Team01 Lab1 Report

– 使用所需器材與架設方式

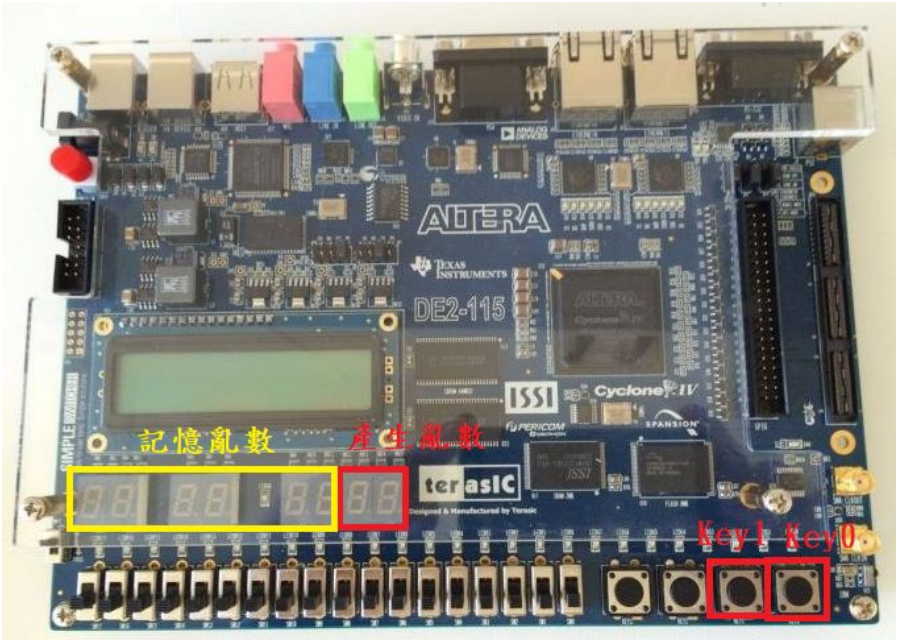
這次實驗用到的器材有DE2-115 FPGA 板、電源線與USB傳輸線。打開FPGA版的電源並將左下角的開關切到RUN。利用 USB 傳輸線將電腦與 FPGA 板連接後，用Quartus把程式燒到FPGA上，即架設完成。

DE2-115 FPGA 板

電源線

USB 傳輸線

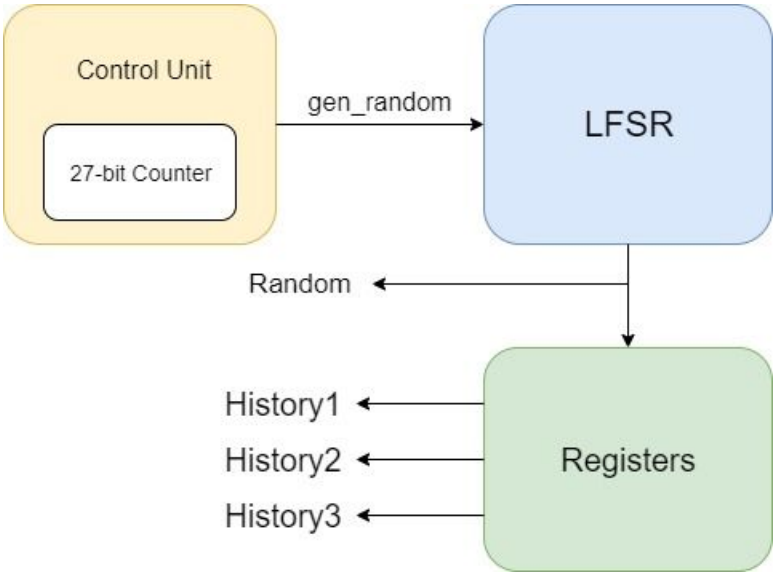
– 使用方式與詳細步驟



Key1：RESET 按鈕
Key0：START 按鈕
黃色的七段顯示器：分別紀錄前三次產生的亂數，從左到右依次為舊到新的三個亂數
紅色的七段顯示器：產生新的亂數

