Verilog HDL Simulation

LAB 2 - 硬體描述語言模擬及驗證

負責助教：陳憲億、黃琮閔、鄭東昇、趙文駿、陳韋廷

Outline

Verilog HDL Simulation

- Review Data type 、Data assignment
- Equivalence checking
- Simulating framework
- Example
- Waveform viewer
- Observation of circuit wave
- Homework
- Demo

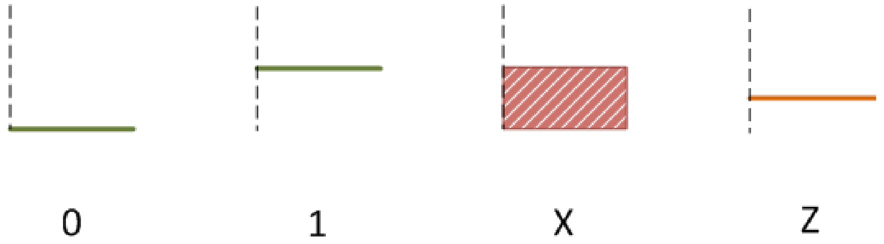
Review - Data type

- variables 的 data types 具有以下四種狀態

0	represents a logic zero, or a false condition
1	represents a logic one, or a true condition
x	represents an unknown logic value // It can be 1 or 0, where it means “ don't care ”
z	represents a high-impedance state

下圖是如何在時序圖和 simulation 的波形中表示這些值

大部分 simulation tool 皆是如此，其中紅色代表任意值，橙色代表高阻抗



Review - Data assignment

- **Continuous assignment**
  - Imply that whenever any change on the RHS of the assignment occurs, it is evaluated and assigned to the LHS.

Ex.
```
wire [3:0] a, b, c ;
assign a = b + c ;
```

- **Procedural assignment**
  - Assignment to "register" data types may occur within always, initial, task and function. These expressions are controlled by triggers which cause the assignment to evaluate.

Ex.
```
reg a, clk ;
Always #5 clk = ~clk;
```

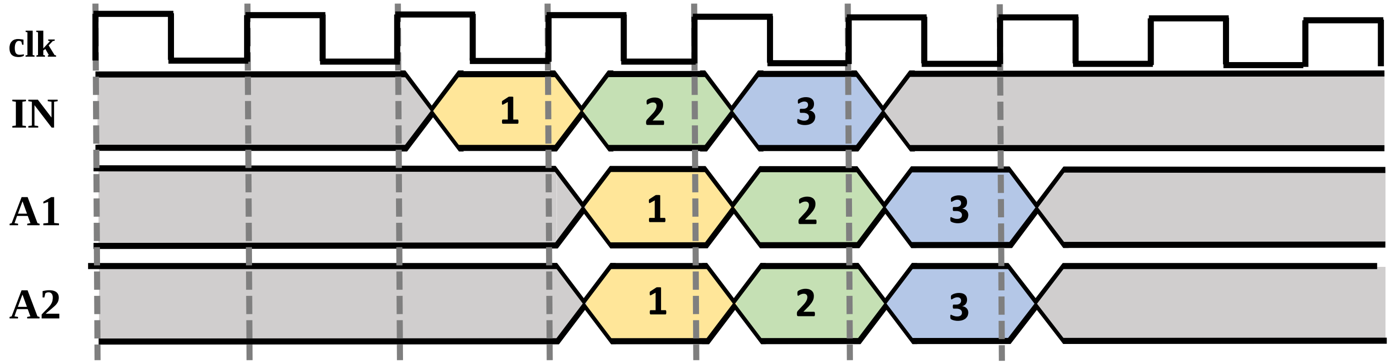
Ex.
```
always @(b)
if  (clk == 1'b1)
a = ~b;
```

Data assignment

- Blocking assignment

  Ex：

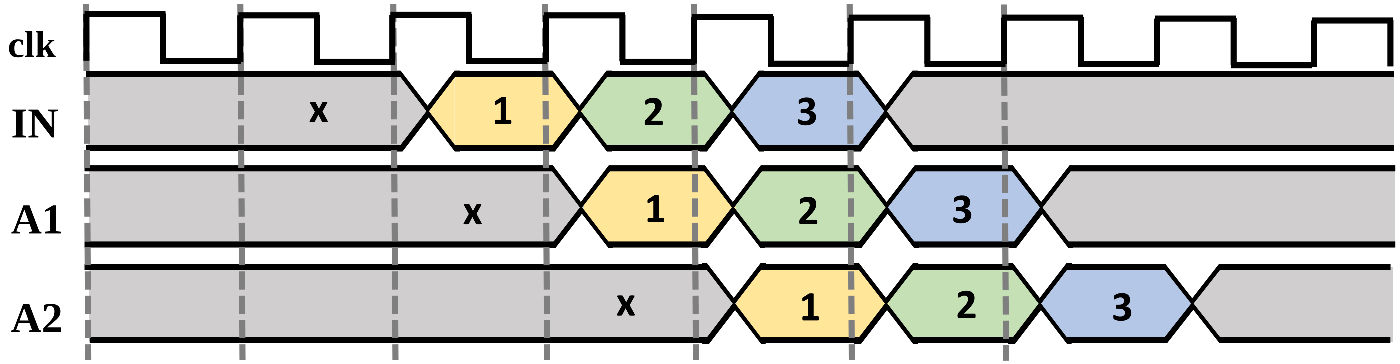
```
always @( posedge clk )
begin
A1 = IN;
A2 = A1;
A3 = A2;
end
```



