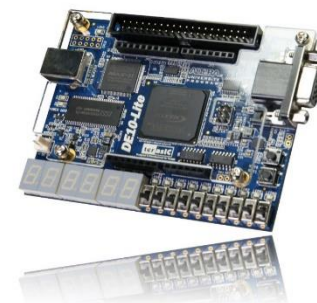


VHDL除頻電路與狀態機電路

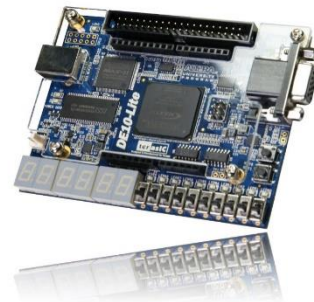
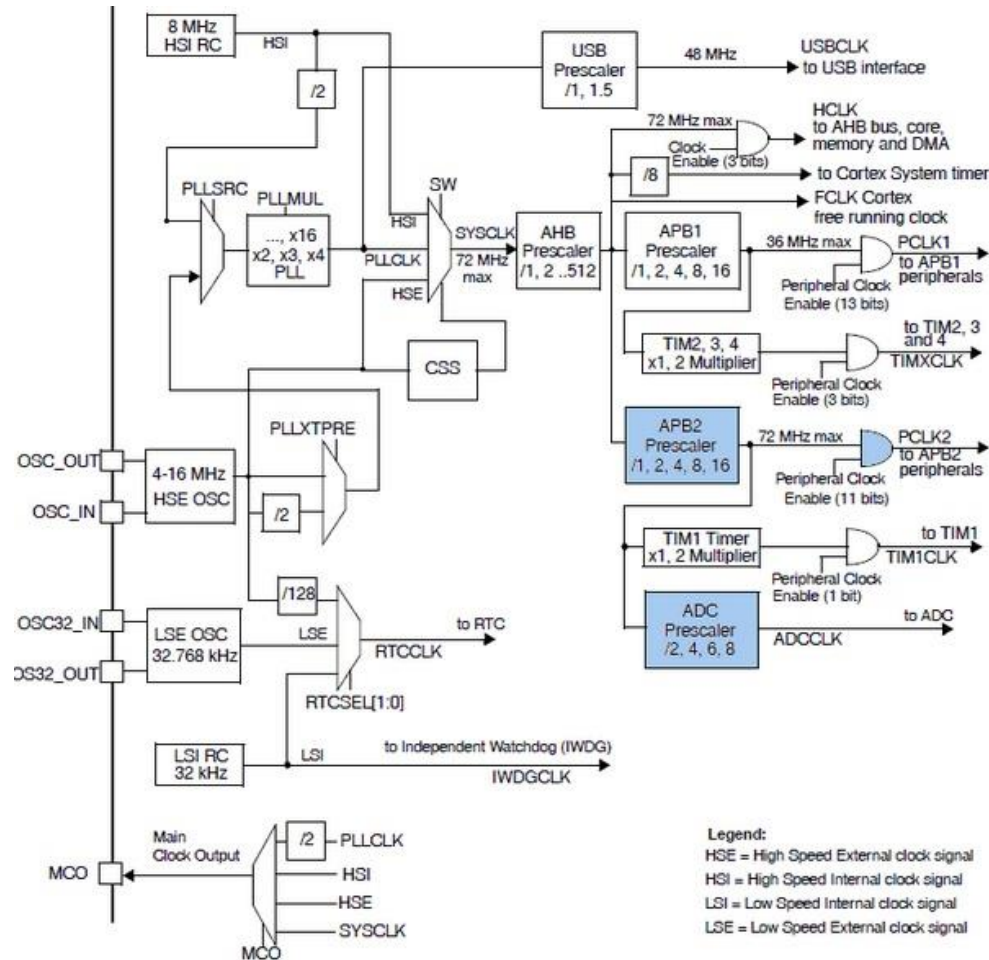
Outline

- 除頻電路
 - 非同步計數器
 - 同步計數器
 - 補充:RTL Schematic
- 延遲電路
 - 累積時脈延時
 - 基本延時電路延時
- 有限狀態機設計
 - 米利機(Mealy Machine)
 - 莫爾機(Moore Machine)



除頻器

- 為什麼需要除頻器



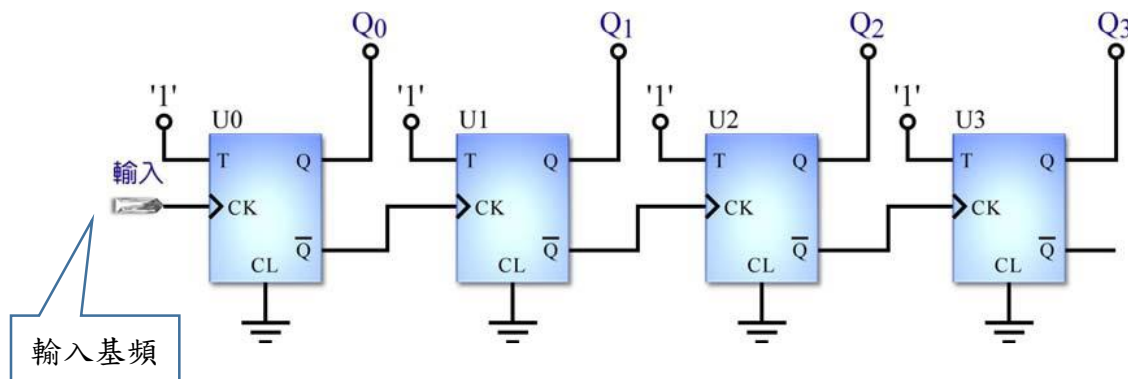
除頻器

- **非同步**計數器：

- 延遲誤差(Propagation Delay)會累積

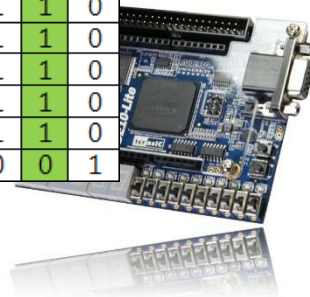
- 基頻來源

- 有意義的頻率



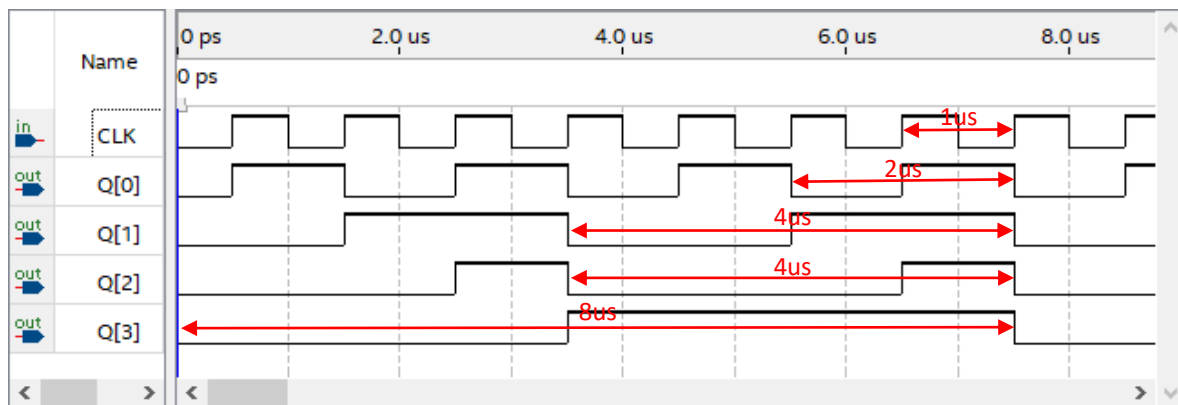
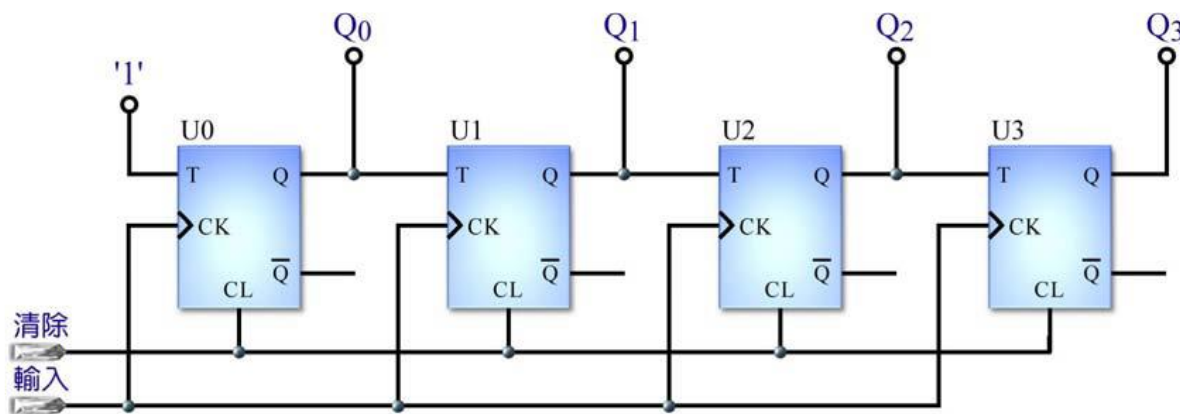
PR	CL	CK	T	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	Q	\bar{Q}
1	1	↑	1	\bar{Q}	Q

