

數位系統導論實驗

# Lab 6

## Structural Modeling CLA

負責助教：蔡帛洋

## 課程目的

在前一次的 lab 中，同學們已練習過以 Verilog (structural modeling & behavioral modeling) 來描述硬體設計。在本次 lab 中同學們將跟隨範例練習以 structural modeling 完成 16-bit CLA (carry lookahead adder) 的實做，並自行完成 64-bit CLA 的課堂作業。

# Outline

- 課程內容與範例
  - 16-bit CLA w/ structural modeling
- 課堂作業
- 課堂教材
- 評分方式
- 其他事項

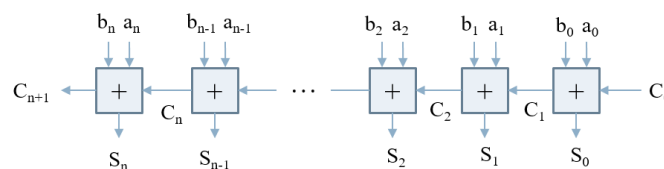
課程內容&範例

# 16-bit CLA w/ Structural Modeling

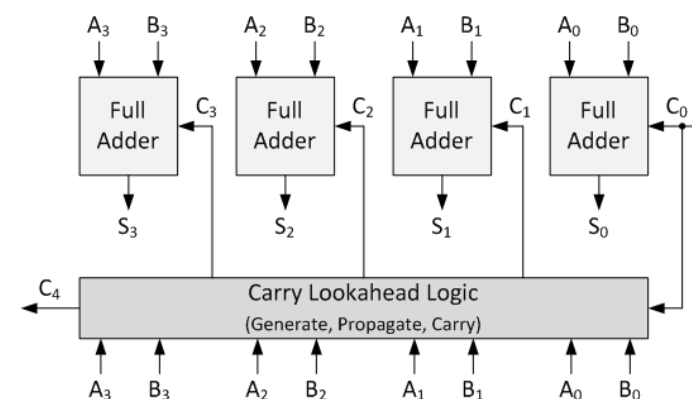
## 16-bit CLA w/ Structural Modeling

# Carry Lookahead Adder (CLA)

由於 RCA 每個 FA 都需等待前面的 carry-out (如圖a)，執行效率較差，因此有人提出了CLA 的概念，如下圖b所示，以較複雜的電路，預先產生所有的 carry-out，減少 RCA 中 carry-out 傳遞造成的電路延遲，是一種較有效率的加法器。此類加法器由於預先計算了每一個 FA 的 carry-out，因此被稱為 CLA (carry lookahead adder)



