5G通訊演算法工程師培訓課程

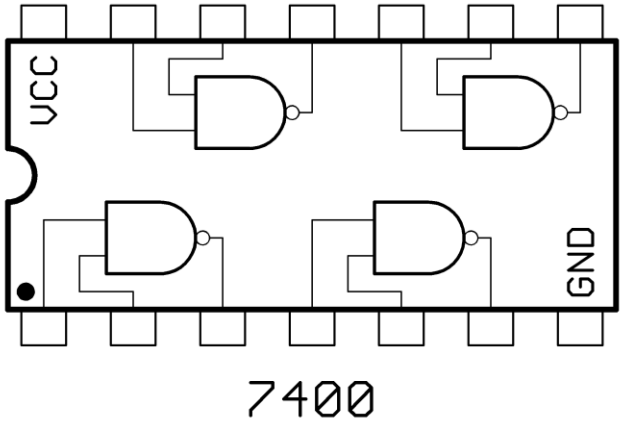
Course Overview

2021

Outlines

* FPGA/Verilog 數位邏輯電路設計
  * Introduction to FPGA
  * Introduction to Verilog
* 通道編碼技術(Channel Coding)
* OFDM 與多通道通訊系統

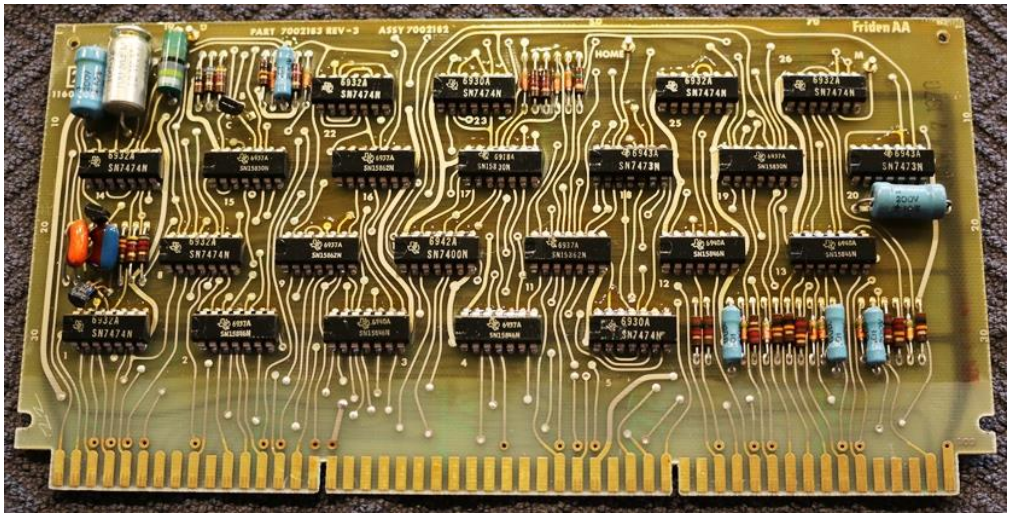
Old ways of implementing Digital Circuits



7400

* 高成本
* 速度慢
* 易產生接線問題

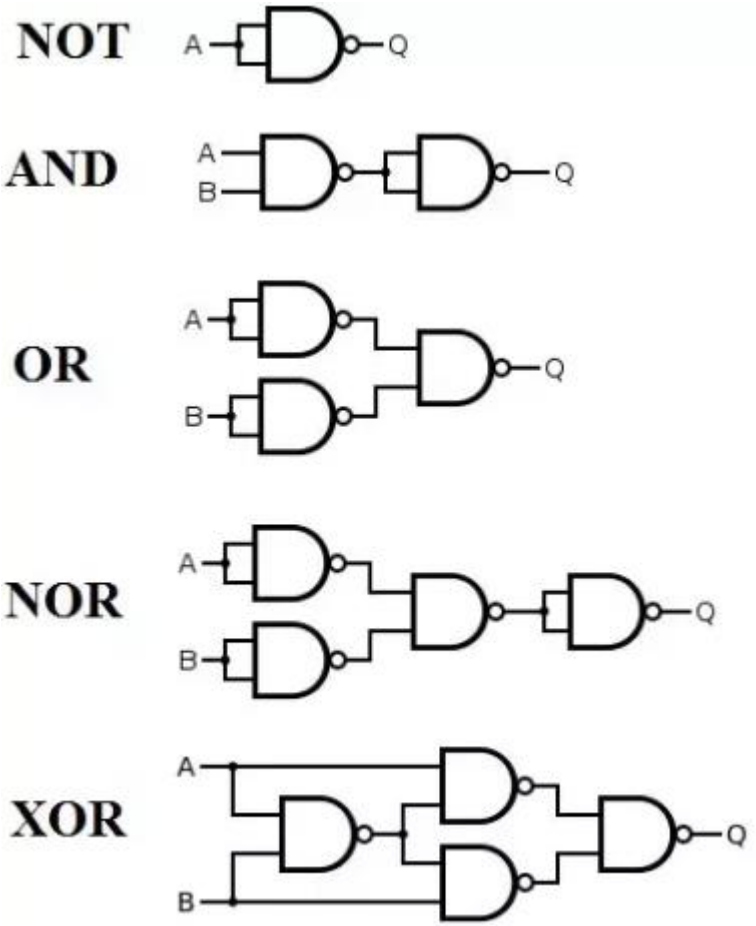
Input	Input	Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