CK	Q0	Q0'	Q1	Q1'	Q2	Q2'	Q3	Q3'
0	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1
0	1	0	0	1	0	1	0	1
1	0	1	1	0	0	1	0	1
0	0	1	1	0	0	1	0	1
1	1	0	1	0	0	1	0	1
0	1	0	1	0	0	1	0	1
1	0	1	0	1	1	0	0	1
0	0	1	0	1	1	0	0	1
1	1	0	0	1	1	0	0	1
0	1	0	0	1	1	0	0	1
1	0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	0	1
1	1	0	1	0	1	0	0	1
0	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	1	0
0	0	1	0	1	0	1	1	0
1	1	0	0	1	0	1	1	0
0	1	0	0	1	0	1	1	0
1	0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1	0
1	1	0	1	0	0	1	1	0
0	1	0	1	0	0	1	1	0
1	0	1	0	1	1	0	0	1



除頻器

• 同步計數器



PR	CL	CK	T	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	Q	\bar{Q}
1	1	↑	1	\bar{Q}	Q

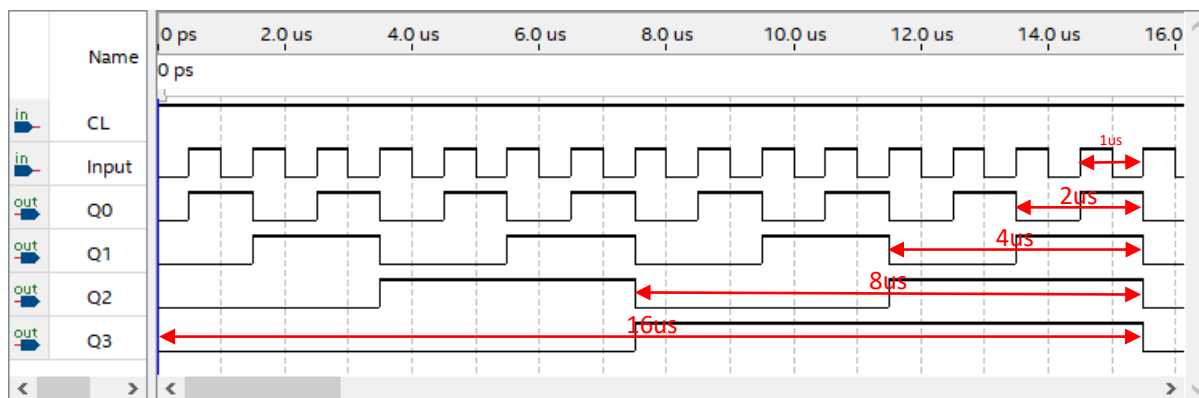
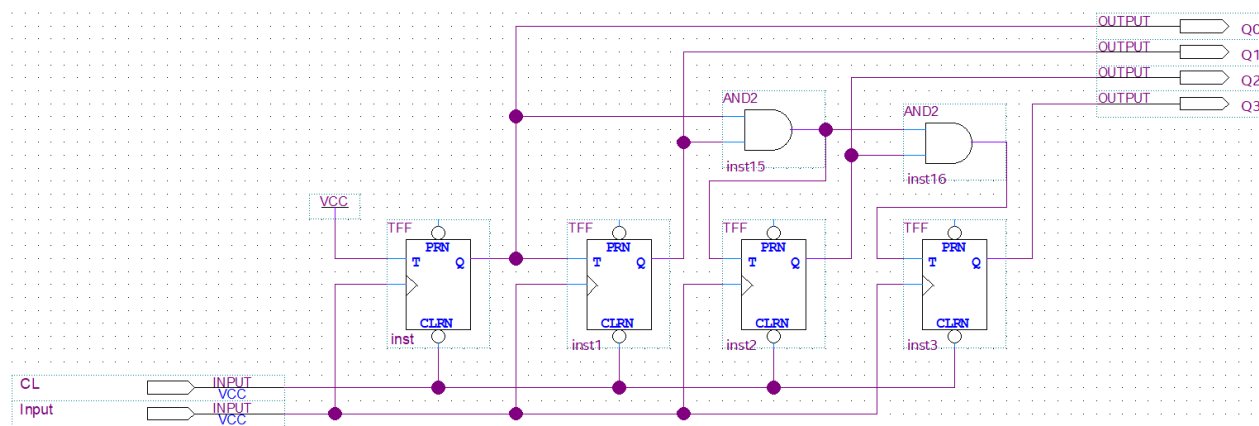
4位元同步下數計數器

CK	T1	Q0	Q1	Q2	Q3	
0	1	0	0	0	0	0
1	1	1	1	1	1	15
0	1	1	1	1	1	
1	1	0	1	0	1	10
0	1	0	1	0	1	
1	1	1	0	0	1	9
0	1	1	0	0	1	
1	1	0	0	0	1	8
0	1	0	0	0	1	
1	1	1	1	1	0	7
0	1	1	1	1	0	
1	1	0	1	0	0	2
0	1	0	1	0	0	
1	1	1	0	0	0	1
0	1	1	0	0	0	
1	1	0	0	0	0	0
0	1	0	0	0	0	
1	1	1	1	1	1	15
0	1	1	1	1	1	
1	1	0	1	0	1	10
0	1	0	1	0	1	
1	1	1	0	0	1	9
0	1	1	0	0	1	
1	1	0	0	0	1	8
0	1	0	0	0	1	
1	1	1	1	1	0	7
0	1	1	1	1	0	
1	1	0	1	0	0	2
0	1	0	1	0	0	
1	1	1	0	0	0	1
0	1	1	0	0	0	
1	1	0	0	0	0	0
0	1	0	0	0	0	



除頻器

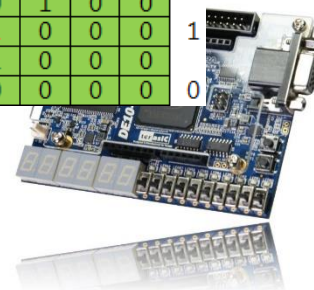
• 同步計數器(修正)