(a) RCA 結構示意圖



(b) CLA 結構示意圖

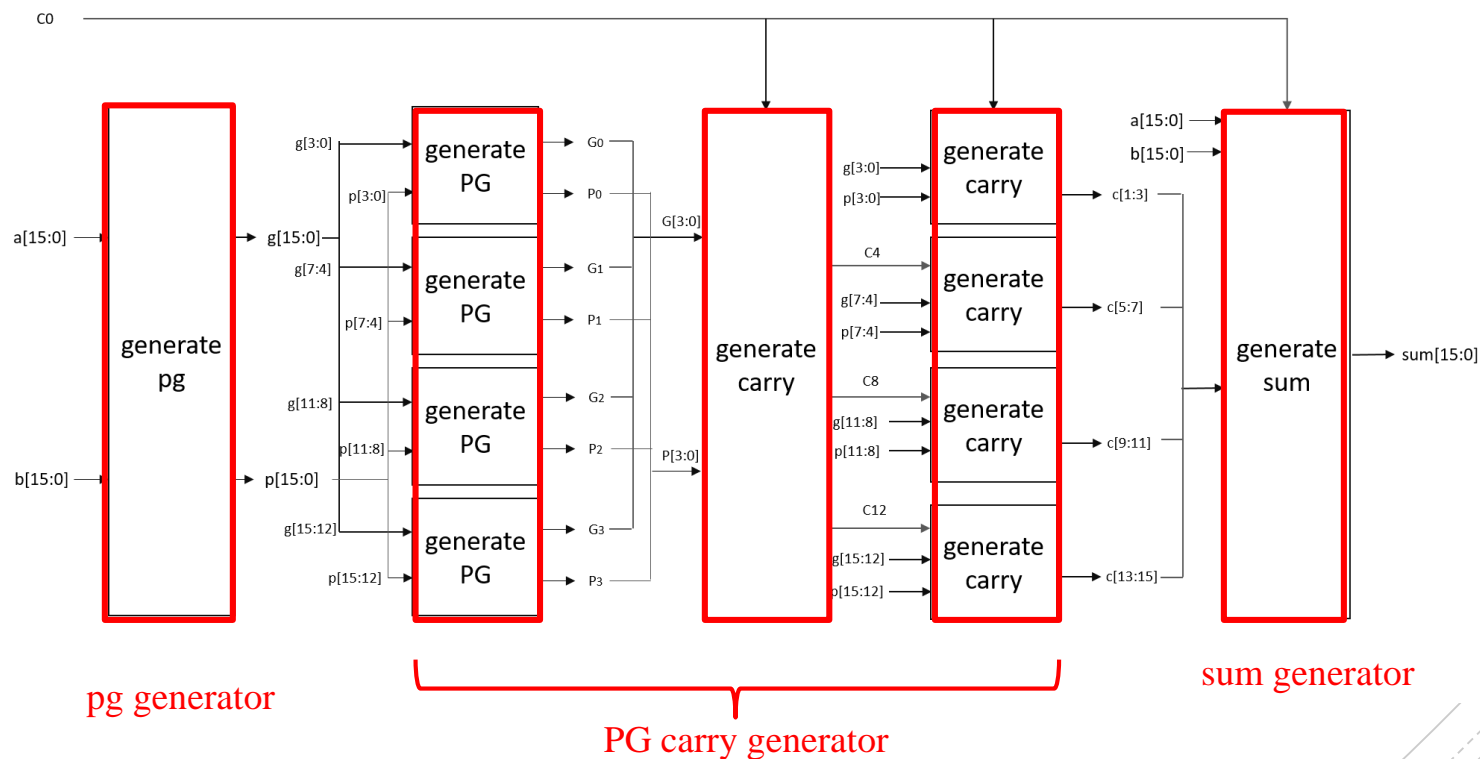
## 16-bit CLA w/ Structural Modeling

# 16-bit CLA Structural Modeling

下圖以 16-bit CLA ( $\text{fan-in} \leq 4$ ) 為例，架構主要可分為三個子模組

1. pg generator：此模組用來產生 p 與 g
2. PG carry generator：此模組用來產生 carry、group P 與 group G
3. sum generator：此模組用來產生 sum