使用步驟：
1. 任何時候按下 RESET 按鈕之後，所有七段顯示器會歸零。
2. 按下 START 按鈕之後，紅色的七段顯示器會隨機產生0~15的亂數，產生亂數的頻率會由快逐漸變慢，最終停在一個數字上。
3. 當紅色的七段顯示器停在最終的亂數時，再次按下START 按鈕，紅色的七段顯示器會像步驟二一樣產生新的亂數，而黃色的七段顯示器則會擷取前一次停留的亂數。
4. 當紅色的七段顯示器上的亂數還在變動時，即按下START 按鈕，紅色的七段顯示器會像步驟二一樣產生新的亂數，而黃色的七段顯示器會直接擷取那一刻的亂數。

– 實作設計技術細節與巧思



主要架構：
電路主要架構如上圖，Control Unit會在適當的時機發送gen_random信號給LFSR，LFSR就會產生一組新的亂數，而此適當時機由Counter控制。
FSM：
Control Unit有兩個state，分別為IDLE及PROC。在IDLE時，Counter為0且不增加，此時若START被按下，就會跳至PROC。在PROC時，Counter會持續計數，當Counter計算至 $2^{27}-1$ 時，下一個cycle會回到IDLE。
控制跳動速度：
當Counter計數至某些特定數字時，會發送gen_random信號給LFSR產生新的亂數，這些特定數字之間的間隔若是越來越大，跳動的速度就會越來越慢。
中途擷取結果及儲存歷史紀錄：

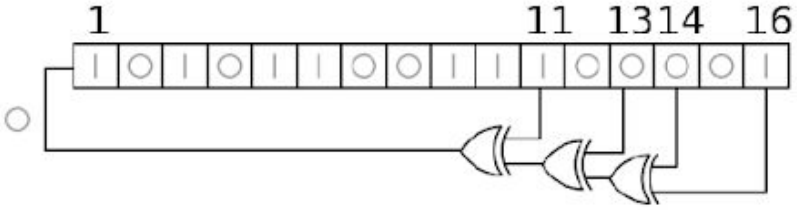
在任意時間按下START，會強制將Counter歸零及將state設至PROC，並把目前的亂數儲存至Register內。對應到的點名器行為即是，擷取目前跳動的數字儲存至歷史紀錄，並開啟新的一輪跳動。

– 碰過的問題或挑戰與解決方式

我們這次的實驗主要遇到了以下四個問題：

1. Linear feedback shift register的初始值以及抽頭

(抽頭:影響下一個狀態的比特位)



如圖為實驗說明所附的範例圖。決定利用LFSR產生亂數之後，我們原本想要直接依照此圖來進行設計，然而我們突然想到一個問題：如果初始值以及抽頭設計得不夠好，有沒有可能造成某個時刻之後的反饋值都為0或是都為1，進而使得取得的亂數值開始陷入一個過於簡單的週期，而看起來不再像是隨機產生？我們上網找了一些資料之後，發現一個如下所示的表。

