Didital Circuit Lab Team08 Lab1 Report

B06901190 陳昱仁, B06505021 楊欣哲, B06505011 傅敬倫

一.所需器材

1.Ncverilog:

用來 debug Verilog code

2.Quartus:

將 Verilog code 合成為可供 FPGA 使用的 sof 檔案

3.FPGA:

亂數生成器的呈現平台

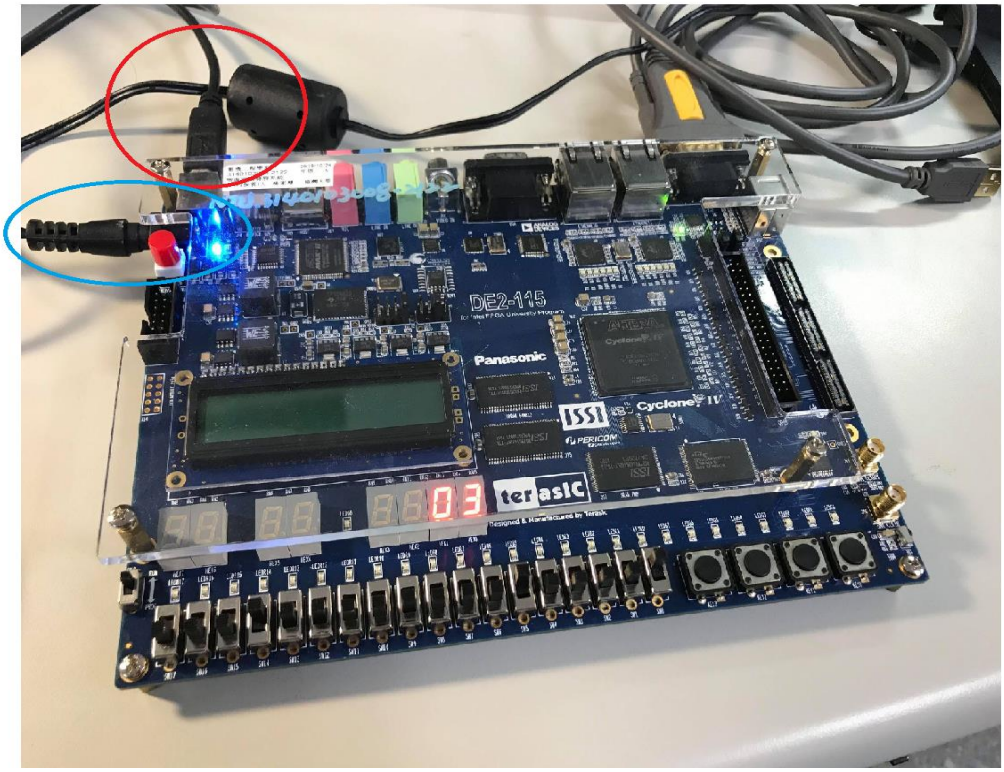
4.USB Cable:

連接電腦和 FPGA，將 Quartus 產生的 sof 檔案燒入 FPGA

5.12 volt adapter:

用來提供 FPGA 電源

二.架設方式



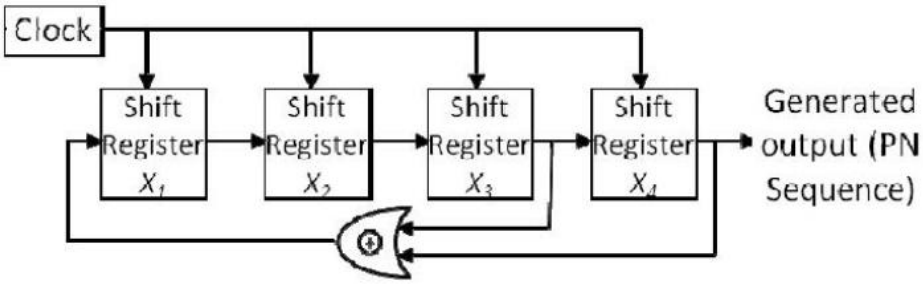
圖一

上圖為 FPGA 的架設圖。紅色圈圈中的是 USB Cable，用來連接電腦和 FPGA。藍色圈圈中的是 12 volt adapter，用來提供 FPGA 電源。

