Digital System Design

HW03 : FIFO

105060016 謝承儒 電資 20

Part 1: 32X8 FIFO

一. 目的

設計出一個 FSM 來控制 FIFO 的 Push、Pop。

二. Input、Output、Others

1. Input :
   A. clk : 頻率 100MHz 的週期。
   B. rst_n : 當它為 0 則系統初始化。(回到 Empty 狀態)
   C. push : 當它為 1 時，將 datain 放入 RAM 裡。
   D. pop : 當它為 1 時，將存在 RAM 裡的值丟出來。
   E. [7:0] datain : 準備放入 RAM 裡面的值。。
2. Output :
   A. almost_full : 當它為 1 時，代表 RAM 只剩 1 個位置。
   B. full : 當它為 1 時，代表 RAM 已經滿了。
   C. almost_empty : 當它為 1 時，代表 RAM 只剩 1 個值。
   D. empty : 當它為 1 時，代表 RAM 已經空了。
   E. error : 當它為 1 時，代表有錯誤發生，例如 :
      a. 在 RAM 已經空的時候，仍然 Pop。
      b. 在 RAM 已經滿的時候，仍然 Push。
      c. 同時按下 Push、Pop。
      d. 在 Pop 的 state 時，仍然 Push。
   F. [7:0] dataout : 被 Pop 出來的值。
3. Others :
   A. cen, wen, oen : 分別控制 RAM、寫入 data、丟出 data 的開關，皆為 0 啟動。
   B. [4:0] addr : 指向現在要對哪一個位置做事(寫入、讀出 data)。

三. 設計過程

首先，我們的 FSM 有 4 個狀態，分別是 Empty、 Between、 Readout、Full。

此外，還有分別代表存到哪的 head、輸出到哪的 tail，以及代表要指向下一個的訊號 head_plus、tail_plus。

1. head 跟 tail 的值為何

   起初，head 跟 tail 的值皆為 0，每當 Push 一個值進來後，就讓 head + 1(下個 data 往下一個地方存)。

   同理，每當 Pop 一個值進來後，就讓 tail + 1(往下一個地方丟出 data)。

   但為了避免讓 head、tail 發生 overflow，所以最後要再+1 後放上%32，如此便不會超出 0~31 這個範圍。

2. 如何判斷是否已經要滿或是要空

   當 head 已經到 tail 的前一個時，就代表這是最後一個空位，此時就是已經要滿了。

   反之，當 tail 已經到 head 的前一個時，就代表這是最後一個 data，此時就是已經要空了。

3. 各個 state 間的轉換

   基本上就和老師給的 FSM 圖一樣。下表是待會用到的名稱，但表 4 種不同的 Input 情形。

	Push	pop
do_idle	0	0
do_push	1	0
do_pop	0	1
do_push_pop	1	1

   A. ENPTY ： RAM 裡面是空的

      這是最初始的狀態，若 rst_n=0 就會回到這個 state。

      a. 當 do_push 時，下個 state 為 BETWEEN，而因為要讓 data 進去，要讓 wen=0、addr=head，並且讓 head_plus=1，如此在放完 data 後，head 就會指向下一個空位。

      b. 其餘 3 種情形，並不會讓 state 有任何變化。若是為 do_pop 或 do_push_pop，則會產生出 error。

B. BETWEEN ： RAM 裡面有值，可以 Push 或 Pop

a. 當 do_push 時，做的事和前面所提的一樣，但要額外注意是否已經要滿了。若已經要滿了(almost_full=1)，下個 state 為 FULL;反之，仍然為 BETWEEN。

b. 當 do_pop 時，下個 state 為 READOUT，因為要準備把 data 丟出來，將 addr=tail 並把寫入 data 的功能關掉(wen=1)。