- Non-Blocking assignment

  Ex：

```
always @( posedge clk )
begin
A1 <= IN;
A2 <= A1;
A3 <= A2;
end
```

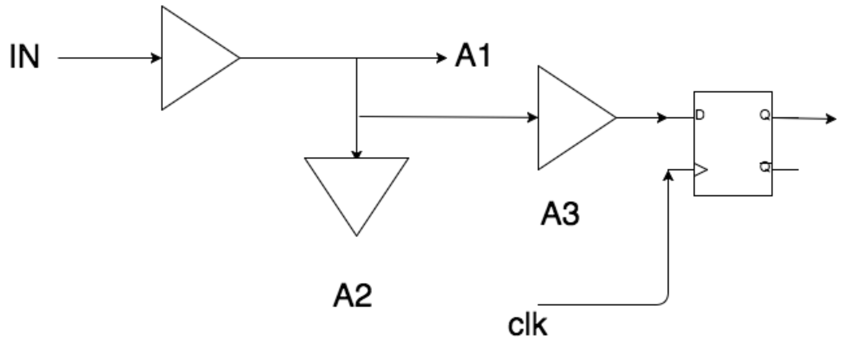


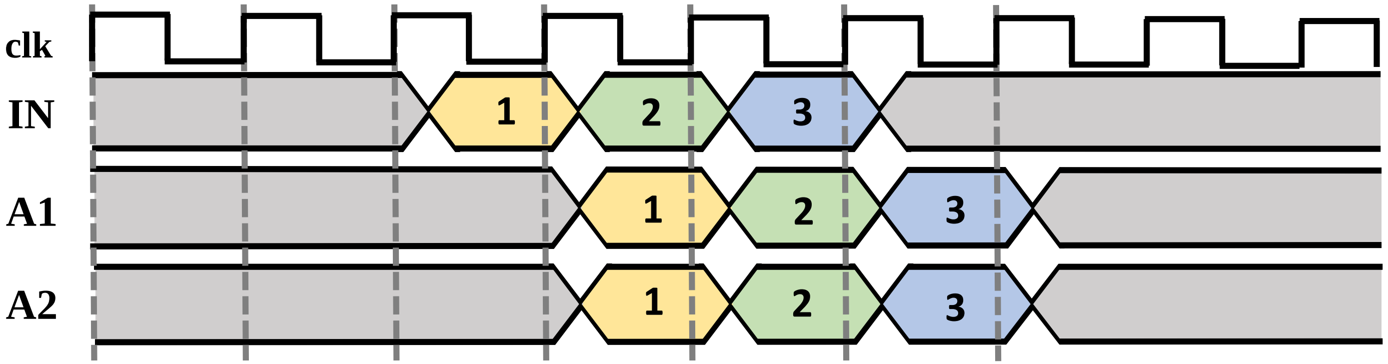
Data assignment

- Blocking assignment

Ex：

```
always @( posedge clk )
begin
A1 = IN;
A2 = A1;
A3 = A2;
end
```



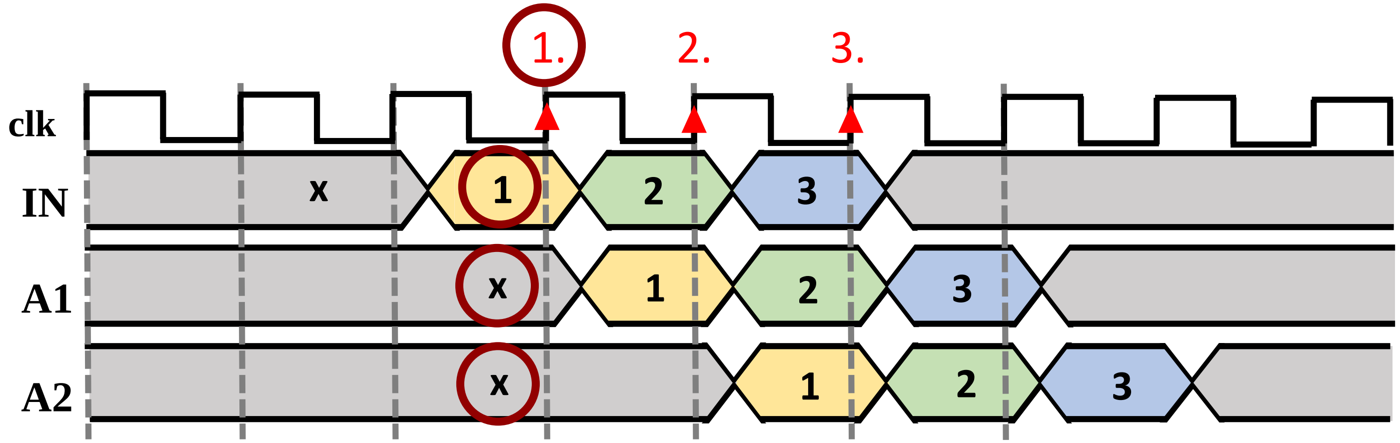


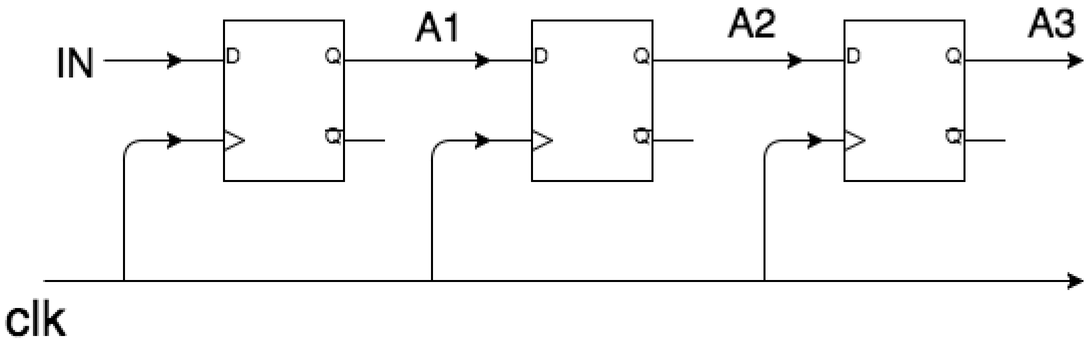