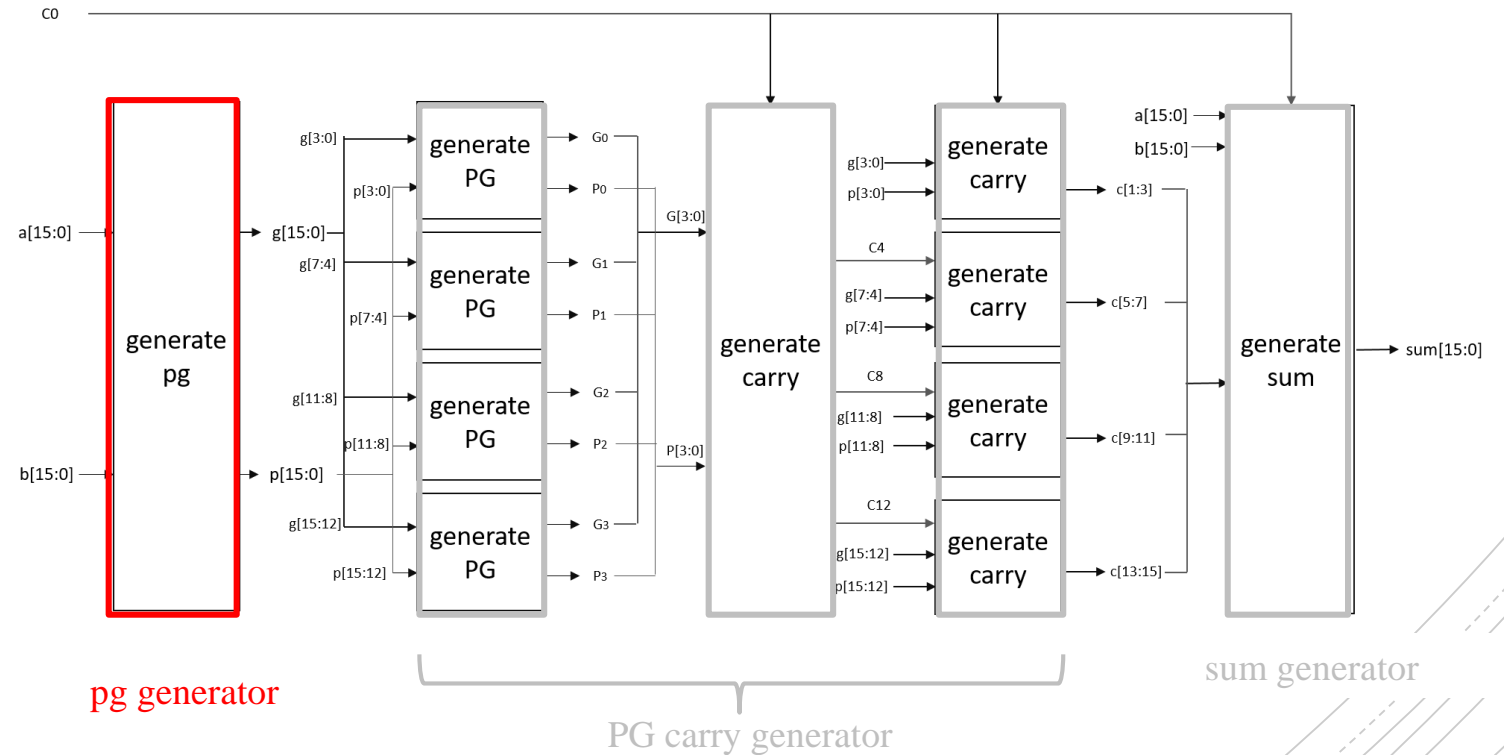
以下就三個子模組分別做介紹



## 16-bit CLA w/ Structural Modeling

### Sub-module : pg Generator (1/3)

```
module pg_generator( a, b, p, g);  
  
    input [15:0] a, b;  
    output [15:0] p, g;  
  
    // Carry Propagate  
    assign p = a | b;    //p = a + b  
    assign g = a & b ;   //g = a * b  
  
endmodule
```



## 16-bit CLA w/ Structural Modeling

### Sub-module : PG Carry Generator (2/3)

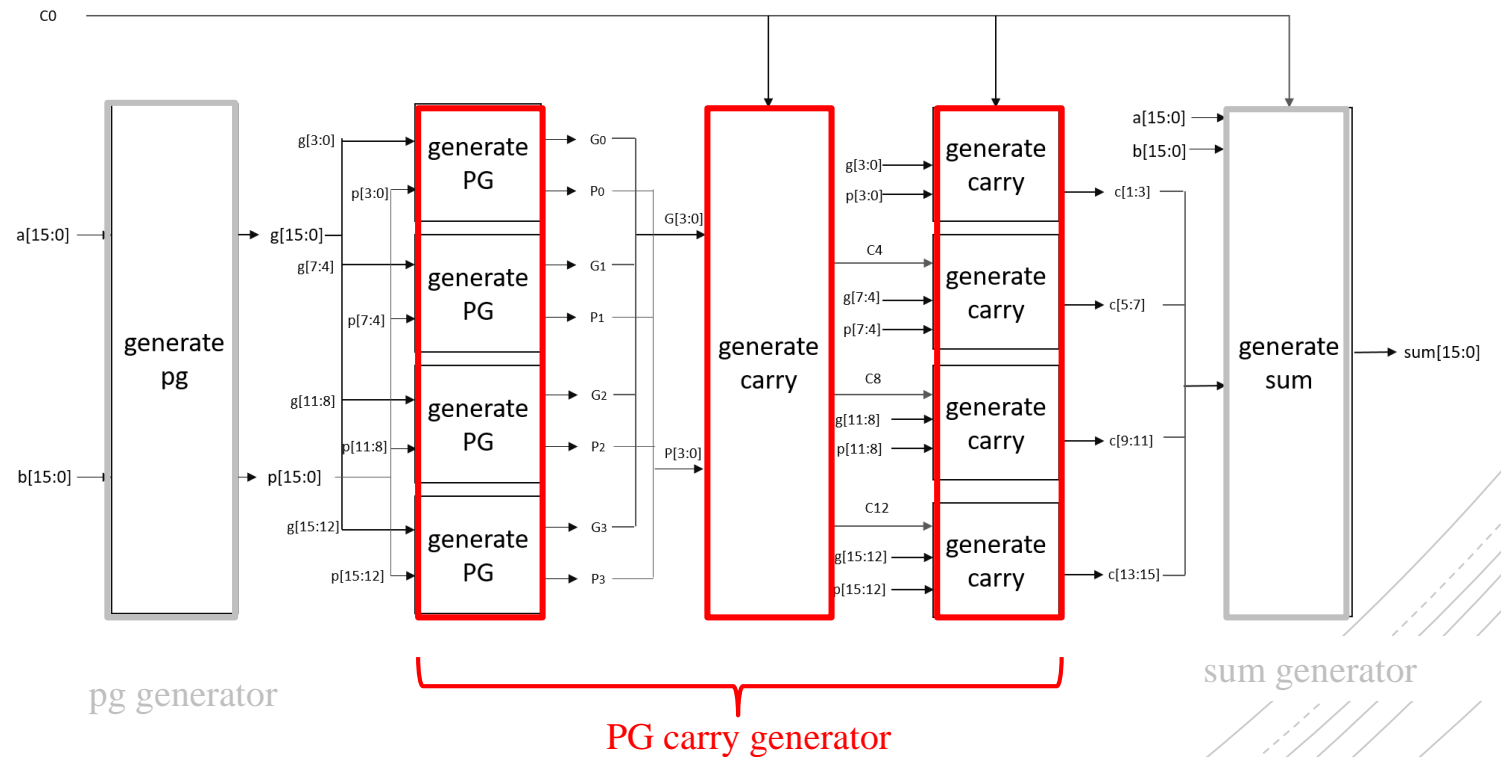
```
module PG_carry_generator( p, g,cin, gG, gP,c);

    input [3:0] p, g;
    input cin;
    output gG, gP;
    output [3:1] c;

    assign gG = (g[0] & p[1] & p[2] & p[3])|(g[1]&p[2]&p[3])|(g[2]&p[3])|g[3];
    assign gP = p[0] & p[1] & p[2] & p[3];

    assign c[1] = g[0]|(p[0] & cin);
    assign c[2] = g[1]|(p[1] & g[0])|(p[1] & p[0] & cin);
    assign c[3] = g[2]|(p[2] & g[1])|(p[2] & p[1] & g[0])|(p[2] & p[1] & p[0] & cin);

endmodule
```