1MHz
500KHz
250KHz
125KHz
62.5KHz

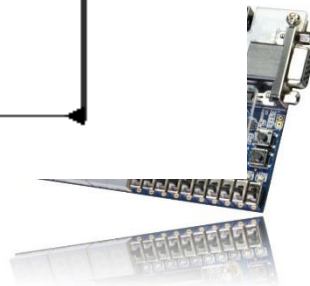
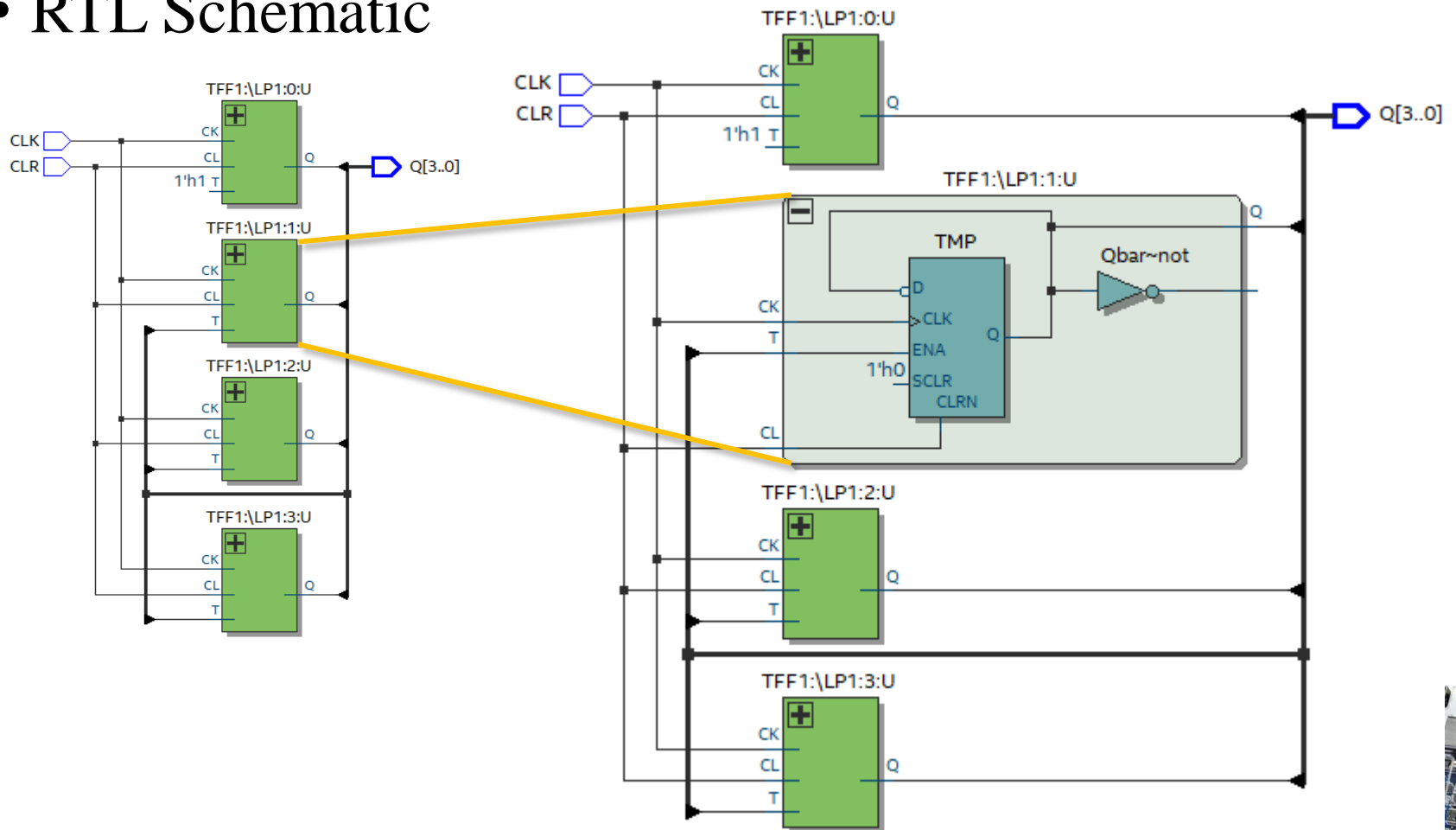
4位元同步下數計數器(修正)

CK	T1	Q0	Q1	Q2	Q3	
0	1	0	0	0	0	0
1	1	1	1	1	1	15
0	1	1	1	1	1	
1	1	0	1	1	1	14
0	1	0	1	1	1	
1	1	1	0	1	1	13
0	1	1	0	1	1	
1	1	0	0	1	1	12
0	1	0	0	1	1	
1	1	1	1	0	1	11
0	1	1	1	0	1	
1	1	0	1	0	1	10
0	1	0	1	0	1	
1	1	1	0	0	1	9
0	1	1	0	0	1	
1	1	0	0	0	1	8
0	1	0	0	0	1	
1	1	1	1	1	0	7
0	1	1	1	1	0	
1	1	0	1	1	0	6
0	1	0	1	1	0	
1	1	1	0	1	0	5
0	1	1	0	1	0	
1	1	0	0	1	0	4
0	1	0	0	1	0	
1	1	1	1	0	0	3
0	1	1	1	0	0	
1	1	0	1	0	0	2
0	1	0	1	0	0	
1	1	1	0	0	0	1
0	1	1	0	0	0	
1	1	0	0	0	0	0



補充

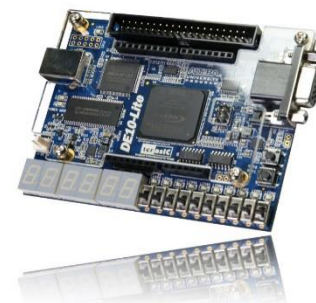
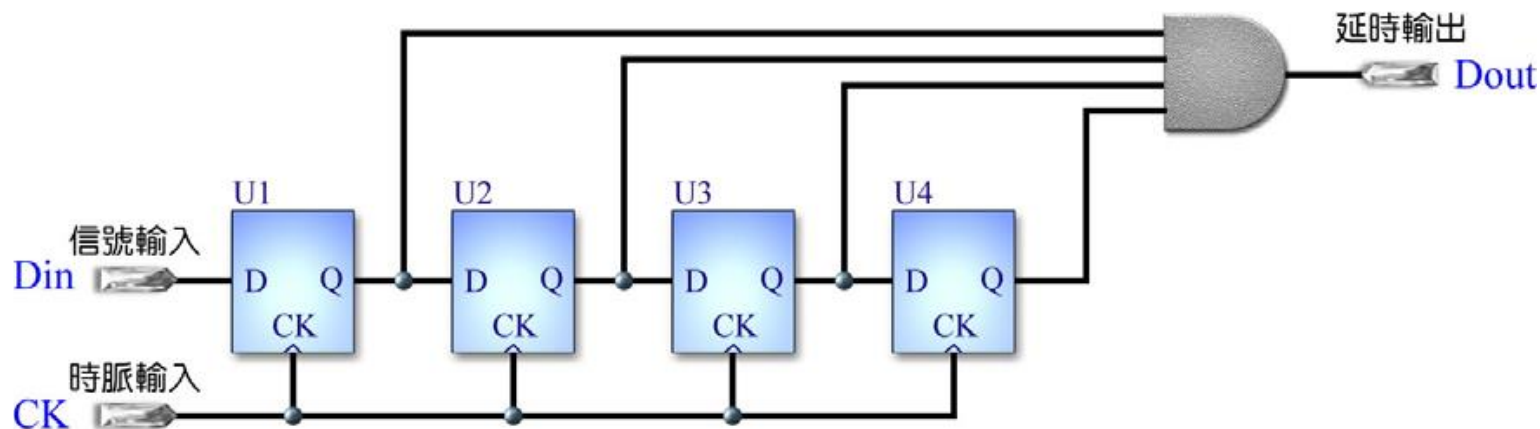
- RTL Schematic