Data assignment

- Non-Blocking assignment

Ex：

```
always @( posedge clk )
begin
A1 <= IN;
A2 <= A1;
A3 <= A2;
end
```



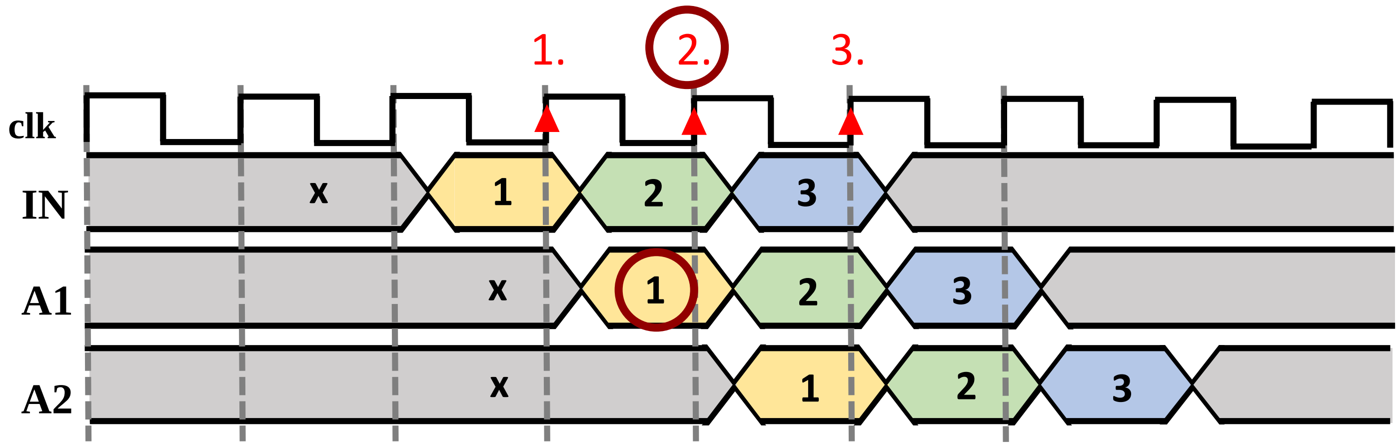


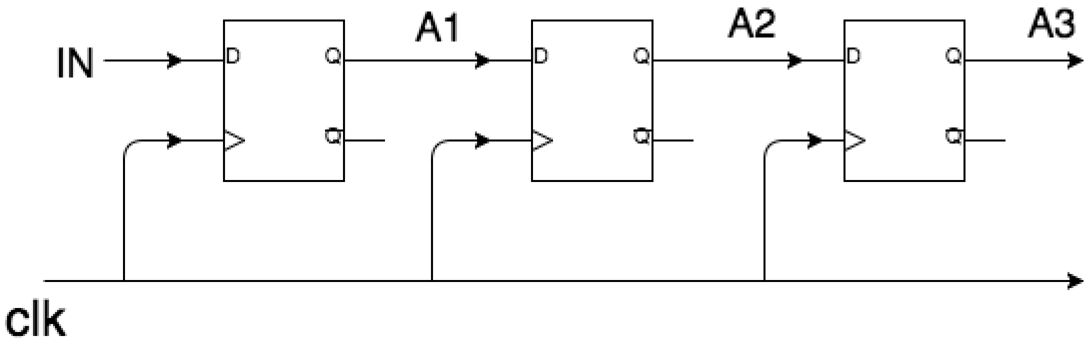
Data assignment

- Non-Blocking assignment

Ex：

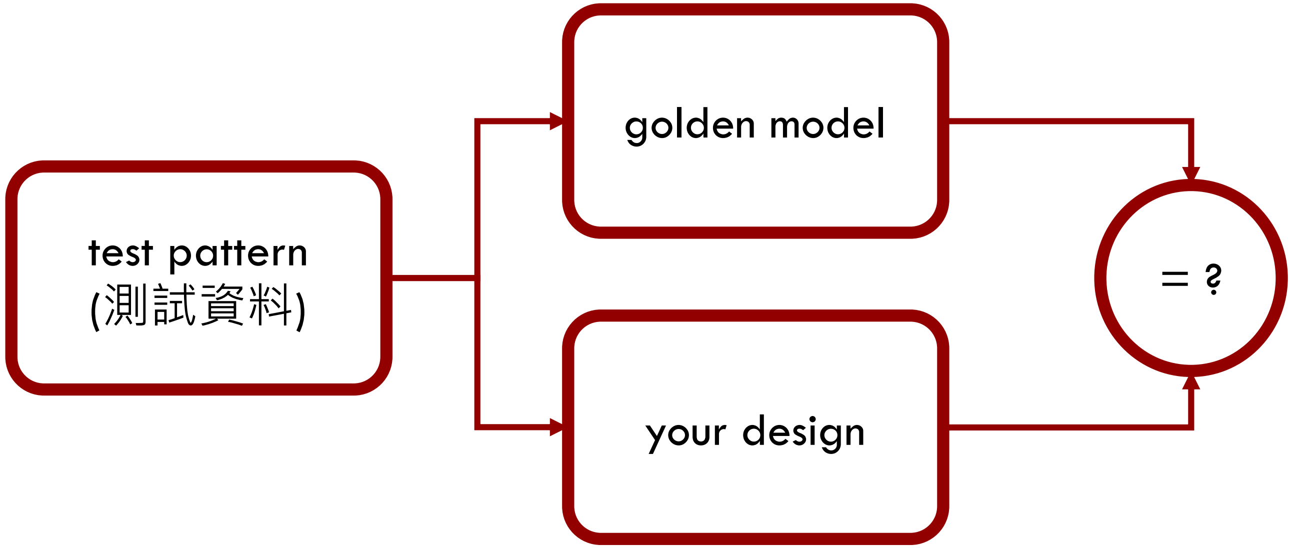
```
always @( posedge clk )
begin
A1 <= IN;
A2 <= A1;
A3 <= A2;
end
```