c. 其餘 2 種情況，state 維持不變，若為 do_push_pop 則產生 error。

C. Readout ： RAM 正在 Pop

因為正在把 data 丟出來，所以 oen=0。在這 state 只需要判斷丟出來的 data 是否為最後一個，以此來決定下個 state 為和。

a. data 不是最後一個(almost_empty=0)，下個 state 回到 BETWEEN。

b. data 是最後一個(almost_empty=1)，下個 state 回到 Empty。

而在 Pop 完後，需要將 tail 指向下一個 data，因此 tail_plus=1。

而在這 state 時，有 do_pop 或 do_push_pop，則會產生出 error，就會產生出 error。

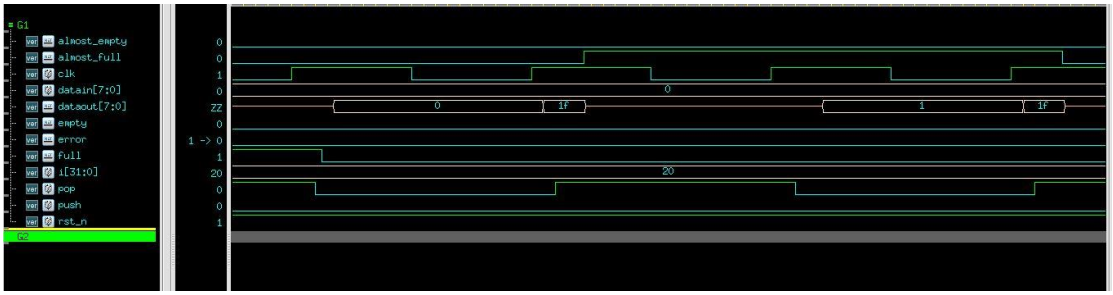
D. Full ： RAM 裡面是滿的

a. 當 do_pop 時，發生的事情就和在 BETWEEN 一樣。

b. 其餘 3 種情形時，state 不會變。但因為已經滿了，所以 do_push 會產生出 error。而 do_push_pop 本來就會造成 error。

四. Bug 與解決方法

1. 在 Pop 出 data 後，會在後面短暫出現其他 data，但不久後就消失了。如圖 ：



仍不曉得原因。

Part 2 : 64X16 FIFO

一. 目的

藉由 Part1 的 32x8 FIFO 做出 64x16 FIFO。

二. Input、Output、Others

基本上和 Part1 的部份一樣，只提出更動或新增的東西。

1. Input :
   A. [15:0] datain
2. Output :
   A. [15:0] dataout
3. Others :
   A. [5:0] addr : 因為從 32 變成 64，故多 1 bit。
   B. [4:0] addr1, addr2 : 將 64x16 拆成 2 個 32x8 來看，故會有 2 個 addr。
   C. [7:0] dataout1, dataout2, dataout3, dataout4 : 利用 2 個 8-bit 相接在一起，製造出能裝下 16-bit 的 datain 的空間(1、2 一組、3、4 一組)。
   D. wen1, wen2, oen1, oen2:
   1 結尾的是 No.1、2 一組 32x16 的 RAM 的 wen、oen，2 結尾的是 No.3、4 一組 32x16 的 RAM 的 wen、oen。

三. 設計過程

FSM 不需要做修改。

為了製造出 64x16 的空間，我們需要先做 4 個 RAM，並且想像他們是如下圖般組合 :

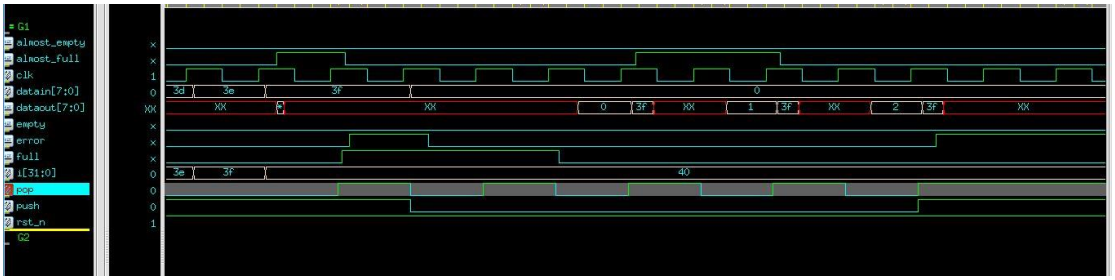
RAM1	RAM2
RAM3	RAM4

1. 如何存放 16-bit 的 datain
   使用 2 個 8-bit 的東西來接前後 8-bit([15:8]、[7:0])

2. FSM 出來的 5-bit addr，如何指向 addr1(1、2 一組)或是 addr2(3、4 一組)
   若 addr 小於 32，代表是要存放在上方 32x16 的 RAM(No.1、2)，所以 addr1=addr[4:0]。
   反之，代表是要存放在下方 32x16 的 RAM(No.3、4)，所以 addr2=addr[4:0]。
   addr 後面寫[4:0]是為了避免 overflow。
3. 上下兩組的 wen1/oen1、wen2/oen2 如何控制
   利用第一點的 addr 比較方法，來決定是上或下。此外還要判斷此時 wen/oen 是否為 0，才能將 wen1/oen1、wen2/oen2 設為 0。
   因此，像是寫入 data 到上方 RAM 的條件，會寫成這樣(~wen & (addr < 32))。

四. Bug 與解決方法

1. 在 syn 的波形圖中，在沒有 Pop 的情況下，dataout 會是 XX，如圖：



   仍然不曉得原因。

五. 討論

1. 如何改善需要 2-cycle 的 Pop 機制
   不曉得。
2. 如何把 FIFO 做成 Stack
   不使用 tail，Push 後仍然把 head+1，但 Pop 後改為 head-1。

六. 資料來源

1. 老師的 PPT