

# Lab 3: Sequential Circuits

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

## Table of Contents

## 1 Q1: 4-bit Ping-Pong Counter

### 1.1 Implement

這題需要實作一個 4-bit 的 Ping-Pong Counter，會從 0 開始累加，直到 15 後再從 15 開始減少至 0，不斷循環。

在 verilog 程式碼方面，我們使用了多個 if 條件式，判斷目前屬於什麼狀態：

- $rst\_n = 0$ : 將  $started, direction$  設為 true, 並將輸出  $out$  設為 0。
- $started \& enable$ : 代表 Counter 正在運作，此時會根據  $direction$  進行加減：
  - $out = 15, direction = 1$ : 代表再繼續往上就會超出範圍，因此將  $direction$  設為 0 並將輸出  $-1$
  - $out = 0, direction = 0$ : 與上個情況相反，將  $direction$  設為 1 並將輸出  $+1$
  - 其他情況：根據  $direction$  進行加減

這裡要放 code 截圖

### 1.2 Circuit

在電路方面，需要實作的部分如下：

- $started, direction, out$  儲存：使用了三個 D-Flip-Flop 來儲存
- 計算下一個 clock cycle 的  $out, direction$ : 每個 clock cycle 會根據當下的  $out, direction$ ，透過加減法器預先算出  $out + 1, out - 1$ ，再透過多個 AND gate, NOT Gate，來實作出上述的條件式，分別導至四條線路。最後，再利用 MUX，根據對應的線路與條件來決定下一個 clock cycle 的  $out$  以及  $direction$ 。
- $enable$ : 算出  $out, direction$  之後，利用 MUX 處理  $enable$ ，若是  $enable = True$ ，則將新的  $out, direction$  輸入至 D-Flip-Flop，否則就輸入原來的值。
- $reset$ : 在上述訊號都處理完後，會再經過一關判斷是否需要 reset，如果不用則使用上述計算出來的值，否則就將該值設為初始值。

這裡要放電路圖

### 1.3 Testbench

由於操作相較單純，因此直接利用一個迴圈跑  $2^8$  次，每次都將  $enable$  反轉，觀察輸出是否符合預期：

這裡放程式碼截圖

這裡放波形圖部分

這裡放波形圖全貌

## 2 Q2: First-In First Out (FIFO) Queue

### 2.1 Implement

這邊要實現一個 8 個 8-bit 資料的 Queue (First in First out)，實作內容主要以這幾個數值的計算為主：

- *started*：是否開始運作，為 register，會在每個 posedge 時被計算
- *raddr*：讀取的位址，為 register，會在每個 posedge 時被計算
- *waddr*：下一次要寫入的位址，為 register，會在每個 posedge 時被計算
- *count*：目前 Queue 中有幾個資料，為 register，會在每個 posedge 時被計算
- *error*：是否有錯誤，為 wire，判斷方式為：當  $count = 0$  且  $ren = 0$ ，或是  $count = 8$  且  $wen = 1$  時，*error* 為 True，前者代表沒有資料可以讀取，後者代表沒有空間可以寫入。

而操作過程如下：

- (1) reset：當 *rst\_n* 與 *started* 皆為 False 時，將 *started* 設為 True
- (2) 開始後，如果沒有錯誤且  $ren = 1$ ，則將輸出設定為 *raddr* 的位置、將 *raddr* 設為  $raddr + 1$ ，以及將 *count* 減一
- (3) 如果上述條件都沒有達到，且要寫入的話，則將 *waddr* 的位置設定為 *din* (data input)、將 *waddr* 設為  $waddr + 1$ ，以及將 *count* 加一

以此邏輯實現的程式碼就會如下圖所示：

這裡放程式碼截圖

## 2.2 Circuit

### Memory

首先先介紹我們設計的記憶體單位，我們透過 8 個 8-bit 的 D-Flip-Flop 作為儲存單位，而記憶體控制的部分，除了 clock 之外會接收 4 個參數，分別為：

- *ren*：是否讀取記憶體
- *wen*：是否寫入記憶體
- *addr*：讀取或寫入的記憶體位置
- *din*：寫入的資料

並且會輸出 *dout*，代表讀取的資料。

判斷邏輯會是這樣的步驟：

- (1) 如果  $ren = 1$ ，那麼不執行寫入，直接輸出 *addr* 這個位置的值
- (2) 如果沒有要讀取，且要寫入的話，那麼就將 *din* 的值設為是 *addr* 這個位置新的值
- (3) 沒有要讀取也沒有要寫入的話，就輸出 0 然後不對 DFF 做任何動作