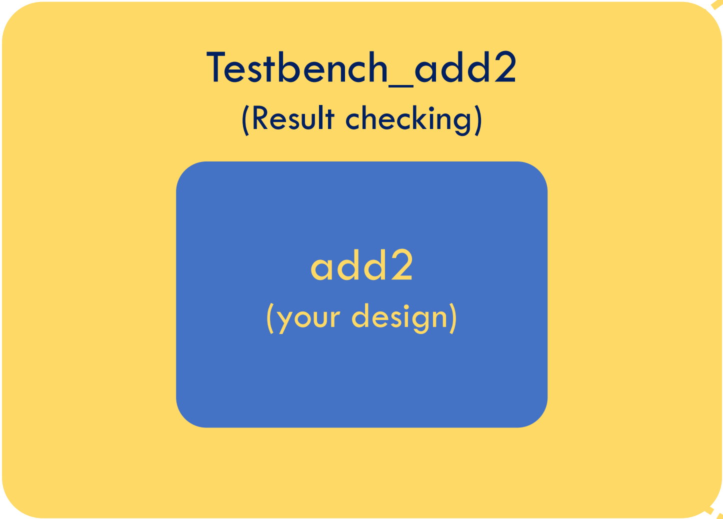
Equivalence checking

- 等效性檢查 (Equivalence Checking, EC)：
  簡單來說，EC 即是確認你的設計是否與標準樣本 (golden model) 有相同的結果；其中 golden model 可以是設計者本身或是可執行之規格，或以不易出錯的方式完成的設計



Simulating framework

- Verilog 模擬是為了驗證待測設計的正確性，透過撰寫 testbench，產生 input 訊號的值給待測設計，蒐集並驗證 output 結果是否正確




```verilog
`timescale 1ns / 1ps
module testbench_add2;

reg [3:0] a, b;
wire [3:0] sum;
wire ov;
reg clk;
reg rst;
reg [3:0] correct_sum;
reg [6:0] test_num;
reg correct_ov;

always #5 clk = ~clk;
always #5 rst = ~rst;

add2 DUT(a, b, sum, ov);

initial
begin
    clk <= 0;
    rst <= 0;
    a <= 0;
    b <= 0;
    test_num <= 0;
    $dumpfile("add2.vcd");
    $dumpvars;
end
```

testbench_add2.v

```verilog
module add2(a, b, sum, ov);

    input [3:0] a, b;
    output [3:0] sum;
    output ov;

    assign sum = a + b;
    assign ov = 1'b0;

endmodule
```

add2.v

Example(1/4)

- 我們使用 2-operand adder (二輸入加法器) 作為我們要驗證的 design
- 另外設計了 overflow 永遠等於 0，這樣當有 overflow 發生時，testbench 就會報錯

```verilog
module add2(a, b, sum, ov);

    input [3:0] a, b;
    output [3:0] sum;
    output ov;

    assign sum = a + b;
    assign ov = 1'b0;

endmodule
```

add2.v

Example(2/4)

- 在 testbench 利用 high-level 描述產生正確的結果，並與 add2 的結果做比對

```verilog
else begin
    {correct_ov, correct_sum} = a + b;
    if({ov, sum} == {correct_ov, correct_sum}) begin
        $display ("Test %d ", test_num);
        $display ("CORRECT! No overflow.");
        $display ("%d + %d = ?", a, b);
        $display ("your answer: ov = %d, sum = %d", ov, sum);
        $display ("correct answer: ov = %d, sum = %d", ov, sum);
        $display ("\n");
    end
    else begin
        $display ("Test %d ", test_num);
        $display ("//////////////////////");
        $display ("//////// Fail! ////////");
        $display ("/// Occur Overflow! ///");
        $display ("//////////////////////");
        $display ("%d + %d = ?", a, b);
        $display ("your answer: ov = %d, sum = %d", ov, sum);
        $display ("correct answer: ov = %d, sum = %d", correct_ov, correct_sum);
        $display ("\n");
    end
end
```

testbench_add2.v

Example(3/4)

- 1 ns 為仿真時間，1 ps 為時間精度
- # 是延遲的意思，井號後面數字是延遲的數量，延遲的單位由`timescale控制

```verilog
`timescale 1ns / 1ps
```

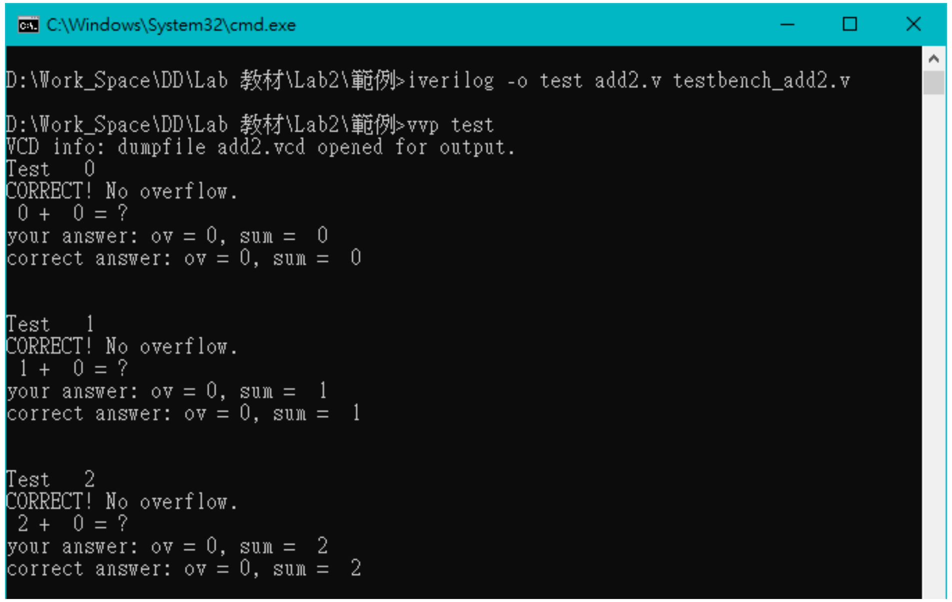
```verilog
always #5 clk = ~clk;
always #5 rst = ~rst;

add2 DUT(a, b, sum, ov);

initial
begin
    clk <= 0;
    rst <= 0;
    a <= 0;
    b <= 0;
    test_num <= 0;
    $dumpfile("add2.vcd");
    $dumpvars;
end