https://electronics.stackexchange.com/questions/246887/ttl-7400-nand-gate-circuit-not-functioning

Old ways of implementing Digital Circuits

* De Morgan's theorem (笛摩根定理):
  - We only need 2-input **NAND** or **NOR** gates to build anything



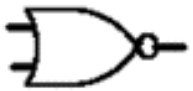
NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Boolean Expression

$Y = (A \cdot B)' = A' + B'$

NOR Gate



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

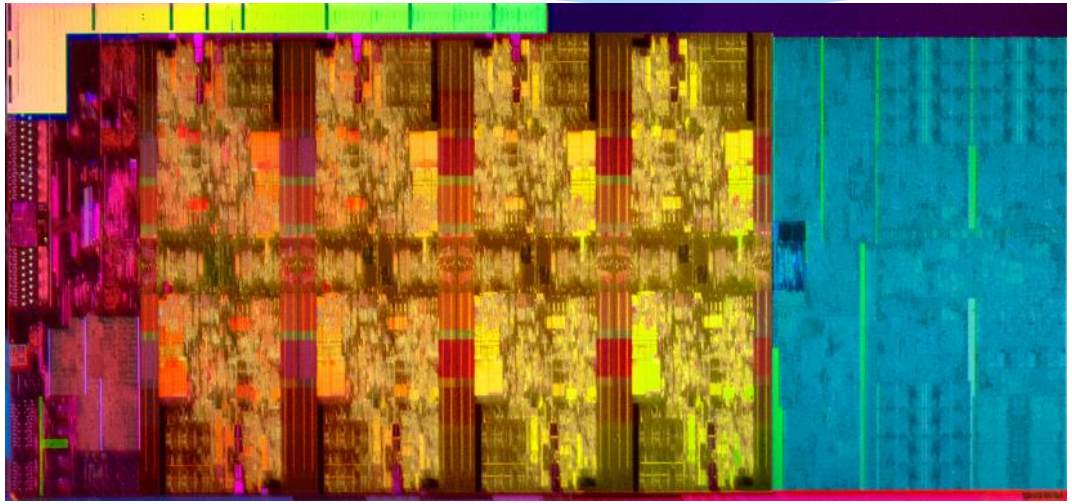
Boolean Expression