延遲電路

- 累積時脈延時(1/4)
 - 延時輸出
 - 確保穩定輸入訊號
 - 防彈跳電路

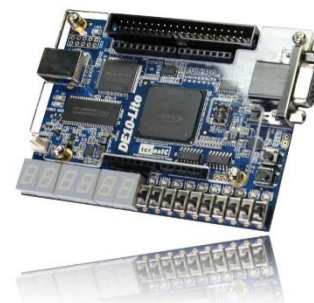
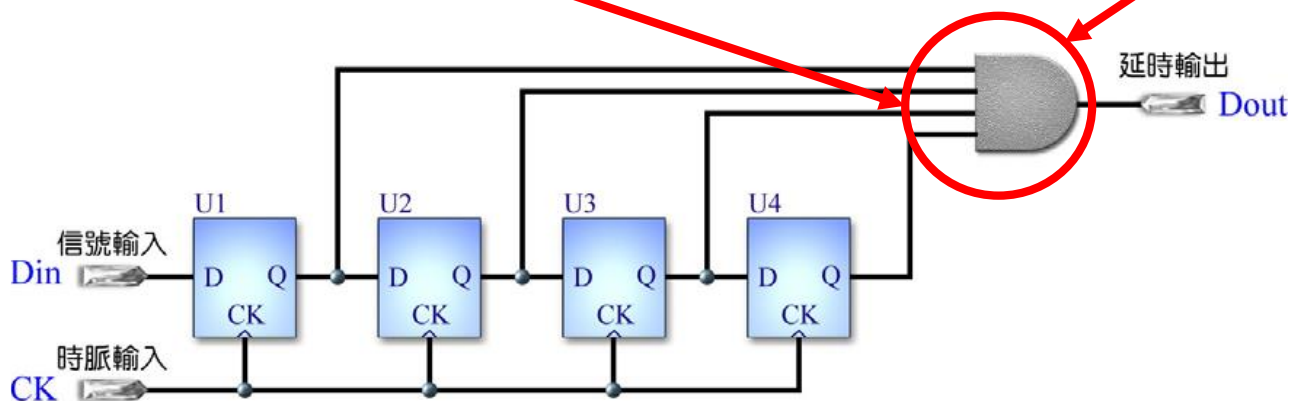
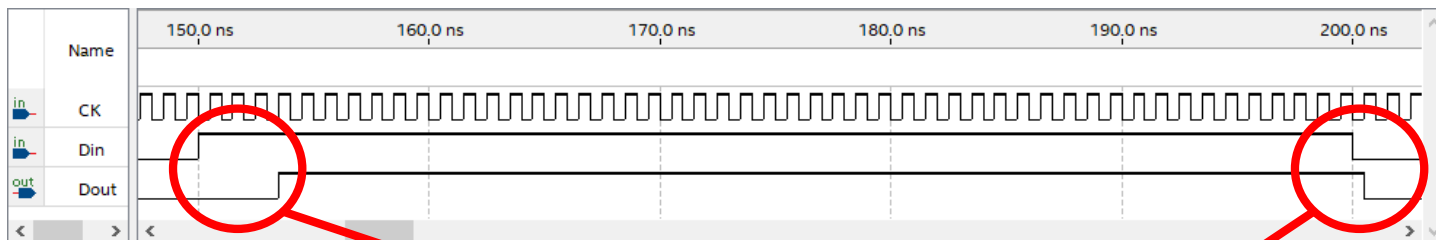
PR	CL	CK	D	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0



延遲電路

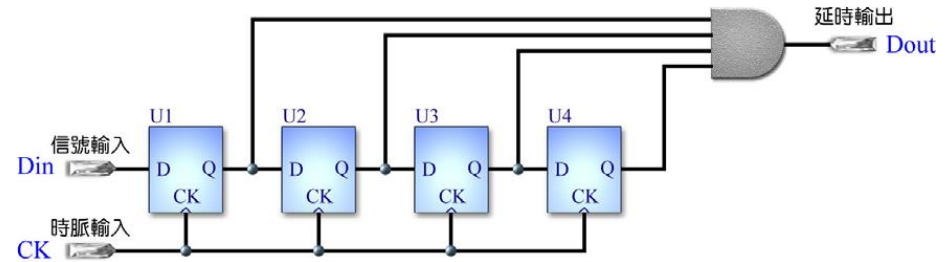
- 累積時脈延時(2/4)

PR	CL	CK	D	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0



延遲電路

- 累積時脈延時(3/4)



```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity Debouncing is
    Port( Din,CK:in std_logic;
          Dout:out std_logic);
End Debouncing;

Architecture ARCH of Debouncing is
    Component DFF1
        Port( D,CK:in std_logic;
              Q:out std_logic);
    End Component;

    signal TMP: std_logic_vector(4 downto 0);
    Begin
        TMP(0) <= Din;
        Lp1:For I in 1 to 4 Generate
            U1:DFF1 Port Map(TMP(I-1), CK,TMP(I));
        End Generate;
        Dout <= TMP(4) and TMP(3) and TMP(2) and TMP(1);
    End ARCH;
```

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity DFF1 is
    Port( D,CK:in std_logic;
          Q:out std_logic);
End DFF1;

Architecture ARCH of DFF1 is
    Begin
        Process(CK)
        Begin
            If Rising_Edge(CK) Then Q <= D;
            End If;
        End Process;
    End ARCH;
```



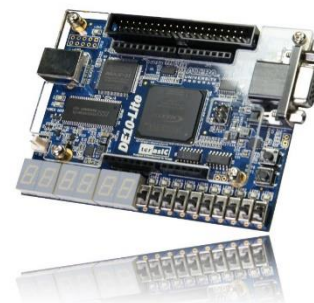
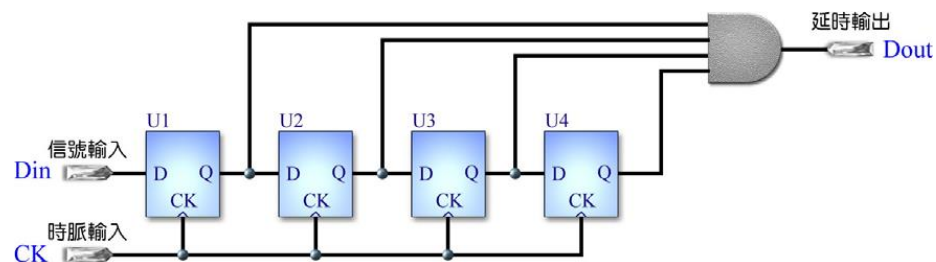
延遲電路

- 累積時脈延時(4/4)
 - 不使用DFF，利用VHDL直接敘述計時器

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity Debouncing2 is
Port( Din,CK:in std_logic;
      Dout:out std_logic);
End Debouncing2;

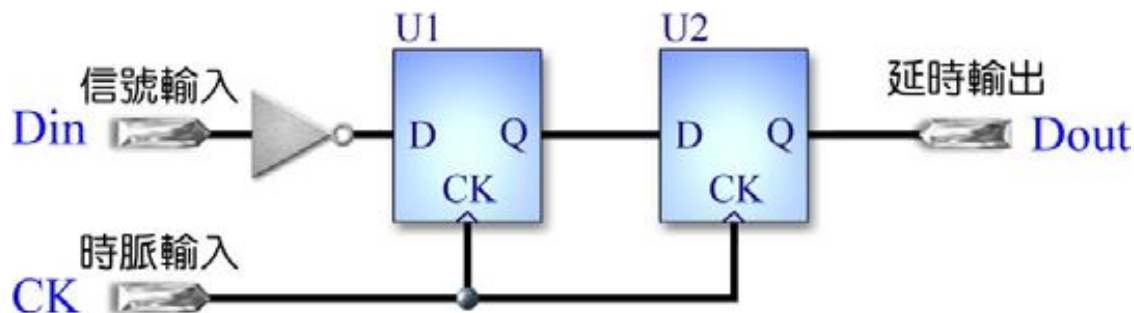
Architecture ARCH of Debouncing2 is
Begin
  Process(CK)
    Variable TMP:integer range 0 to 9;
  Begin
    If Rising_Edge(CK) Then
      If Din = '1' Then
        TMP := 0;
        Dout <= '1';
      Else
        TMP := TMP + 1;
        If TMP = 4 Then
          TMP := 0;
          Dout <= '0';
        End If;
      End If;
    End If;
  End Process;
End ARCH;
```



延遲電路

- 基本延時電路延時
 - 延時輸出

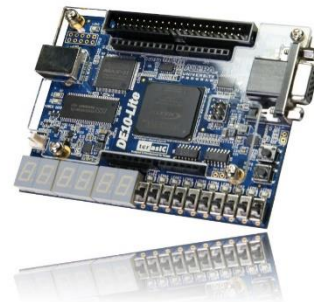
PR	CL	CK	D	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0



有限狀態機設計

- **狀態機(State Machine)**

- 依據循序邏輯電路的各種狀態繪製其狀態圖或狀態表來設計電路。
- 狀態越多，電路越複雜，由於無法實作出無限多的狀態數，所以稱為有限狀態機(**Finite State Machine**)。
- 傳統數位邏輯電路設計裡，有限狀態機的設計程序相當複雜，而採用VHDL電路設計，有限狀態機的設計程序變得相對簡單。
- 有限狀態機可有多個輸入、多個輸出，而依其輸出與輸入是否相關，可區分為下列兩種：
 - 米利機(Mealy Machine)
 - 莫爾機(Moore Machine)