n	LFSR-2	LFSR-4	n	LFSR-2	LFSR-4	n	LFSR-2	LFSR-4
2	2, 1		24		24, 23, 21, 20	46		46, 40, 39, 38
3	3, 2		25	25, 22	25, 24, 23, 22	47	47, 42	47, 46, 43, 42
4	4, 3		26		26, 25, 24, 20	48		48, 44, 41, 39
5	5, 3	5, 4, 3, 2	27		27, 26, 25, 22	49	49, 40	49, 45, 44, 43
6	6, 5	6, 5, 3, 2	28	28, 25	28, 27, 24, 22	50		50, 48, 47, 46
7	7, 6	7, 6, 5, 4	29	29, 27	29, 28, 27, 25	51		51, 50, 48, 45
8		8, 6, 5, 4	30		30, 29, 26, 24	52	52, 49	52, 51, 49, 46
9	9, 5	9, 8, 6, 5	31	31, 28	31, 30, 29, 28	53		53, 52, 51, 47
10	10, 7	10, 9, 7, 6	32		32, 30, 26, 25	54		54, 51, 48, 46
11	11, 9	11, 10, 9, 7	33	33, 20	33, 32, 29, 27	55	55, 31	55, 54, 53, 49
12		12, 11, 8, 6	34		34, 31, 30, 26	56		56, 54, 52, 49
13		13, 12, 10, 9	35	35, 33	35, 34, 28, 27	57	57, 50	57, 55, 54, 52
14		14, 13, 11, 9	36	36, 25	36, 35, 29, 28	58	58, 39	58, 57, 53, 52
15	15, 14	15, 14, 13, 11	37		37, 36, 33, 31	59		59, 57, 55, 52
16		16, 14, 13, 11	38		38, 37, 33, 32	60	60, 59	60, 58, 56, 55
17	17, 14	17, 16, 15, 14	39	39, 35	39, 38, 35, 32	61		61, 60, 59, 56
18	18, 11	18, 17, 16, 13	40		40, 37, 36, 35	62		62, 59, 57, 56
19		19, 18, 17, 14	41	41, 38	41, 40, 39, 38	63	63, 62	63, 62, 59, 58
20	20, 17	20, 19, 16, 14	42		42, 40, 37, 35	64		64, 63, 61, 60
21	21, 19	21, 20, 19, 16	43		43, 42, 38, 37	65	65, 47	65, 64, 62, 61
22	22, 21	22, 19, 18, 17	44		44, 42, 39, 38	66		66, 60, 58, 57
23	23, 18	23, 22, 20, 18	45		45, 44, 42, 41	67		67, 66, 65, 62

相關資料顯示，只要是如此表的設計，產生亂數的週期便為最大值，於是我們便採用了15個bits的設計，並只選取2個bits作為抽頭以減少電路面積

2. 從LFSR產生亂數值時比特位的選取

原本我們隨機選了4個bits做為亂數值的表達，然而再燒到FPGA板之後卻發現如此會造成初始值不是00，雖然實驗說明並無針對按下start鍵前的初始值做要求，但我們最後還是透過選取初始值均為0的4個bits做為亂數值的表達來將初始值設定為00，看起來較為正常。

3. Simulation使用的 clock 頻率與實際 FPGA 上的不同

這點雖然已經有在實驗說明中特別提到，但我們在進行debug的過程中卻仍曾經忘記此重要的特性。舉例而言，當我們一開始在透過模擬來debug時，觀察到電腦螢幕上數字的跳動頻率與我們預想的差不多，便沒有發覺其實我們的code是有bug的：gen_random始終為True，故LFSR每一cycle都給予register一個新的亂數值，但因為模擬時的頻率特別慢，所以我們一開始以為數字的跳動頻率是我們設計的cycle數，但卻沒想到當時的頻率其實是每個clock都跳動一次。我們也在後來突然注意點這點後才找到bug，並修正此問題。

4. syncronous reset or asyncronous reset?

原本的設計為syncronous reset，但當我們在檢查code的時候突然想到此裝置應該要為asyncronous reset較為合理，因為操作者不論何時按下reset鍵時，亂數都必須歸0。然而我們當時選擇先燒到FPGA板上觀察實驗結果再進行此類細節的修改，而這時我們發現，其實syncronous reset的設計也可以正常運作，原因是FPGA板上的clock頻率特別高，一個clock的週期遠遠小於我們按下按鍵的時間長短，所以儘管reset只能在clock的edge被觸發，按鍵處於被按下的時期中也一定包含clock的edge，所以事實上操作起來兩者的設計是感覺不到差異的，於是我們最後就沒有更改我們的設計，繼續採用syncronous reset了。