這裡放程式碼截圖

在電路實現的部分，大致會分成幾層：

- (1) 判斷輸入：首先使用比較器對每個位址  $i$  判斷  $addr$  是否等於  $i$ ，並將其結果與  $!ren \& wen$  做 AND，得到的結果便是這個位址是否要寫入。
- (2) D-Flip-Flop 控制：在每個 DFF 前加上一個 MUX，根據上述的結果決定 DFF 的輸入是  $din$  還是原本的值
- (3) 輸出判斷：與第一步相似但相較簡單，直接使用比較器判斷  $addr$  是否等於  $i$ ，並將結果與 DFF 的輸出做 AND，就能得到這個位址的輸出值。由於只會有一個位址會有輸出值，因此在最後直接做 Bitwise OR 就可以得到最終的輸出。

下圖演示的是 8 個 8-bit 的記憶體，可根據需求調整每個單位的位元大小以及有幾個單位。

這裡放電路圖

## FIFO

FIFO 的部分則是利用了上述的記憶體單位，計算出要傳給 Memory 的  $ren, wen, addr, din$  後，得到輸出的結果。

這裡放電路圖解釋

這裡放電路圖截圖

## 2.3 Testbench

我們寫了兩個 Testbench，首先是重現題目範例中的波形圖：

接著是我們使用 SystemVerilog 提供的 Queue，進行更完整的測試。由於  $ren, wen, din$  的組合數太多，因此我選擇了使用跑  $2^{10}$  測試，每次將這些輸入值設定為一個隨機值，藉此來跑到盡可能多的 case，並透過與 SystemVerilog 中，絕對正確的 Queue 來比對是否有錯誤。

## 3 Q3: Multi-Bank Memory

### 3.1 Implement

這題需要實作一個 Multi-Bank Memory，每個 Bank 有 4 個 Sub-Bank，每個 Sub-Bank 由 128 個 8-bit memory 組成，總共有 4 個 Bank。

輸入的部分由以下幾個部分組成：

- $ren, wen$ ：是否讀取或寫入
- $raddr$  (11 bit)：讀取的位址，前兩個 Bit 代表著是第幾個 Bank，接下來的兩個代表的是第幾個 Subbank。
- $waddr$  (11 bit)：寫入的位址，格式同上
- $din$  (8 bit)：寫入的資料

## Bank

這邊首先建立了四個長度 128，位寬 8-bit 的記憶體，而我們要做的事就是正確的算出要輸入到記憶體的參數是什麼：

- $ren$ ：利用與  $raddr[8:7] == i$  ( $i \in [0, 3]$ ) 做 AND 運算，來決定這個記憶體單位要不要讀取

- *wen*：與 *ren* 的處理方式一樣，只是改成使用 *waddr* 來判斷。
- *address*：透過三元運算子，當 *ren* = 1 時，將 *address* 設為 *raddr*，否則就設為 *waddr*

### 主 Module

主要的 Module 負責整體的輸入輸出，將輸入的參數正確的分配給各個 Bank，並將正確的 Bank 輸出導向至最後的輸出。

在處理輸入的部分，主要分為以下步驟：

- *ren*：直接利用  $ren \& (raddr[10:9]) == i$  ( $i \in [0, 3]$ ) 的方式來決定每個 Bank 的 *ren*
- *wen*：分為兩個步驟
  - (1) 判斷是否能夠輸入，題目規定，當要求同時讀寫同一個 Sub-bank 時，就以讀的操作為主，因此先利用 `wen && (ren == 0 || (waddr[10:7] != raddr[10:7]))` 判斷出這次的寫入操作是否合法
  - (2) 與 *ren* 用一樣的方式將上述的信號分配到正確的 Bank

至於輸出的部分，我使用了四個 8 bit 的 wire: *out*[3 : 0]，分別接至四個 Bank 的輸出，由於在處理輸入信號的部分，當該 Bank 不是要求的 Bank 時，output 會直接是 0，因此要取得正確的輸出就只要將四個 output 做 Bitwise OR 即可。

## 3.2 Circuit

### Memory

這部分沿用的是前面提到的 Memory，將其長度擴充為 128，由於篇幅關係就不額外展示長度 128 的電路圖。

### Bank

### 主 Module

## 4 Q4: Round-Robin FIFO Arbiter

### 4.1 Implement

## 5 Q5: 4-bit Parameterized Ping-Pong Counter