模組產生的向下計數脈衝。
* led：4 位元的輸出，顯示計數器的值。
* counter (在 MC 模組內)：計數器的內部值，用於除錯。
* cs，ns (在 BTN\_DEB 模組內)：BTN\_DEB 模組的當前狀態和下一個狀態，用於驗證去雜訊邏輯。
* executed\_once，pulse (在 Pulse\_GEN 模組內)：Pulse\_GEN 模組的內部訊號，用於驗證脈衝產生。

功能驗證：

* 重置： 驗證當 rst 為高電位時，led 輸出在幾個時脈週期後變為 0。
* 按鈕去雜訊：
  + 觀察 btn、btn0\_deb 和 btn1\_deb。 驗證 BTN\_DEB 模組是否消除了按鈕訊號中的任何雜訊或彈跳。 btn0\_deb 應該在 btn[0] 穩定一段時間後才改變，btn1\_deb 也是如此。
  + 驗證 BTN\_DEB 模組的狀態 (cs、ns) 是否按照狀態機邏輯正確轉換。
* 脈衝產生：
  + 觀察 btn0\_deb 和 up\_pulse，以及 btn1\_deb 和 down\_pulse。 驗證 Pulse\_GEN 模組是否在每個去雜訊後的按鈕訊號的上升緣產生一個單時脈週期的脈衝。
  + 驗證 Pulse\_GEN 模組的內部訊號 (executed\_once、pulse) 是否按照其邏輯正確變化。
* 計數：
  + 驗證 led 輸出是否根據 up\_pulse 和 down\_pulse 正確遞增和遞減。
  + 驗證計數器是否在達到最大值 (15) 和最小值 (0) 時飽和。 當計數器為 15 且 up\_pulse 為高電位時，led 應該保持在 15。 同樣地，當計數器為 0 且 down\_pulse 為高電位時，led 應該保持在 0。
* 時序： 驗證所有訊號是否在正確的時脈邊緣變化。

波形截圖

**[ Practice 4 ]**

**Q1: Please explain your overall design concept**

1. How to determine whether it's time to move to the next LED?

設計使用一個計數器 (counter) 和不同的速度模式來決定何時移動到下一個 LED

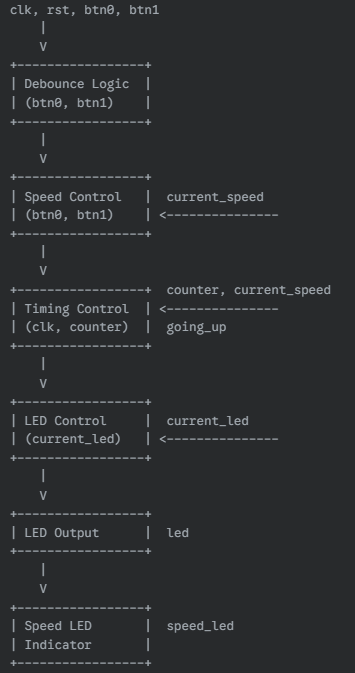
* 四個速度模式 (SPEED\_2S\_MODE, SPEED\_1S\_MODE, SPEED\_05S\_MODE, SPEED\_025S\_MODE) 對應於不同的計數器最大值 (SPEED\_2S, SPEED\_1S, SPEED\_05S, SPEED\_025S). 這些最大值決定了 LED 切換的速度。例如，SPEED\_2S 定義了最慢的速度，而 SPEED\_025S 定義了最快的速度
* always @(posedge clk or posedge rst) 區塊在每個時脈上升緣執行. 在這個區塊中，counter 會遞增
* 當 counter 的值達到當前速度模式對應的最大值時，current\_led 暫存器會被更新，從而改變點亮的 LED. update\_led 訊號也會被設置為 1，以指示 LED 需要更新
* 按鈕的輸入 (btn0 和 btn1) 會經過 debounce 處理（雖然題目假設已經處理過，但程式碼中包含了 debounce 邏輯）. 當檢測到有效的按鈕按下事件時，current\_speed 暫存器會被更新，從而改變 LED 移動的速度

2. Finite State Machine (FSM)

* **一張含有 文字, 字型, 圖表, 黑色 的圖片

  AI 產生的內容可能不正確。**FSM Diagram:
* State Description:
  + Going\_Up: LED 從 LED[0] 移動到 LED[3].
  + Going\_Down: LED 從 LED[3] 移動到 LED[0].
* Transition Conditions:
  + 從 Going\_Up 到 Going\_Down: 當 going\_up 為真且 current\_led 等於 3'd3 時.
  + 從 Going\_Down 到 Going\_Up: 當 going\_up 為假且 current\_led 等於 3'd0 時.
  + rst 為高電位時，going\_up 初始化為 1'b1 (Going\_Up).
* Output Behavior:
  + 在 Going\_Up 狀態，current\_led 遞增，直到達到 3'd3.
  + 在 Going\_Down 狀態，current\_led 遞減，直到達到 3'd0.
  + led 的輸出由 current\_led 的值決定: led <= (1 << current\_led);