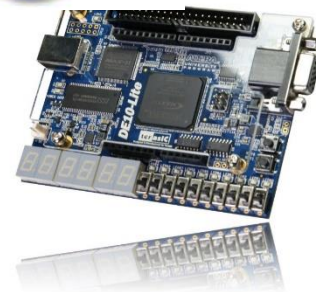
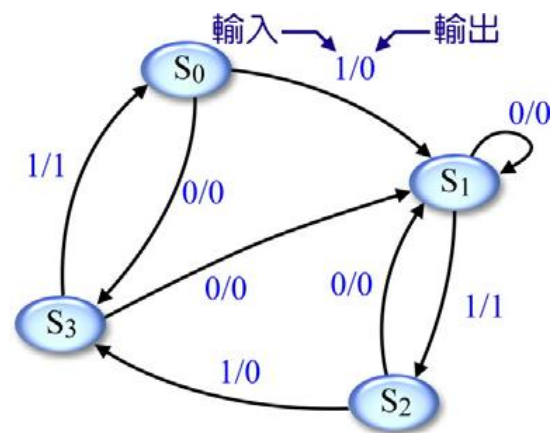
有限狀態機設計

• 米利機(Mealy Machine)

- 輸出狀態與目前狀態與輸入狀態有關。
- $Q: I \times PS \rightarrow O$ (I :輸入, O :輸出, PS :目前狀態 Present State)

右圖中包括 $S_0 \sim S_3$ 等4個狀態，如下說明：

1. S_0 狀態時，若輸入1，則輸出0且跳至 S_1 狀態。
若輸入0，則輸出0且跳至 S_3 。
2. S_1 狀態時，若輸入1，則輸出1且跳至 S_2 狀態。
若輸入0，則輸出0且維持在 S_1 狀態。
3. S_2 狀態時，若輸入1，則輸出0且跳至 S_3 狀態。
若輸入0，則輸出0且跳回 S_1 狀態。
4. S_3 狀態時，若輸入1，則輸出1且跳至 S_0 狀態。
若輸入0，則輸出0且跳至 S_1 狀態。



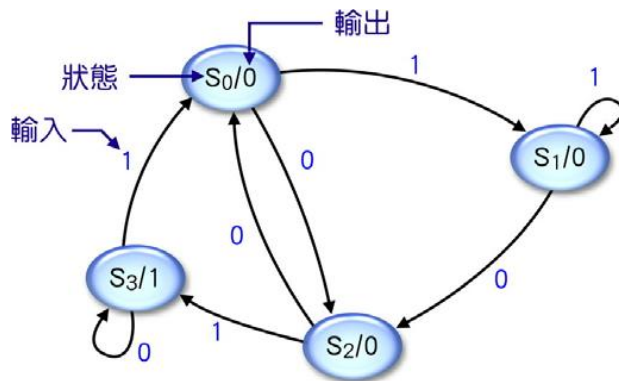
有限狀態機設計

• 莫爾機(Moore Machine)

- 輸出狀態只受目前狀態影響，與輸入狀態無關。
- $Q: PS \rightarrow O$ (I: 輸入, O: 輸出, PS: 目前狀態 Present State)

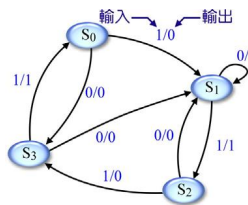
右圖中包括 **S0~S3** 等4個狀態，如下說明：

1. **S0** 狀態時輸出0，若輸入1，則跳至 **S1** 狀態。
若輸入0，則跳至 **S2** 狀態。
2. **S1** 狀態時輸出0，若輸入1，則保持在 **S1** 狀態。
若輸入0，則跳至 **S2** 狀態。
3. **S2** 狀態時輸出0，若輸入1，則跳至 **S3** 狀態。
若輸入0，則跳回 **S0** 狀態。
4. **S3** 狀態時輸出1，若輸入1，則跳至 **S0** 狀態。
若輸入0，則保持在 **S3** 狀態。

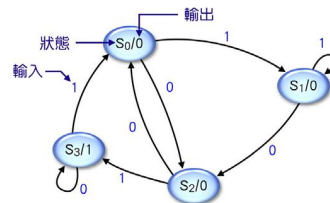


隨堂練習

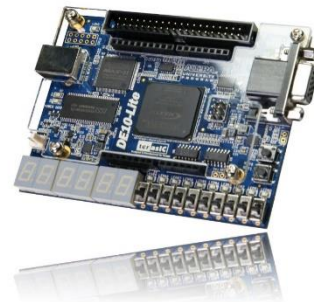
- 請使用VHDL完成下列電路，並完成紀錄，包括**VHDL Source Code**、**模擬波形圖**。
 - a) 累積時脈延時電路(以DFF為基礎設計電路)並觀察RTL
 - b) 累積時脈延時電路(使用VHDL直接描述DFF電路)並觀察RTL
 - c) 米利機電路設計



- a) 莫爾機電路設計

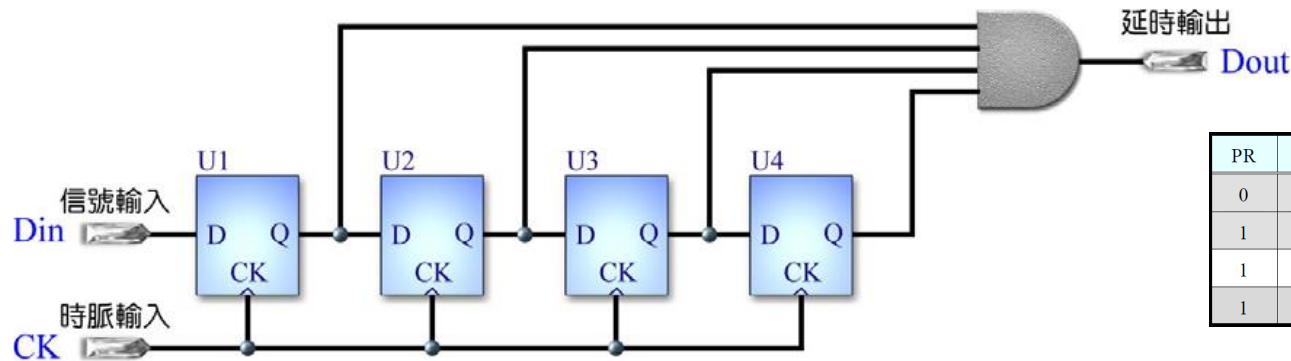


- 本次實驗完成後需助教確認正確，全部完成後，將專案與報告壓縮上傳EE-Class。
- Lecture9_組別XX.ZIP



隨堂練習(一)

- 累積時脈延時電路(以DFF為基礎設計電路)並觀察RTL



PR	CL	CK	D	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity Debouncing is
  Port( Din,CK:in std_logic;
        Dout:out std_logic);
End Debouncing;

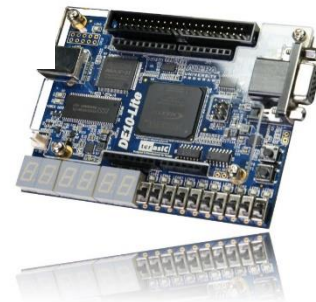
Architecture ARCH of Debouncing is
  Component DFF1
    Port( D,CK:in std_logic;
          Q:out std_logic);
  End Component;

  signal TMP: std_logic_vector(4 downto 0);
Begin
  TMP(0) <= Din;
  Lp1:For I in 1 to 4 Generate
    U1:DFF1 Port Map(TMP(I-1), CK,TMP(I));
  End Generate;
  Dout <= TMP(4) and TMP(3) and TMP(2) and TMP(1);
End ARCH;
```

```
Library IEEE;
Use IEEE.std_logic_1164.all;