三.使用方式與詳細步驟

1. 根據亂數點名器的 requirements 設計相對應的 Finite state machine 和硬體電路設計圖。
2. 將上述設計用 verilog 實作，並同時寫出相對應的 testbench 檢查 verilog code 是否符合亂數點名器的 requirements。如果沒有通過 testbench，則需反覆 debug 直到通過為止，才能進入第三點
3. 如果沒有安裝 Quartus 需事先安裝。
4. 將 verilog code 和 sdf 檔案匯入 Quartus。選擇對應的 FPGA 板子，並 compile。如果沒有 error 則可進入第五點
5. 合成出可供 FPGA 運作的 sof 檔
6. 用 12 volt adapter 提供 FPGA 電源， USB Cable 連接電腦與 FPGA，詳見架設圖。
7. 操作 FPGA，檢驗是否符合亂數點名器的 requirements

四.硬體架構

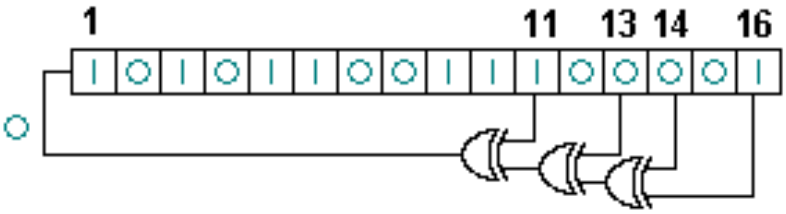


圖二

原先實作的 4-bits 亂數產生電路如上，是使用 LFSR，假設一開始的初始值 X4X3X2X1=1111，會產生如下的亂數循環:

1111→1110→1100→1000→0001→0010→0100→1001→0011→0110→1101→1010→0101→1011→0111→1111。

總共 15 個 state 依次循環，很明顯地看到，此電路有個明顯的缺點，他無法產生 0000 的數字，若進入 0000，此電路只會一直循環 0000，無法產生數字 0，且亂數序列過於固定，隨機性過低，每個數字前後的數字都一樣。



圖三

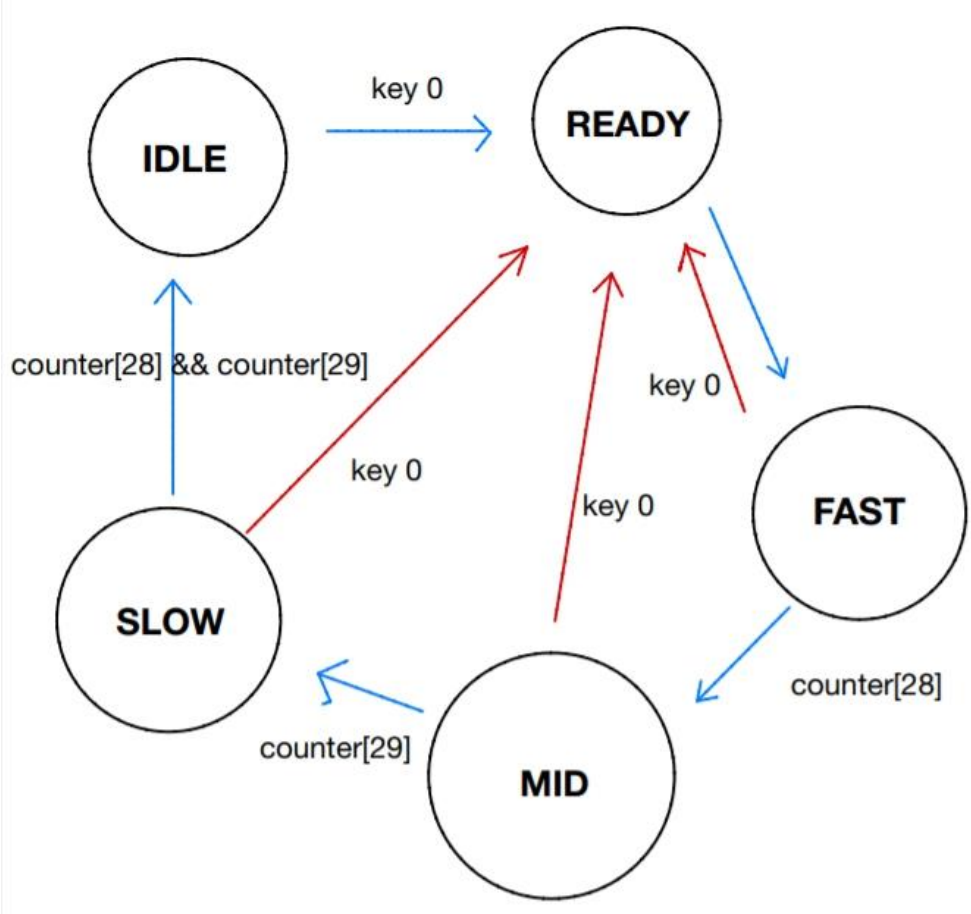
改良方法是把 4-bits LFSR 擴展成 16bits LFSR，每 4 個 bits 一組，每一組再用 4 to 1 MUX 選擇其中 1 bit 當作輸出，總共 4 bits 輸出，而作為 MUX 選擇的變數，則用 counter 或 timer 來代替，只要電路開始啟動，timer 就會從 0 開始數，擷取 timer 最小的 8 bits 來做為變數，且無規則的選取，如下圖程式碼。

```
assign select[0] = { timer[0], timer[4] };
assign select[1] = { timer[5], timer[3] };
assign select[2] = { timer[7], timer[6] };
assign select[3] = { timer[2], timer[1] };
```