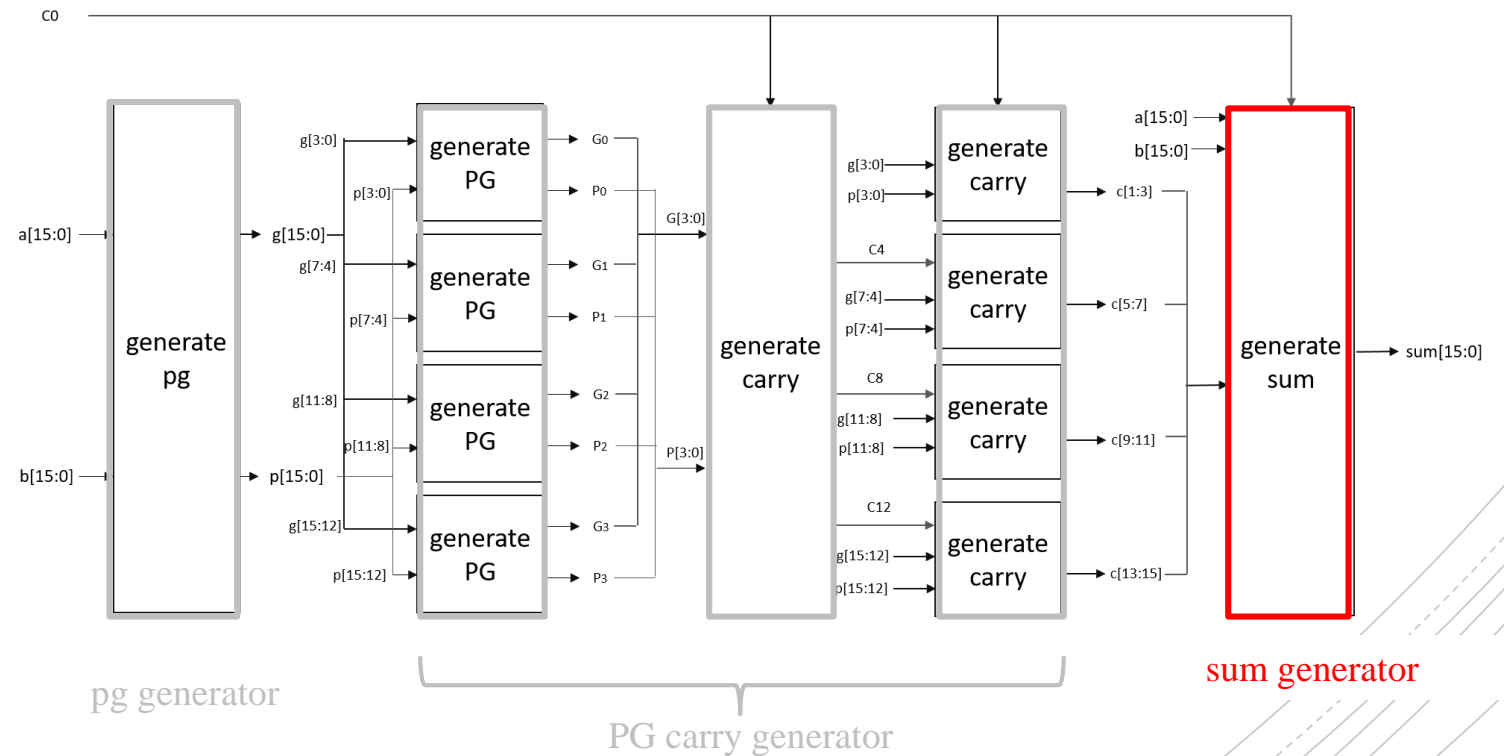




## 16-bit CLA w/ Structural Modeling

### Sub-module : Sum Generator (3/3)

```
module sum_generator( a,b,cin,c, sum);  
  
    input [15:0] a, b;  
    input cin;  
    input [15:1] c;  
    output [15:0] sum;  
  
    assign sum = a^ b^{c,cin};  
  
endmodule
```



## 16-bit CLA w/ Structural Modeling

# 16-bit CLA Structural Modeling

下圖為 16-bit CLA Verilog code，圖中邏輯區段的部分以 structural modeling 方式來完成

```
module cla_16bit( a, b, cin, sum);
```

I/O definition

```
    input [15:0] a, b;  
    input cin;  
    output [15:0] sum;  
    output cout;
```

```
    wire [3:0] G, P;  
    wire [15:0] p, g;  
    wire [16:0] c;
```

Variable declaration

```
    //generate p & g  
    pg_generator pg( a[15:0], b[15:0],p[15:0], g[15:0]);  
  
    //generate group g & p  
    PG_carry_generator PG1( p[3:0], g[3:0], ,G[0], P[0],);  
    PG_carry_generator PG2( p[7:4], g[7:4], ,G[1], P[1],);  
    PG_carry_generator PG3( p[11:8], g[11:8], ,G[2], P[2],);  
    PG_carry_generator PG4( p[15:12], g[15:12], ,G[3], P[3],);  
  
    //generate c4, c8 & c12  
    PG_carry_generator carry_c4x1(P[3:0], G[3:0], cin,, {c[12],c[8],c[4]}); //c4,c8,c12  
  
    //generate carry  
    PG_carry_generator carry1(p[3:0], g[3:0], cin, ,, c[3:1]); //c1,c2,c3  
    PG_carry_generator carry2(p[7:4], g[7:4], c[4], ,, c[7:5]); //c5,c6,c7  
    PG_carry_generator carry3(p[11:8], g[11:8], c[8], , ,c[11:9]); //c9,c10,c11  
    PG_carry_generator carry4(p[15:12], g[15:12], c[12], ,, c[15:13]); //c13,c14,c15  
  
    //generate sum  
    sum_generator sum1( a[15:0], b[15:0], cin, c[15:1], sum[15:0]);
```

Structural modeling

```
endmodule
```

16-bit CLA w/ Structural Modeling

## 16-bit CLA 驗證

完成16-bit CLA模組後，透過 testbench 進行行為驗證，確認模組運算結果是否正確

```
PS D:\Sheep\course\Master\助教\DDLAB2019\DDLAb6\DDLAb6_2019ans> iverilog -o CLA_16bit .\testbench_16bit.v
PS D:\Sheep\course\Master\助教\DDLAB2019\DDLAb6\DDLAb6_2019ans> vvp .\CLA_16bit
VCD info: dumpfile lab6_16bit.fsdb opened for output.
Test 1
// Successfull 1//
13604 + 24470 + 1= ?
sum = 38075

Test 2
// Successfull 2//
22115 + 2176 + 1= ?
sum = 24292

Test 3
// Successfull 3//
33893 + 25730 + 1= ?
sum = 59624

Test 4
// Successfull 4//
52493 + 26558 + 1= ?
sum = 13516





Test 5
// Successfull 5//
22509 + 1001 + 1= ?
sum = 23511
```