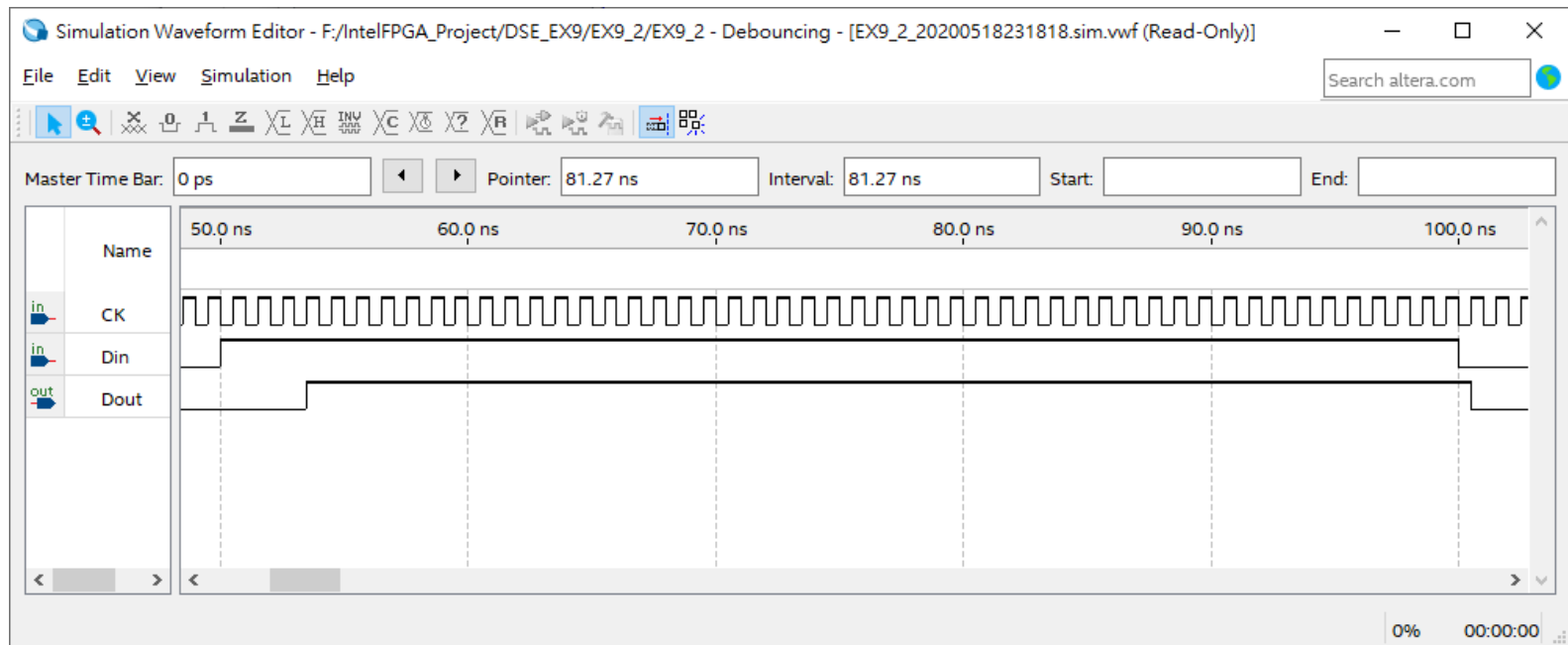
Entity DFF1 is
  Port( D,CK:in std_logic;
        Q:out std_logic);
End DFF1;

Architecture ARCH of DFF1 is
Begin
  Process(CK)
  Begin
    If Rising_Edge(CK) Then Q <= D;
    End If;
  End Process;
End ARCH;
```



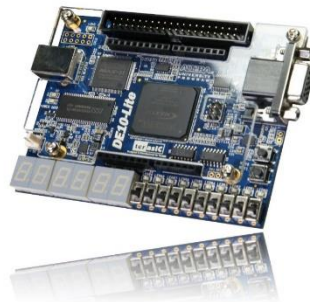
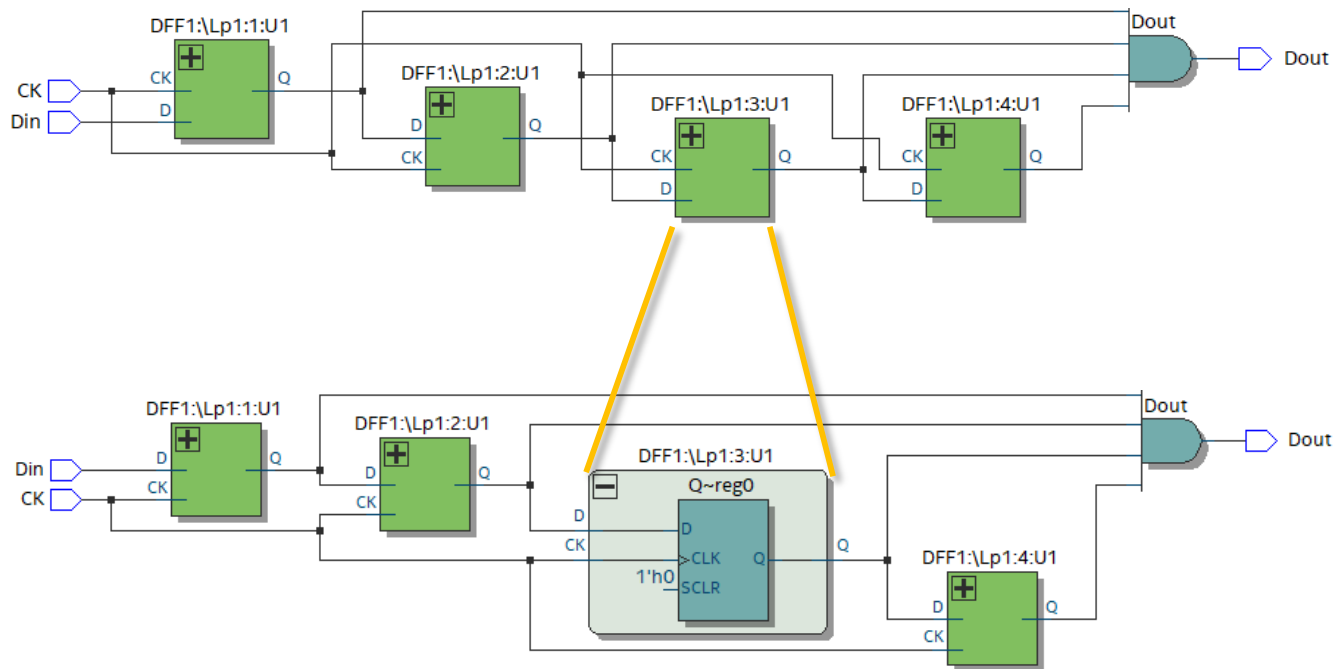
隨堂練習(一)

- 累積時脈延時電路(以DFF為基礎設計電路)並觀察RTL



隨堂練習(一)

- 累積時脈延時電路(以DFF為基礎設計電路)並觀察RTL



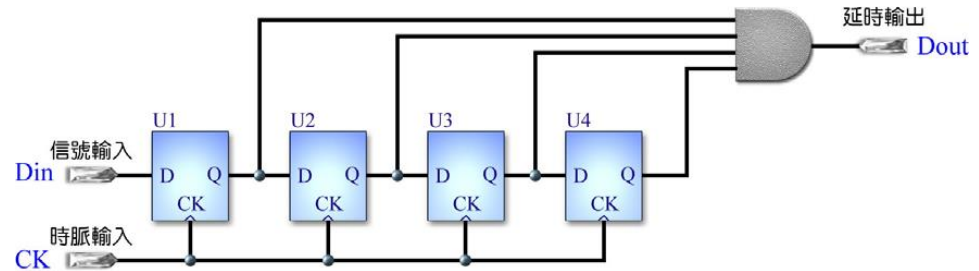
隨堂練習(二)

- 累積時脈延時電路(使用VHDL直接描述DFF電路)並觀察RTL

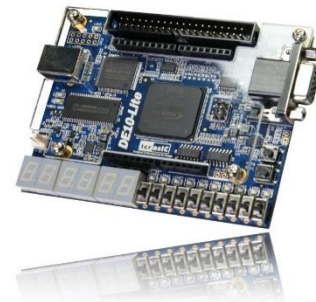
```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity Debouncing2 is
Port( Din,CK:in std_logic;
      Dout:out std_logic);
End Debouncing2;