```

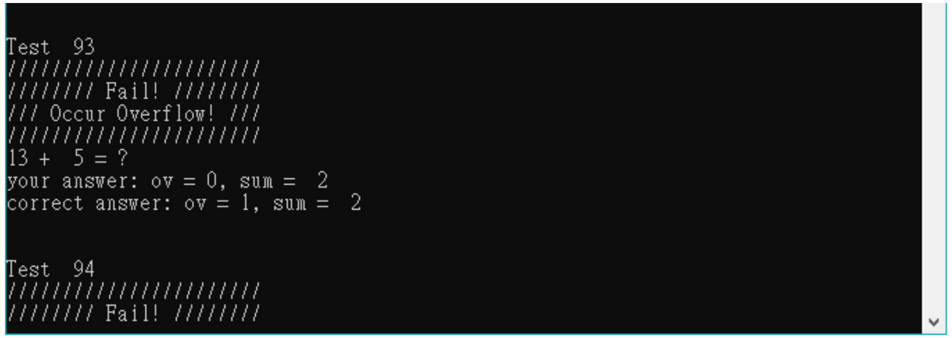
testbench_add2.v

- 利用 $dumpfile 指令產出 vcd 檔，也就是等等會使用到的波形檔

Example(4/4)

- 此圖為編譯結果，並可看到資料夾內產生的 .vcd 電路波形檔

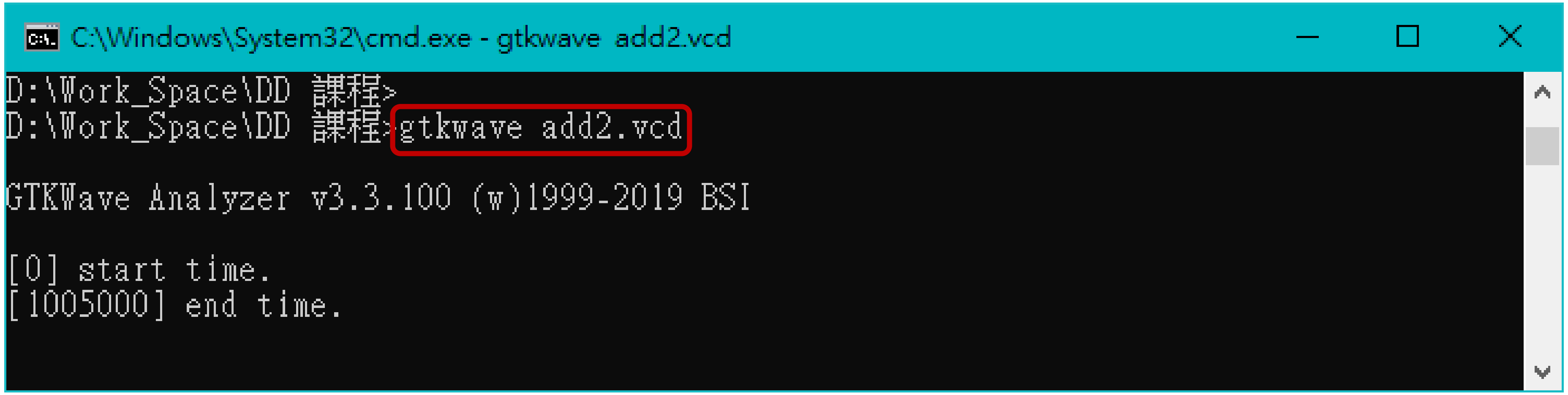




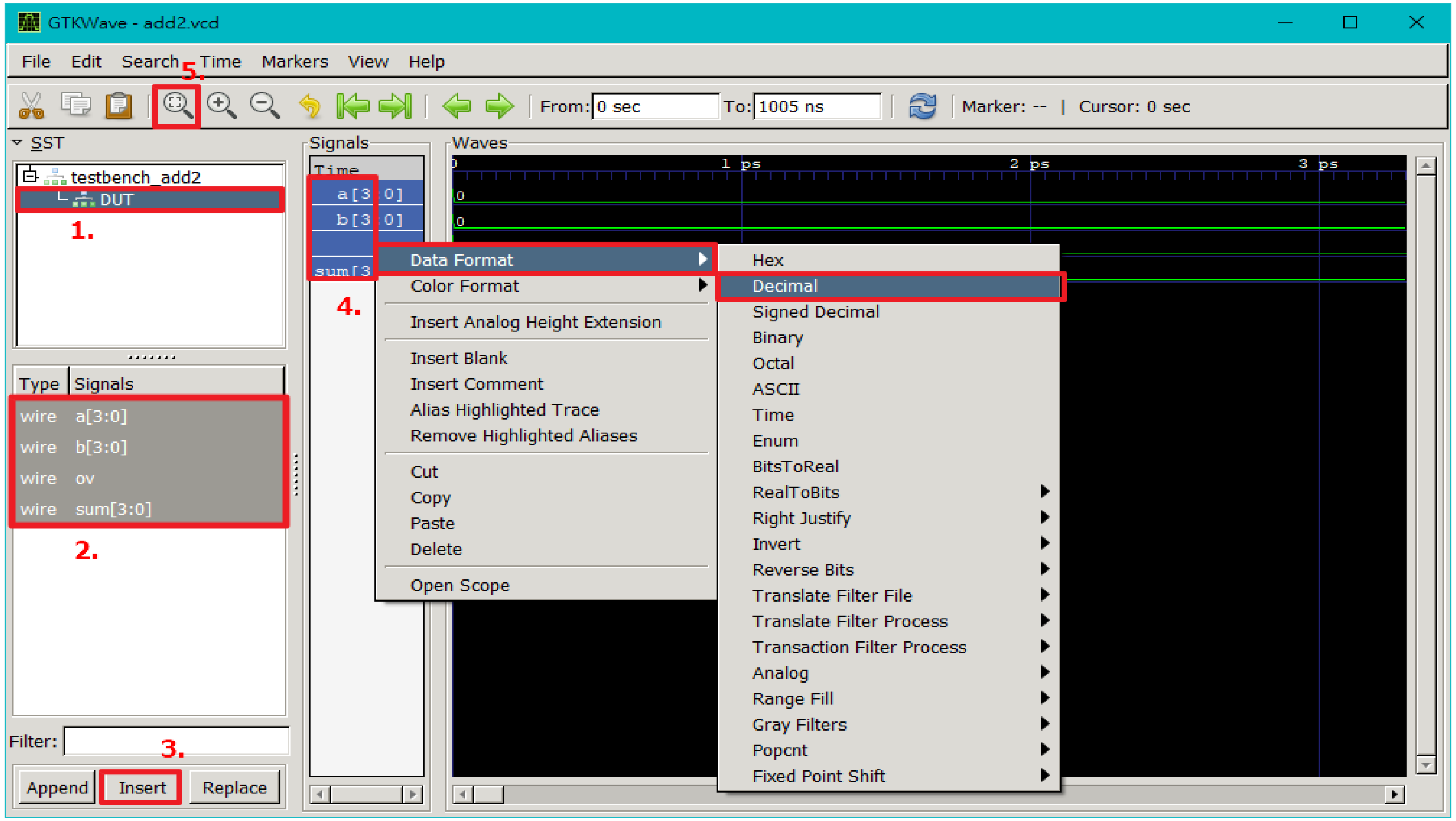
Waveform viewer (1/2)