# 課程作業

依照課程內容以 structural modeling 完成 16-bit CLA 後，同學需自行修改程式碼完成 64-bit CLA ( $\text{fan-in} \leq 4$ )，並根據提供的 testbench 完成驗證

# 課程教材

本次教材會提供以下四個檔案，CLA\_16bit.v 是 16-bit CLA 課堂範例的程式碼，並使用 testbench\_16bit.v 進行驗證。同學需自行修改 CLA\_64bit.v 完成 64-bit CLA 的實作，並用 testbench\_64bit.v 進行驗證。

名稱	修改日期	類型
 CLA_16bit	2019/4/14 下午 1...	V 檔案
 CLA_64bit	2019/4/14 下午 1...	V 檔案
 testbench_16bit	2019/4/14 下午 1...	V 檔案
 testbench_64bit	2019/4/14 下午 1...	V 檔案

# 評分方式

- 完成課程範例 30%
- 完成課堂練習並於 demo 時間展示 45%
- 完成 demo 時的隨堂測驗 25%

## 其他事項

同學們若對本次 lab 的範例、作業或者評分方式有任何疑問，請寄信到負責本次 lab 的助教信箱詢問，謝謝

- 蔡帛洋 [ianpaul32@gmail.com](mailto:ianpaul32@gmail.com)

The background features a series of concentric circles in light gray and dashed lines. A solid blue speech bubble is centered on the page, pointing downwards.

# 附錄



附錄

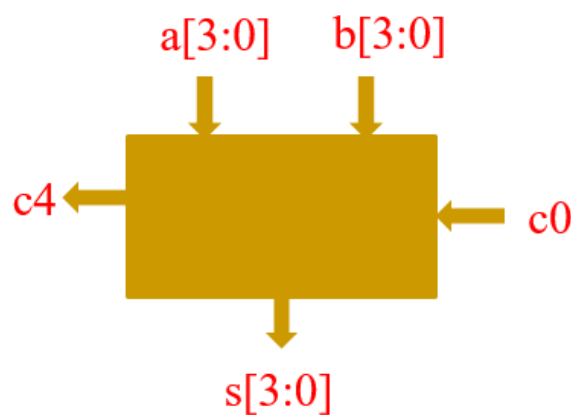
# Introduction to Verilog Modeling

# Verilog Overview

- Verilog was originally designed for simulating logic circuits (in C-like language) before implementation; it was then used for design capture at high abstraction level
- First step of hardware design is to define its I/O, which can be easily mapped into Verilog as a “shell” (i.e. from module to endmodule)
- Detailed hardware design can be described in behavioral or structural Verilog
- Even higher-level descriptions can be used to “test” the designed hardware (i.e. testbench)
- Implementation of Verilog model (logic synthesis) is out of scope of this lab

# I/O Definition

描述硬體設計的第一步便是定義 I/O，如下圖是以一個 4-bit 的加法器為例，說明如何以 Verilog 將左方的硬體 I/O 描述出來。在定義完 I/O 之後，便可在下圖黃色區塊描述硬體的行為 (behavior) 及結構 (structure)



```
module adder(a, b, c0, s, c4);  
  input [3:0] a, b;  
  input c0;  
  output [3:0] s;  
  output c4;
```

I/O

**Describe behavior or  
structure of design**

```
endmodule
```

Introduction to Verilog Modeling

# Verilog Modeling

在定義完硬體的 I/O 後，便可開始描述電路的行為與結構。Verilog 描述電路的方式主要包含 behavioral modeling 和 structural modeling 兩部分，以下就這兩部分做介紹

# Behavioral Modeling

Behavioral modeling 是指以行為描述數位電路，可分為 continuous assignment 和 procedural assignment 兩種：

- Continuous\_assignment: 以 assign 關鍵字描述硬體架構連結，位於 procedural block (i.e. always block) 外，等式的左邊必須是 wire 的型態
- Procedural\_assignment: 位於 procedure block (i.e. always block) 內，賦值於型態為 reg、integer、時間變數等，不可賦值於 wire 型態的變數