Architecture ARCH of Debouncing2 is
Begin
  Process(CK)
    Variable TMP:integer range 0 to 9;
  Begin
    If Rising_Edge(CK) Then
      If Din = '1' Then
        TMP := 0;
        Dout <= '1';
      Else
        TMP := TMP + 1;
        If TMP = 4 Then
          TMP := 0;
          Dout <= '0';
        End If;
      End If;
    End If;
  End Process;
End ARCH;
```

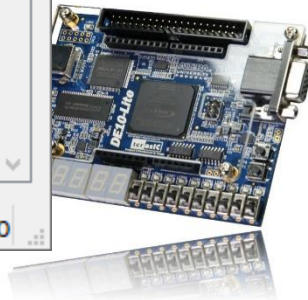
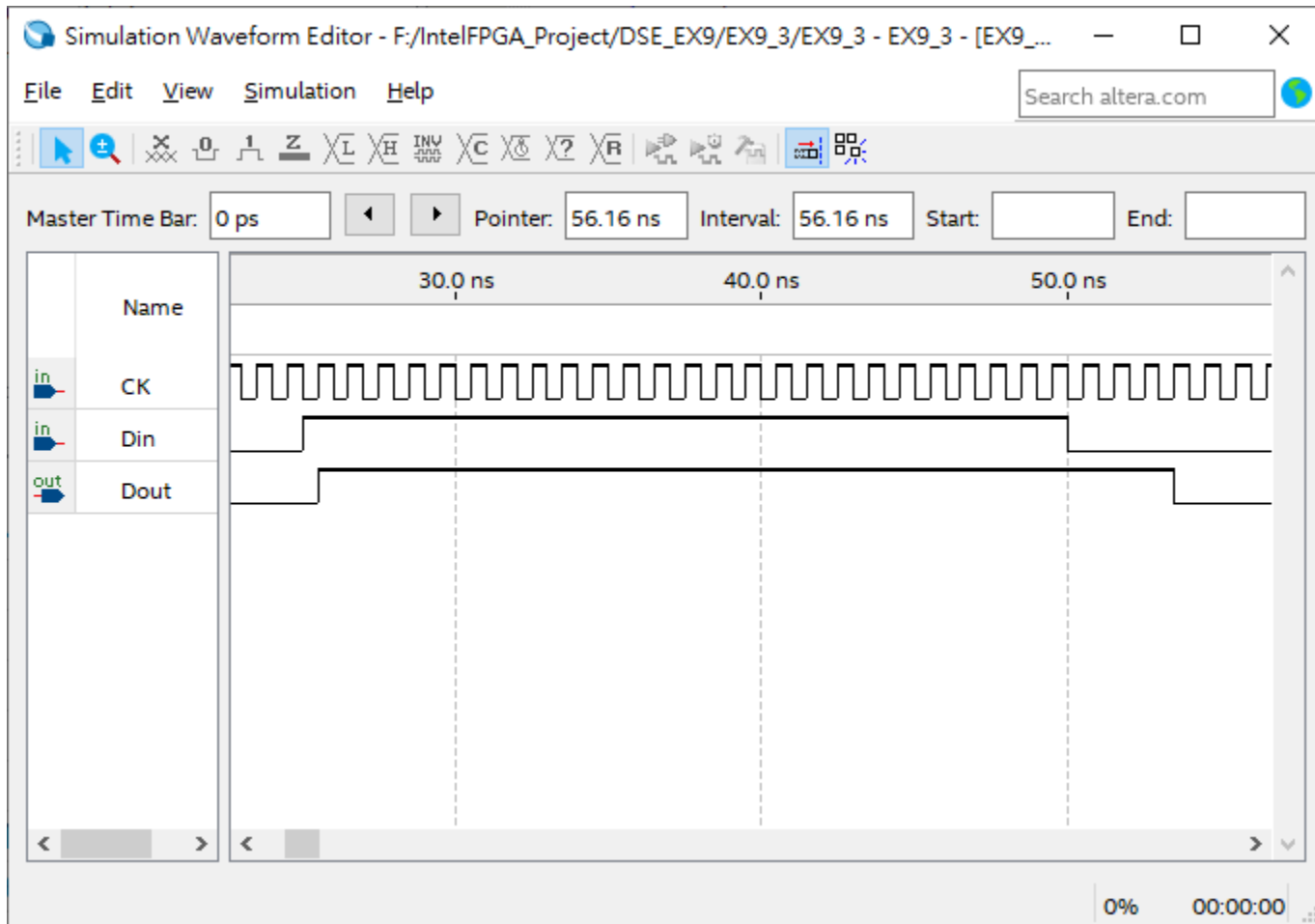


PR	CL	CK	D	Q	\bar{Q}
0	1	-	-	1	0
1	0	-	-	0	1
1	1	↑	0	0	1
1	1	↑	1	1	0



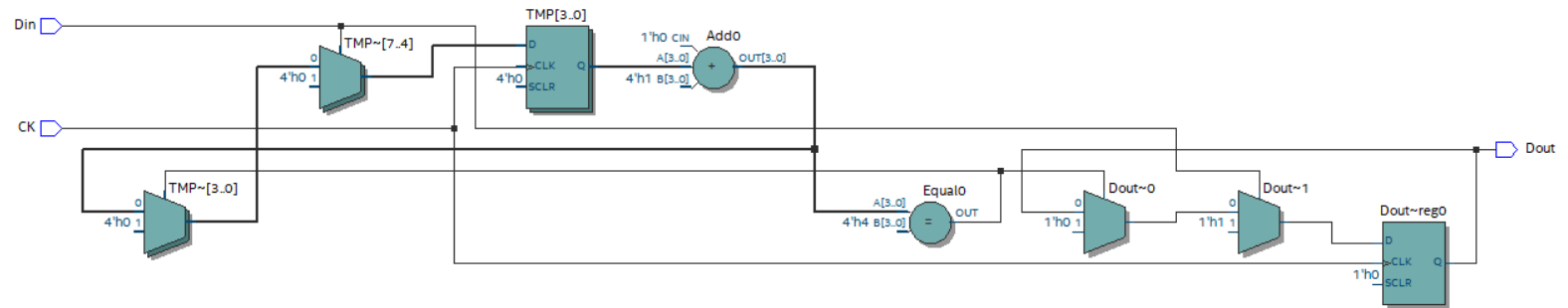
隨堂練習(二)

- 累積時脈延時電路(使用VHDL直接描述DFF電路)並觀察RTL



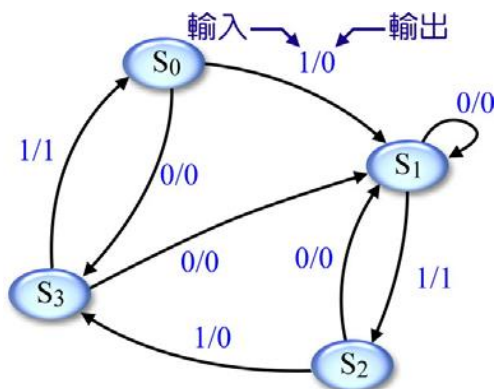
隨堂練習(二)

- 累積時脈延時電路(使用VHDL直接描述DFF電路)並觀察RTL



隨堂練習(三)

• 米利機電路設計(1/2)



右圖中包括**S0~S3**等4個狀態，如下說明：

1. **S0**狀態時，若輸入1，則輸出0且跳至**S1**狀態。
若輸入0，則輸出0且跳至**S3**。
2. **S1**狀態時，若輸入1，則輸出1且跳至**S2**狀態。
若輸入0，則輸出0且維持在**S1**狀態。
3. **S2**狀態時，若輸入1，則輸出0且跳至**S3**狀態。
若輸入0，則輸出0且跳回**S1**狀態。
4. **S3**狀態時，若輸入1，則輸出1且跳至**S0**狀態。
若輸入0，則輸出0且跳至**S1**狀態。

```
Library IEEE;
Use IEEE.std_logic_1164.all;

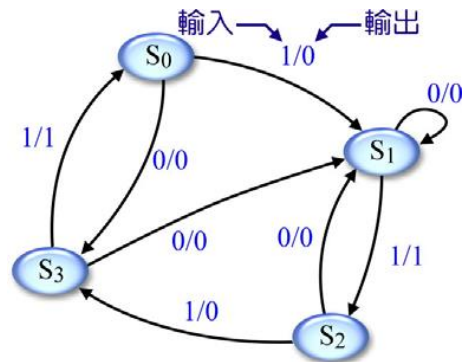
Entity Mealy_Ex is
    Port( Input,CK:in std_logic;
          PState:out integer range 0 to 9; -- 觀察用
          Output:out std_logic);
End Mealy_Ex;