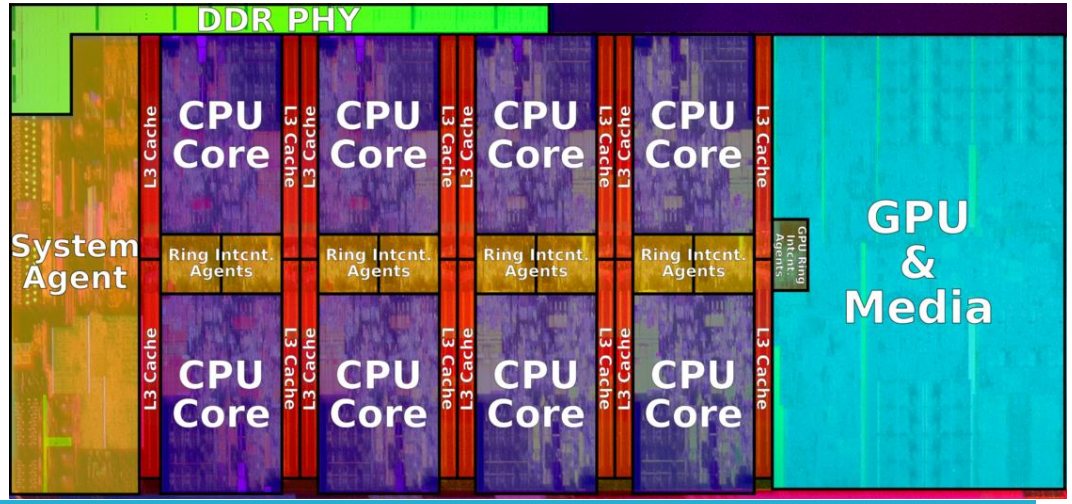
$Y = (A + B)'$

https://electronics.stackexchange.com/questions/246887/ttl-7400-nand-gate-circuit-not-functioning

Modern Digital Design (Full Custom IC)

Intel Core i9-9900K



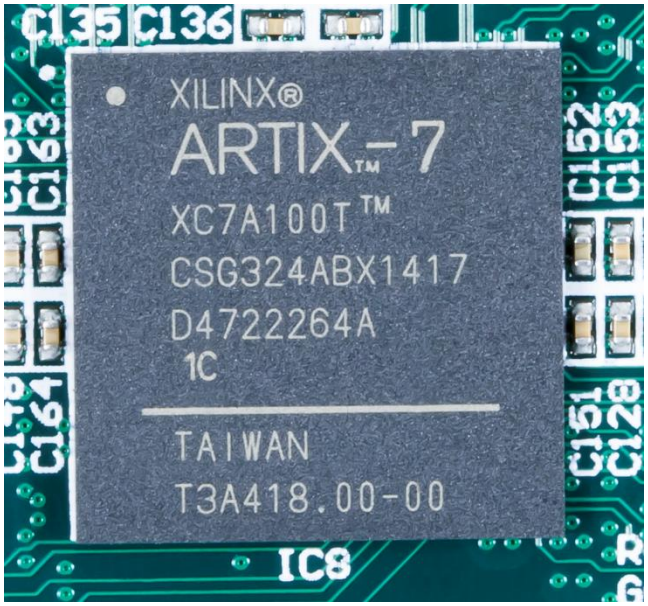


* Total **core transistor** count > 1.736 Billion **transistors**.
* Very expensive to design
* Very expensive to manufacture
* Need very large market

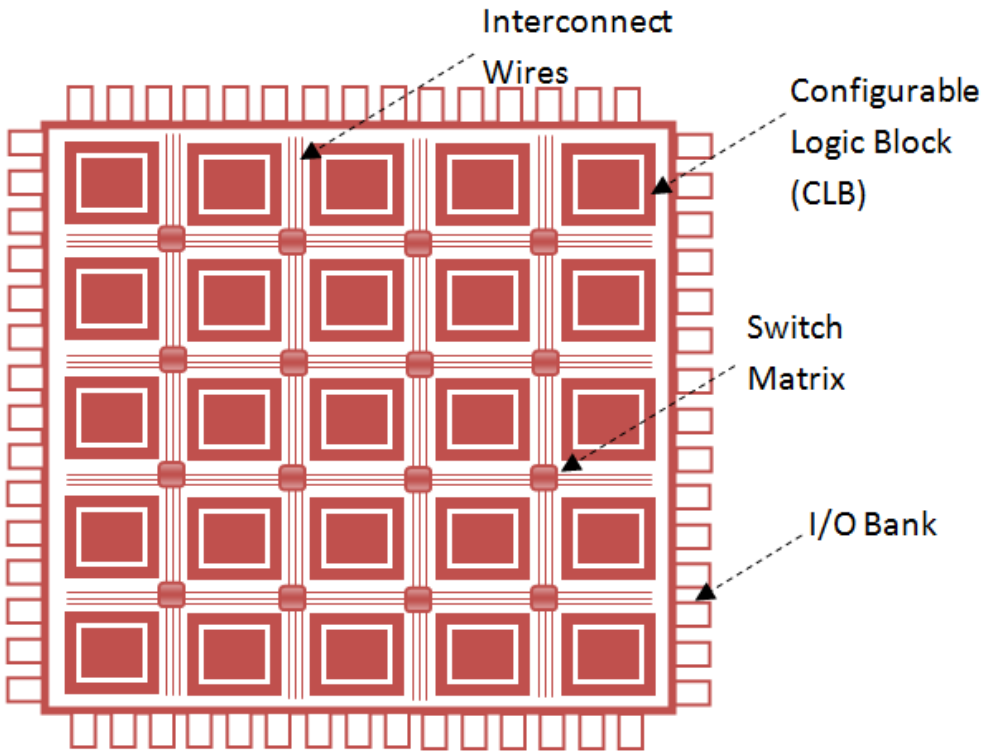
https://en.wikichip.org/wiki/intel/core_i9/i9-9900k