將範例程式編譯，並使用以下指令觀察波形：

> gtkwave add2.vcd



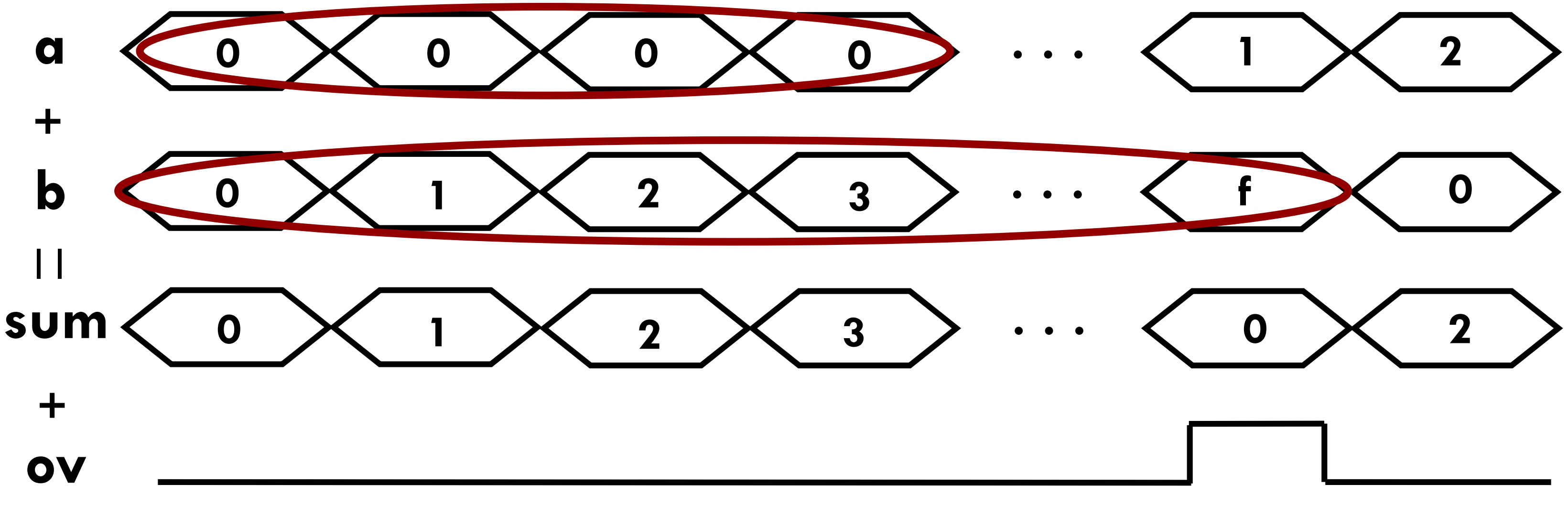
Waveform viewer (2/2)



1. 點選SST 區域中 testbench_add2 中的被測元件 (DUT)
2. 選取變數
3. 點擊 Insert
4. 為方便觀察，按步驟將資料格式改成 10 進位
5. 點擊 Zone Fit