Architecture FSM of Mealy_Ex is
    Type states is (S0, S1, S2, S3);
    Signal PS: states:=S0;
Begin
    Process(CK)
        Variable NS: states:=S0;
    Begin
        If Rising_Edge(CK) Then -- 以CK時脈之上升緣同步觸發
            Case PS is
                when S0 => -- S0狀態的處理
                    If Input='1' Then
                        NS := S1;
                        PState <= 1;
                        Output <= '0';
                    Else
                        NS := S3;
                        PState <= 3;
                        Output <= '0';
                    End If;
                when S1 => -- S1狀態的處理
                    If Input='1' Then
                        NS := S2;
                        PState <= 2;
                        Output <= '1';
                    Else
                        NS := S1;
                        PState <= 1;
                        Output <= '0';
                    End If;
                when S2 => -- S2狀態的處理
                    If Input='1' Then
                        NS := S3;
                        PState <= 3;
                        Output <= '0';
                    Else
                        NS := S1;
                        PState <= 1;
                        Output <= '0';
                    End If;
                when S3 => -- S3狀態的處理
                    If Input='1' Then
                        NS := S0;
                        PState <= 0;
                        Output <= '1';
                    Else
                        NS := S1;
                        PState <= 3;
                        Output <= '0';
                    End If;
                when others => null;
            End Case;
            End If;
            PS <= NS;
        End Process;
    end FSM;
```

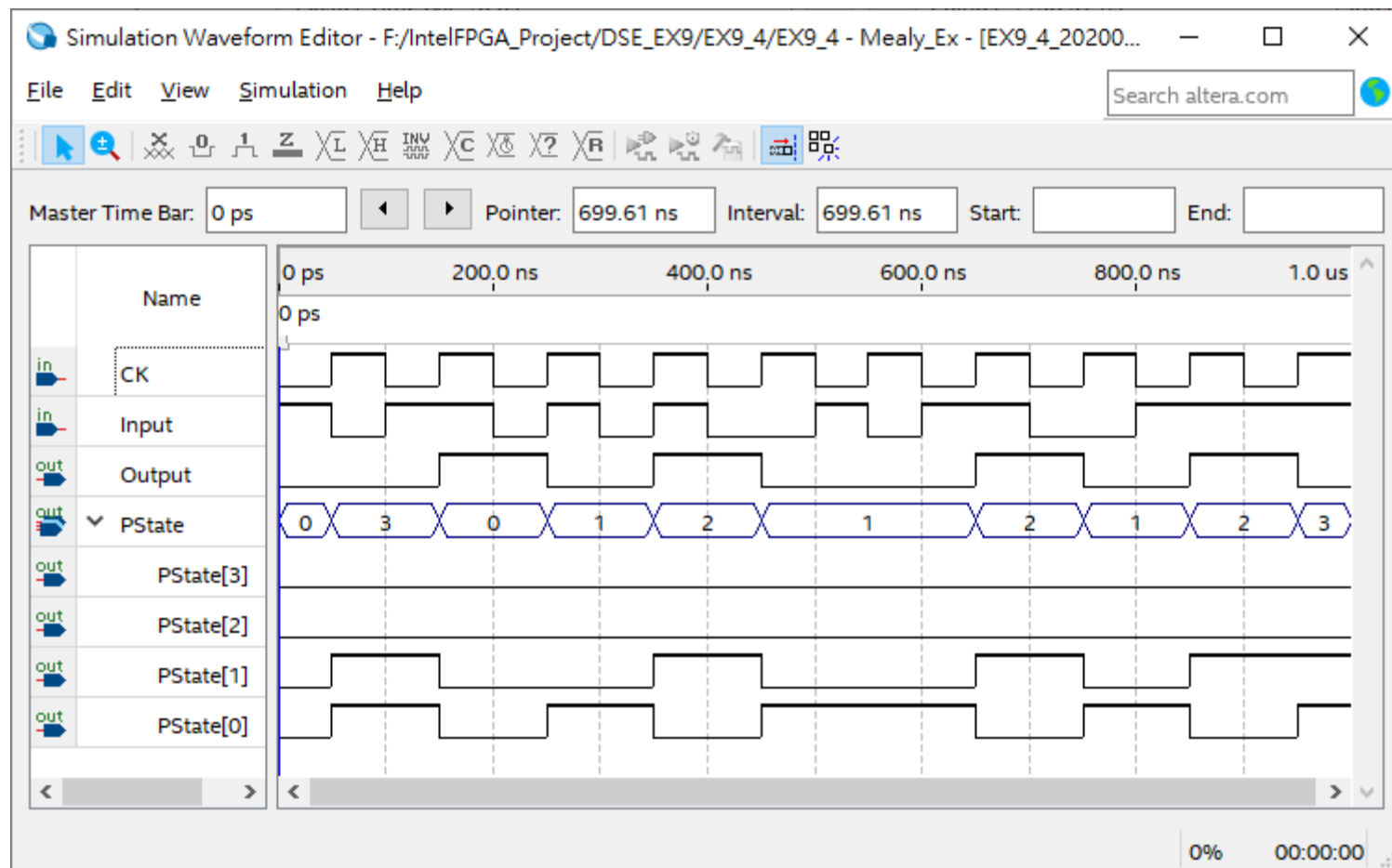


隨堂練習(三)

- 米利機電路設計(2/2)

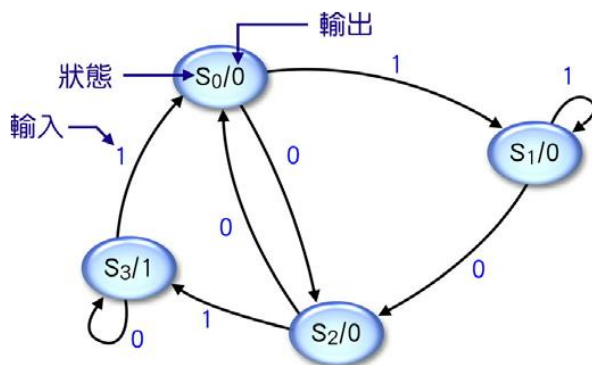


End Time: 1us
Grid Size: 100ns
CK週期: 100ns
Input: Every half grid interval



隨堂練習(四)

• 莫爾機電路設計(1/2)



右圖中包括S0~S3等4個狀態，如下說明：

1. S0狀態時輸出0，若輸入1，則跳至S1狀態。
若輸入0，則跳至S2狀態。
2. S1狀態時輸出0，若輸入1，則保持在S1狀態。
若輸入0，則跳至S2狀態。
3. S2狀態時輸出0，若輸入1，則跳至S3狀態。
若輸入0，則跳回S0狀態。
4. S3狀態時輸出1，若輸入1，則跳至S0狀態。
若輸入0，則保持在S3狀態。

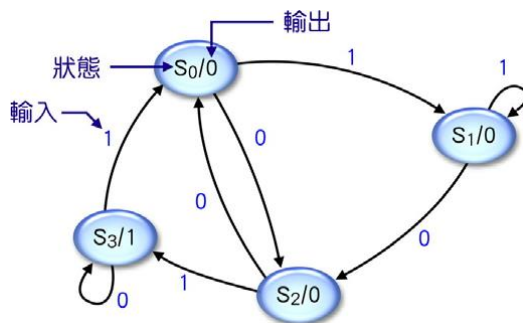
```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity Moore_Ex is
Port( Input,CK:in std_logic;
      Pstate:out integer range 0 to 9; -- 觀察用
      Output:out std_logic);
End Moore_Ex ;

Architecture FSM of Moore_Ex is
Type states is (S0, S1, S2, S3);
Signal PS: states:=S0;
Begin
Output <='1' when PS=S3 Else '0';
Process(CK)
Variable NS: states:=S0;
Begin
If Rising_Edge(CK) Then -- 以CK時脈之升緣同步觸發
Case PS is
when S0 => -- S0狀態的處理
If Input='1' Then
NS := S1;
Pstate <= 1;
Else
NS := S2;
Pstate <= 2;
End If;
when S1 => -- S1狀態的處理
If Input='1' Then
NS := S1;
Pstate <= 1;
Else
NS := S2;
Pstate <= 2;
End If;
when S2 => -- S2狀態的處理
If Input='1' Then
NS := S3;
Pstate <= 3;
Else
NS := S0;
Pstate <= 0;
End If;
when S3 => -- S3狀態的處理
If Input='1' Then
NS := S0;
Pstate <= 0;
Else
NS := S3;
Pstate <= 3;
End If;
when others => null;
End Case;
End If;
PS <= NS;
End Process;
End FSM;
```



隨堂練習(四)

- 莫爾機電路設計(2/2)



End Time: 1us
Grid Size: 100ns
CK週期: 100ns
Input: Every half grid interval