## Continuous Assignment

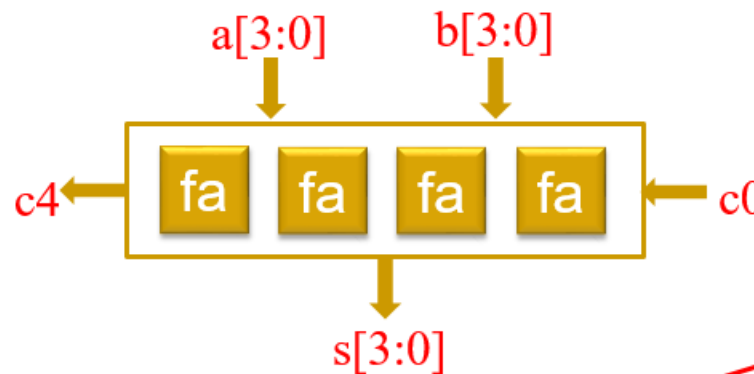
```
module adder(a, b, c0, s, c4);  
    input [3:0] a, b;  
    input c0; output [3:0] s;  
    output c4;  
  
    assign {c4,s}=a+b+c0;  
  
endmodule
```

## Procedural Assignment

```
module adder(a, b, c0, s, c4);  
    input [3:0] a, b;  
    input c0;  
    output [3:0] s;  
    output c4;  
  
    reg [3:0] s;  
    reg c4;  
  
    always@(a or b or c0)  
        {c4,s}=a+b+c0;  
  
endmodule
```

# Structural Modeling

Structural modeling 是以邏輯閘、預定義模組及模組與模組間連結方式來描述數位電路的結構。下圖便是如何以四個 1-bit 的全加器 (full adder; FA)，透過描述 FA 與 FA 間的接線方式，組合出一個 4-bit 的加法器



```
module fa(a, b, ci, s, co);
input a, b, ci;
output s, co;
assign {co,s}=a+b+ci;
endmodule
```

以 Verilog 撰寫之 1-bit 全加器模組

```
module adder(a, b, c0, s, c4);
input [3:0] a, b;
input c0;
output [3:0] s;
output c4;
wire c1, c2, c3;
```

```
fa bit0(a[0], b[0], c0, s[0], c1);
fa bit1(a[1], b[1], c1, s[1], c2);
fa bit2(a[2], b[2], c2, s[2], c3);
fa bit3(a[3], b[3], c3, s[3], c4);
endmodule
```

以 structural modeling 將四個 1-bit 的 FA 連接在一起，組合出一個 4-bit 的加法器

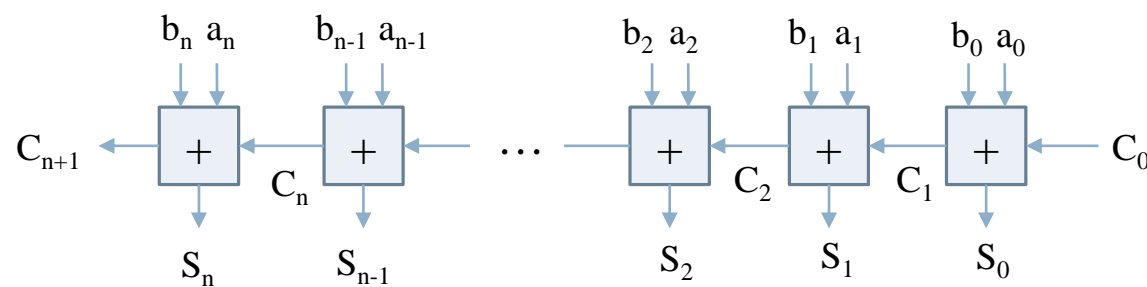
附錄

16-bit RCA  
w/ Structural Modeling

16-bit RCA w/ Structural Modeling

## Ripple Carry Adder (RCA)

RCA 是由多個 FA 組成的加法器，結構如下圖所示，第  $n$  個 FA 需等待第  $n-1$  個 FA 的 carry-out 傳入才能進行運算，信號如同波紋般依序地向前傳遞，因此這類加法器被稱為波紋進位加法器 (ripple carry adder; RCA)，接著我們將介紹組成 RCA 的基本單位 1-bit FA，並練習以 structural modeling 方式撰寫 16-bit RCA 的 Verilog。