Observation of circuit wave

- 波形會以時序圖的方式呈現，接著就可以觀察結果是否與你設計相符

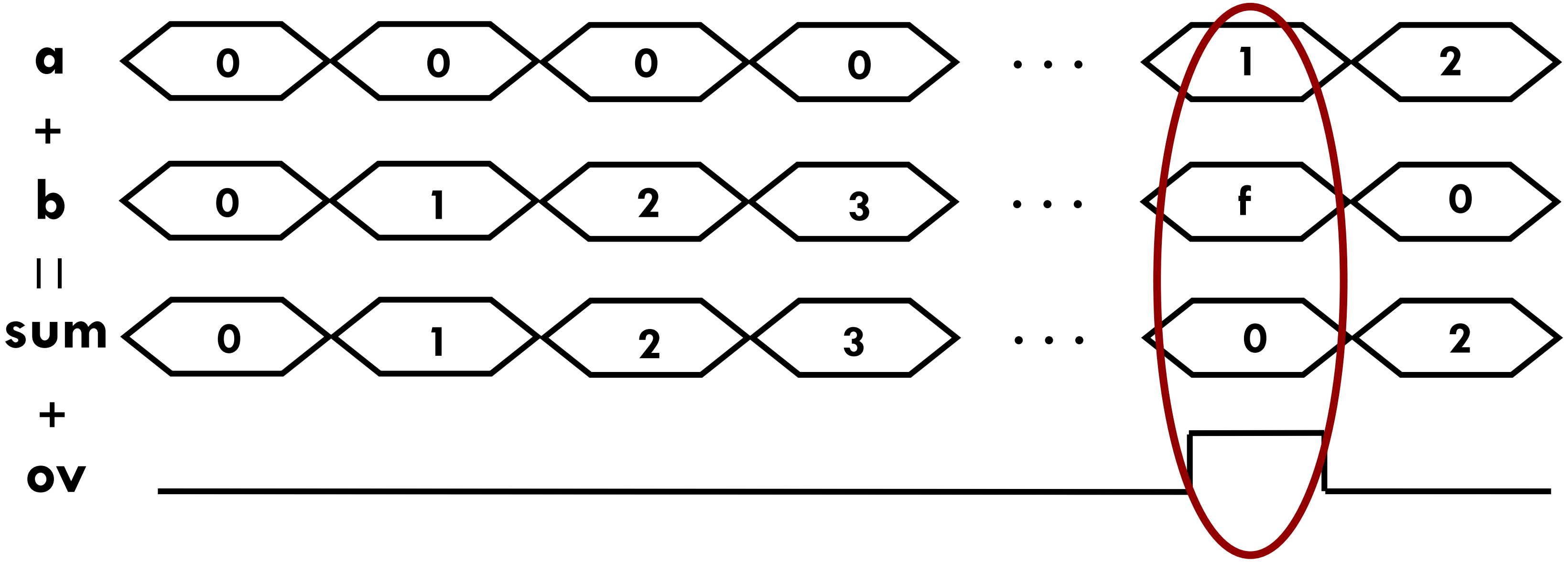


2-input 4-bit adder

$\mathbf{0}_{hex}$	=	0_{dec}	=	0_{oct}	0	0	0	0
$\mathbf{1}_{hex}$	=	1_{dec}	=	1_{oct}	0	0	0	1
$\mathbf{2}_{hex}$	=	2_{dec}	=	2_{oct}	0	0	1	0
$\mathbf{3}_{hex}$	=	3_{dec}	=	3_{oct}	0	0	1	1
$\mathbf{4}_{hex}$	=	4_{dec}	=	4_{oct}	0	1	0	0
$\mathbf{5}_{hex}$	=	5_{dec}	=	5_{oct}	0	1	0	1
$\mathbf{6}_{hex}$	=	6_{dec}	=	6_{oct}	0	1	1	0
$\mathbf{7}_{hex}$	=	7_{dec}	=	7_{oct}	0	1	1	1
$\mathbf{8}_{hex}$	=	8_{dec}	=	10_{oct}	1	0	0	0
$\mathbf{9}_{hex}$	=	9_{dec}	=	11_{oct}	1	0	0	1
$\mathbf{A}_{hex}$	=	10_{dec}	=	12_{oct}	1	0	1	0
$\mathbf{B}_{hex}$	=	11_{dec}	=	13_{oct}	1	0	1	1
$\mathbf{C}_{hex}$	=	12_{dec}	=	14_{oct}	1	1	0	0
$\mathbf{D}_{hex}$	=	13_{dec}	=	15_{oct}	1	1	0	1
$\mathbf{E}_{hex}$	=	14_{dec}	=	16_{oct}	1	1	1	0
$\mathbf{F}_{hex}$	=	15_{dec}	=	17_{oct}	1	1	1	1