Field Programmable Gate Arrays (FPGAs)

Xilinx 7-series



FPGA Architecture



* Configurable logic Block (CLB)
* Input/Output Block
* Switching Matrix Interconnects

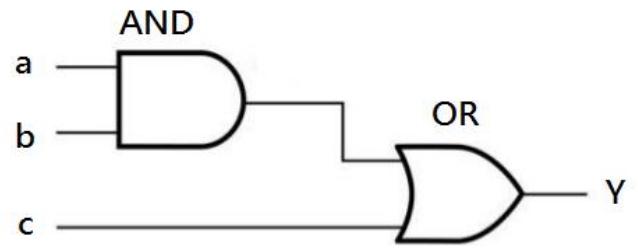
https://blog.digilentinc.com/structure-of-an-fpga/
https://allaboutfpga.com/fpga-architecture/

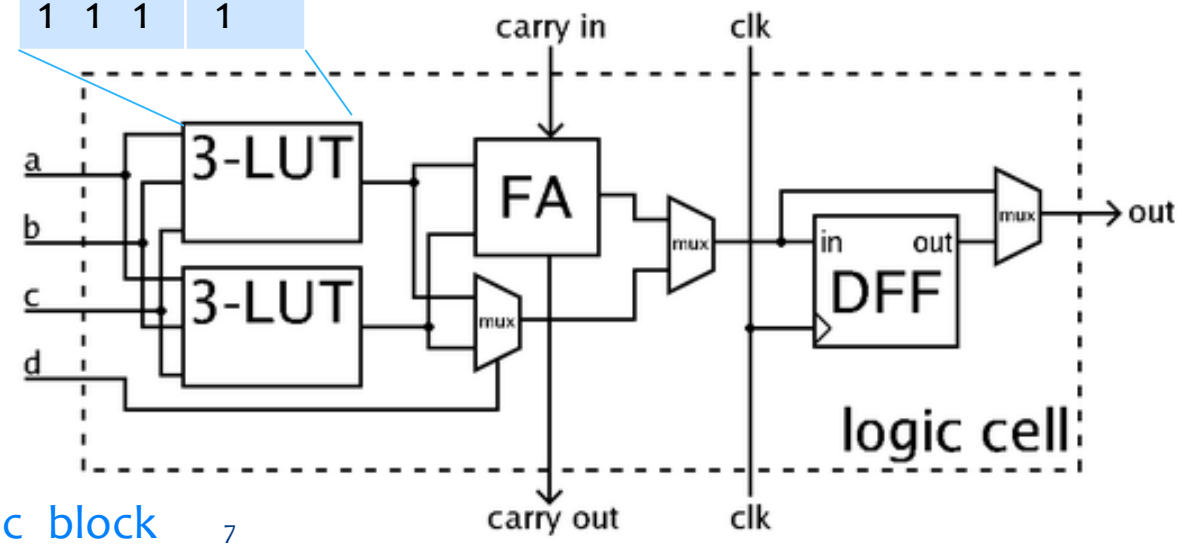
FPGA- Configurable logic Block (CLB)

* Lookup Table (LUT), Arithmetic Circuit (Full Adder) , Multiplexers (Mux logic), and Flip-flop(s)
* **LUT** stores truth table of logic function to be implemented

LUT

a	b	c	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	0	0	1
1	1	1	1





https://en.wikipedia.org/wiki/Logic_block

FPGA - Major Manufacturers

* **Xilinx** (賽靈思公司) 是全球第一大FPGA供應商。FPGA、可程式SoC及ACAP（Adaptive Compute Acceleration Platform) 的發明者。
* **Altera**是FPGA的領先廠商，於2016年被Intel收購，成為Intel子公司。
* **Microchip**
  - **Microsemi** (previously Actel),開發 antifuse, flash-based, mixed-signal FPGAs
  - **Atmel**, a second source of some Altera-compatible devices;
* **Lattice Semiconductor**開發 Low-power SRAM-based FPGAs
* **QuickLogic** 開發 Ultra Low Power Sensor Hubs, extremely low powered, low density SRAM-based FPGAs
* **Achronix** Semiconductor 開發中非常快的FPGA: SRAM based FPGAS with 1.5 GHz fabric speed

https://zh.wikipedia.org/wiki/%E7%8E%B0%E5%9C%BA%E5%8F%AF%E7%BC%96%E7%A8%8B%E9%80%BB%E8%BE%91%E9%97%A8%E9%98%B5%E5%88%97