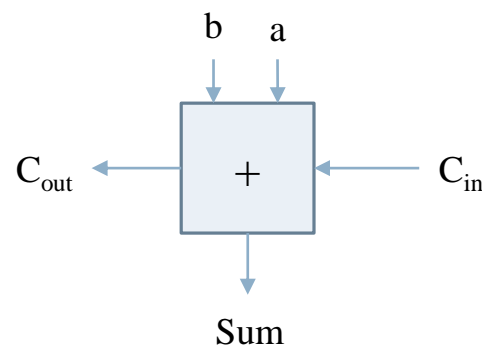
$n$ -bit RCA 結構示意圖



## 16-bit RCA w/ Structural Modeling

### Full Adder (FA)

一個 1-bit 的全加器 (FA) ，如下圖(a) ，是由兩個 1-bit 的 input (a & b)及一個 1-bit 的 carry in ( $C_{in}$ ) 相加，輸出一個 1-bit 的 carry out ( $C_{out}$ ) 及 1-bit 的 sum，輸出訊號的運算邏輯如圖(b)。



(a)

$$\begin{aligned} Sum &= a \oplus b \oplus c_{in} \\ C_{out} &= ab + bc_{in} + ac_{in} \end{aligned}$$

(b)

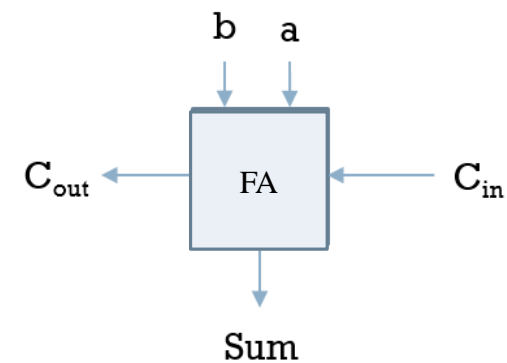
## 16-bit RCA w/ Structural Modeling

# 16-bit RCA Structural Modeling

在了解整體架構後，我們將從撰寫 FA 的 Verilog 模組開始。如圖(a)所示，首先，宣告一個 FA 的模組，並定義此模組的輸入及輸出 (a、b 與 cin 為模組的輸入，sum 和 cout 為模組的輸出)。接著我們便可開始依照圖(b)的結構圖描述模組的行為及架構，完成 FA 模組的撰寫。

```
module FA (a, b, cin, s, cout);  
  
    input a, b, cin;  
    output s, cout;  
    wire s1,c1,c2 ;  
  
    xor (s1, a,b);  
    xor (s, s1, cin);  
    and (c1, a,b);  
    and (c2, s1, cin);  
    or  (cout, c2, c1);  
  
endmodule
```

(a)



$$\begin{aligned} \text{Sum} &= a \oplus b \oplus c_{in} \\ C_{out} &= ab + bc_{in} + ac_{in} \end{aligned}$$

(b)

## 16-bit RCA w/ Structural Modeling

# 16-bit RCA Structural Modeling

在有了 FA 模組後，我們便可利用此模組完成16-bit 的 RCA，透過宣告 wire 型態的變數把數個 FA 模組連結在一起，如下圖示：

```
module RCA_16bit(a, b, cin, sum, cout);  
  
    input [15:0] a, b;  
    input cin;  
    output cout;  
    wire [15:0] sum;  
    wire c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15;  
  
    // structural modeling  
    FA fa0(a[0], b[0], cin, sum[0], c1);  
    FA fa1(a[1], b[1], c1, sum[1], c2);  
    FA fa2(a[2], b[2], c2, sum[2], c3);  
    FA fa3(a[3], b[3], c3, sum[3], c4);  
    FA fa4(a[4], b[4], c4, sum[4], c5);  
    FA fa5(a[5], b[5], c5, sum[5], c6);  
    FA fa6(a[6], b[6], c6, sum[6], c7);  
    FA fa7(a[7], b[7], c7, sum[7], c8);  
    FA fa8(a[8], b[8], c8, sum[8], c9);  
    FA fa9(a[9], b[9], c9, sum[9], c10);  
    FA fa10(a[10], b[10], c10, sum[10], c11);  
    FA fa11(a[11], b[11], c11, sum[11], c12);  
    FA fa12(a[12], b[12], c12, sum[12], c13);  
    FA fa13(a[13], b[13], c13, sum[13], c14);  
    FA fa14(a[14], b[14], c14, sum[14], c15);  
    FA fa15(a[15], b[15], c15, sum[15], cout);  
  
endmodule
```