Observation of circuit wave

- 波形會以時序圖的方式呈現，接著就可以觀察結果是否與你設計相符

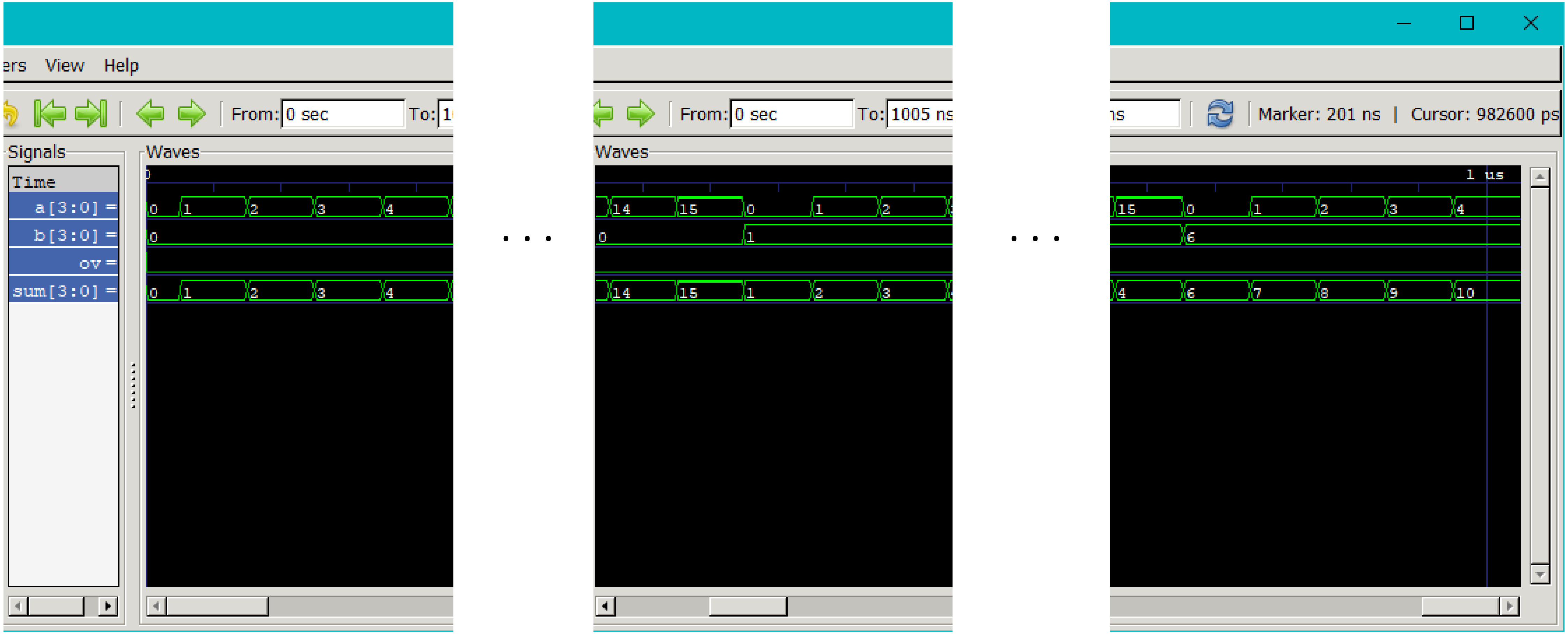


2-input 4-bit adder

$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

Result

查看生成的波形圖，可以確認設計是否正確



Homework

- 題目：完成 4-operand adder 自動結果比對，請同學參考 Example 將輸入從零到十五所有組合當作輸入的 pattern，並 show 出結果
- 我們會拿左下角的 design，用同學們做的 testbench 驗證 ，result 為結果，間隔相加為 5，最後一個 ~~cycle~~ 為 13107 應該是Test 才對，影片中為錯誤的
- 助教會看資料波形，請同學打開 gtkwave 的波形圖

```
add4.v
module add4(a, b, c, d, sum, ov);

    input [3:0] a, b, c, d;
    output [4:0] sum;
    output ov;

    assign sum = a + b + c + d;
    assign ov = 1'b0;

endmodule
```

add4.v

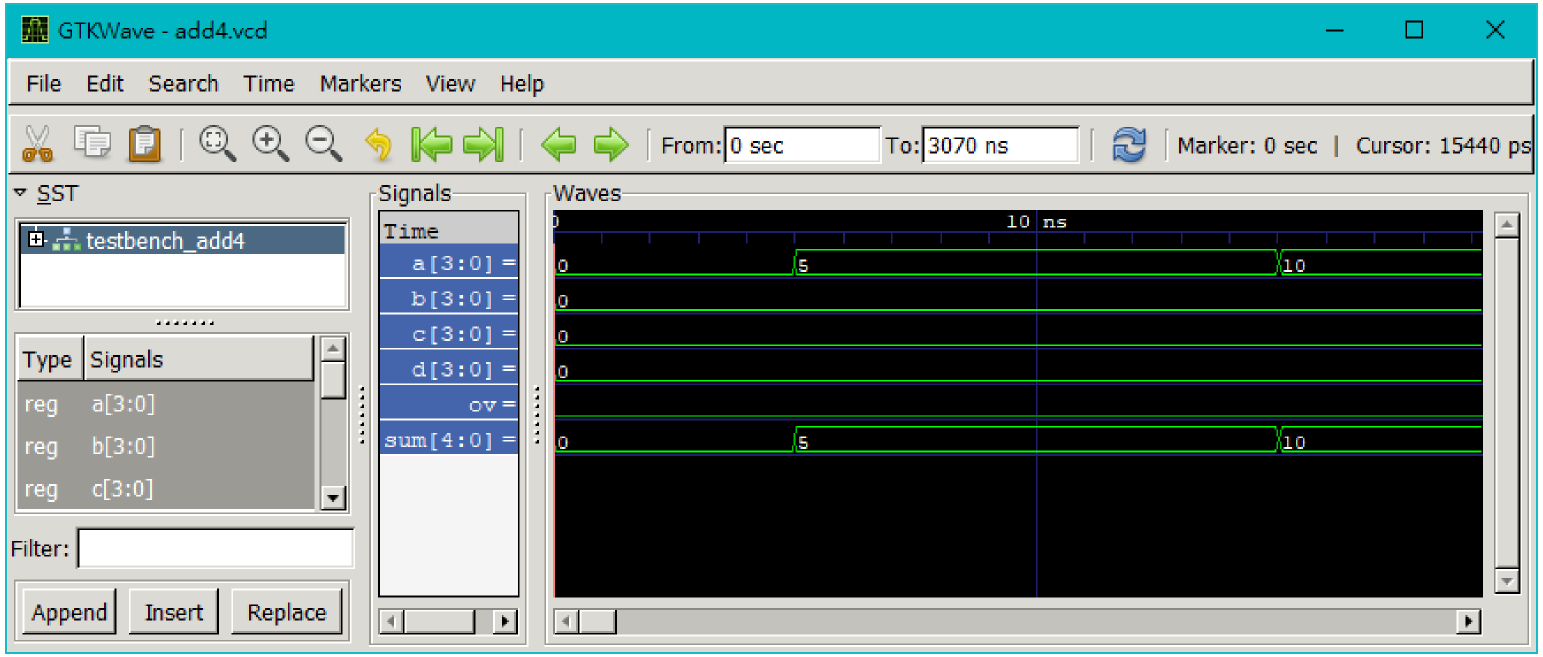
```
C:\Windows\System32\cmd.exe - vvp test
D:\Work_Space\DD\Lab 教材\Lab2\作業>vvp test
VCD info: dumpfile add4.vcd opened for output.
Test      0
CORRECT! No overflow.
 0 +  0 +  0 +  0 = ?
your answer: ov = 0, sum =  0
correct answer: ov = 0, sum =  0

Test      1
CORRECT! No overflow.
 5 +  0 +  0 +  0 = ?
your answer: ov = 0, sum =  5
correct answer: ov = 0, sum =  5
```

```
Test 13107
////////////////////////
//////// Fail! ////////
/// Occur Overflow! ///
////////////////////////
15 + 15 + 15 + 15= ?
your answer: ov = 0, sum = 28
correct answer: ov = 1, sum = 28
```

result

Homework

- 題目：完成 4-operand adder 自動結果比對，請同學參考 Example 將輸入從零到十五所有組合當作輸入的 pattern，並 show 出結果
- 我們會拿左下角的 design，用同學們做的 testbench 驗證
- 助教會看資料波形，請同學打開 gtkwave 的波形圖



add4.vcd

Demo

- Demo 地點：EA 501A
- Demo 時間：依時間表為準
- 可提前5分鐘入場準備，其餘時間違規進入，一次扣總成績3分
- 安排時段內無法展示請即刻離場，違者一次扣總成績5分
- 可攜帶自己的筆電 Demo
- 請以隨身碟攜帶檔案 (不要用雲端)