3. Overall Architecture Diagram

* System Architecture Diagram:
* Component Functions:
  + Debounce Logic (btn0, btn1): 對按鈕輸入進行 debounce 處理，產生乾淨的脈衝訊號.
  + Speed Control (btn0, btn1): 根據按鈕輸入 (btn0, btn1) 調整 current\_speed 的值，從而改變 LED 移動的速度.
  + Timing Control (clk, counter): 使用時脈 (clk) 和計數器 (counter) 來控制 LED 切換的時間. 它也控制 going\_up 訊號來決定 LED 移動的方向.
  + LED Control (current\_led): 根據current\_led值選擇點亮的 LED.
  + LED Output: 4 位元的 LED 輸出 (led).
  + Speed LED Indicator: 根據 current\_speed 的值控制 speed\_led 的輸出，指示當前的速度模式.
* Code Mapping:
  + Debounce Logic (btn0, btn1): 對應到 always @(posedge clk or posedge rst) 區塊中處理 btn0, btn1, btn\_history\_up, btn\_history\_down, debounce\_counter\_up, 和 debounce\_counter\_down 的部分.
  + Speed Control (btn0, btn1): 對應到 always @(posedge clk or posedge rst) 區塊中根據 btn0 和 btn1 的上升緣更新 current\_speed 的 case 敘述.
  + Timing Control (clk, counter): 對應到 always @(posedge clk or posedge rst) 區塊中更新 counter 和 current\_led 的 case 敘述，以及 going\_up 暫存器的控制邏輯.
  + LED Control (current\_led): 對應到 always @(posedge clk or posedge rst) 區塊中 led <= (1 << current\_led); 這一行.
  + LED Output: 對應到 led 輸出埠.
  + Speed LED Indicator: 對應到 always @(posedge clk or posedge rst) 區塊中根據 current\_speed 的值更新 speed\_led 的 case 敘述.

**Q2: Screenshot the waveform and explain how to determine if it is functioning correctly**

1. Please explain what changes you made to the design in order to facilitate waveform viewing during pre-simulation.

因為實際的運作時間需要模擬太久，故運行模擬時將所有執行時間縮小為1/1000000

Testbench 中包含了以下關鍵部分：

* 訊號宣告：Testbench 宣告了需要連接到被測試模組 (ButtonControlledMarquee) 的訊號 (clk, rst, btn0, btn1, led, speed\_led).
* 模組實例化：它實例化了 ButtonControlledMarquee 模組，並將宣告的訊號連接到模組的埠.
* 時脈生成：initial 和 forever 區塊用於生成時脈訊號 clk.
* 重置生成：initial 區塊產生重置訊號 rst.
* 按鈕輸入生成：initial 區塊生成按鈕輸入訊號 btn0 和 btn1，模擬按鈕的按下和釋放.
* 監測訊號：initial 區塊中的 $monitor 用於在模擬過程中印出訊號的值，這對於除錯很有幫助.

2. Please write the testbench yourself and run pre-simulation

* testbench內容已附加於檔案中

3. Add the signals you think are necessary to the waveform, then screenshot the waveform and explain how to determine if it is functioning correctly

為了驗證設計的功能，我們需要在波形檢視器中觀察以下訊號：

* clk：時脈訊號.
* rst：重置訊號.
* btn0：速度加快按鈕輸入.
* btn1：速度減慢按鈕輸入.
* led：LED 的輸出值.
* speed\_led：速度指示 LED 的輸出值.
* counter：計數器的值.
* current\_led：當前點亮的 LED 的索引.
* current\_speed：當前的速度模式.
* going\_up：LED 移動方向的指示器.

波形截圖與功能驗證

**一張含有 螢幕擷取畫面, 電腦 的圖片

AI 產生的內容可能不正確。**

**[ 心得 ]**

組員一 姓名彭淳漵 學號E24131013

(心得從此處開始打)

組員二 姓名劉宸暟 學號E24136487

(心得從此處開始打)

這次的實驗相當充實，我負責了比前幾次還多很多的程式份量，雖然有時候完成一些任務的有些意想不到；也有時候覺得自己是炸專案體質，但整體而言我成長了很多，謝謝我的隊友還有辛苦幫我找錯誤的助教們。