圖四

如此一來，透過擴展成 16-bit 增加了隨機數字循環的 state 數，及 65535 個 state，原先只有 15 states，再加上用 timer 隨機選取其中 4 個 bits，雖然 timer 在每次啟動電路都會重新歸零，但是每次重新啟動時，flip flop 內存的值都不一樣，所以 seed 也不同，所以用 timer 的選取也增加了亂數產生的隨機性。

五.FSM



圖五

如上所示，此為此電路的 FSM，使用 asynchronous active low reset，state 分別代表如下:

1.IDLE: 電路靜止狀態，reset 或數字停止跳動都會在此 state。

2.READY : 不論在哪個 state，key 0 按下會到此 state，並把 counter 歸零，下一個 cycle 後會強制進入 FAST，不須任何條件。

3.FAST: 亂數跳動頻率會相當快，會在 counter 第 29 位數為 1 時跳到 MID。

4.MID: 亂數跳動頻率相較中等，會在 counter 第 30 位數為 1 時跳到 SLOW。

5.SLOW: 亂數跳動頻率最慢，會在 counter 第 29 和 30 位數為 1 時跳到 IDLE。

其中要控制亂數輸出的頻率，LFSR 為 combinatial, 所以數字會一直改動，我們在外部再用 register 存取值，但不是每個 cycle 都會存取 LFSR 吐出來的值，透過 counter，我們只讓他在特定 counter 值會賦予，在 FAST 階段，counter[21:0]的值要皆為 1 才給值，MID 階段 counter[23:0]的值要皆為 1 才給值，最後 SLOW 階段 counter[24:0]的值要皆為 1 才給值，否則輸出的數字皆為原先的數字，控制給值的頻率，如下圖所示。

```
always@(*)begin
    answer_reg_nxt = answer_reg;
    case(state)
        RUN_FAST:begin
            if(&counter[21:0])
                answer_reg_nxt = data_out;
        end
        RUN_MID:begin
            if(&counter[23:0])
                answer_reg_nxt = data_out;
        end
        RUN_SLOW:begin
            if(&counter[24:0])begin
                answer_reg_nxt = data_out;
            end
        end
        default:begin
            answer_reg_nxt = answer_reg;
        end
    endcase
end
```

圖六

六. BONUS

由上方 FSM 圖片可知，藍色線是正常跑的情況，但我們另外設定可以隨時按 key 0 讓他重新跑，圖片中紅色線，就算現在還沒跑完也可以按下 key 0 重新開始跑，是我們的一個小巧思。

七.問題發現與解決

我們這次的實作中，遇到一些問題，其中有些在上方已經提到並改善，如電路不夠隨機性以及如何產生 seed，輸出的快慢也在上方提出解決方法。

卡關的部分在於對 FPGA 不夠熟悉，以及座位常常沒有該有的線路或東西，桌上電腦還有密碼打不開，花了我們蠻多時間在弄 FPGA 及 Quartus 上。在模擬上，工作站圖形化界面數字跳動頻率也和 FPGA 不同，我們最後直接在 FPGA 上面測試並改動數字，上放 counter 的第幾位數字決定與否，從我們得知 FPGA 的頻率是 50M HZ，推估出一個大概後，再實際去跑再來微調。

最一開始跑 key 0 按下去後數字就不動了，我們以為是數字切換速率太慢，所以把切換速率調高，但調高後卻發現也時一樣不會跳動，後來才發現其實是數字切換太快，FPGA 無法顯示太快的數字變化，只會擷取其可適應的頻率上的數字，所以我們最後把切換速率調慢，也就得到上方那些 counter 數字，完成了我們第一次的實驗。