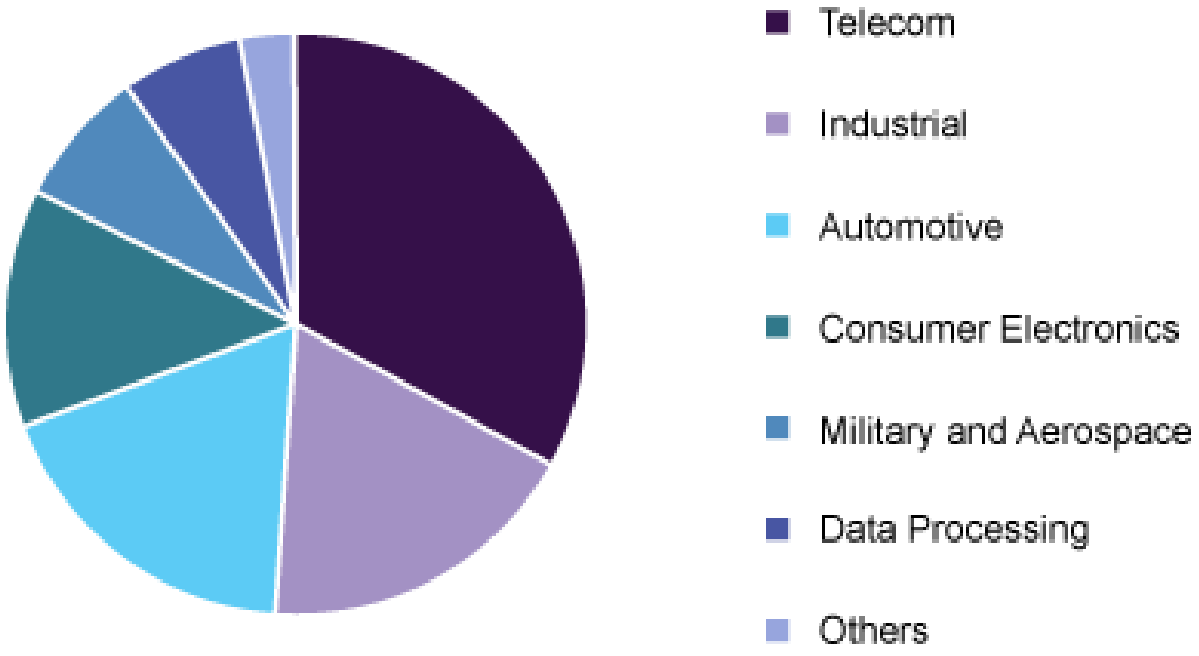
FPGA market share -1

	2015		2016		
Vendor	FPGA Total	Market share	FPGA Total	Market share	Growth CY15-CY16
Xilinx	$2,044	53%	$2,167	53%	6%
Intel (Altera)	$1,389	36%	$1,486	36%	7%
Microsemi	$301	8%	$297	7%	-1%
Lattice	$124	3%	$144	3%	16%
QuickLogic	$19	0%	$11	0%	-40%
Others	$2	0%	$2	0%	0%
TOTAL	**$3,879**	**100%**	**$4,112**	**100%**	**6%**

https://www.programmersought.com/article/66291005870/

FPGA market share -2

Global FPGA market share, by application, 2019 (%)



Source: www.grandviewresearch.com

https://www.grandviewresearch.com/industry-analysis/fpga-market

FPGA market share -2

* 例如: **Intel® Cyclone® 10 FPGA**
  - 提供快速處理、省電能力，能及時處理複雜影像與分析
  - 適用於汽車、工業自動化、專業影音及視覺系統等各種應用
    - 如: 使用於汽車後視攝影機及傳感器融合的視訊處理中，車輛行駛時可以結合車上多個傳感器所收集到的數據，提供更完整的行車狀況資訊

https://www.intel.com.tw/content/www/tw/zh/products/programmable/fpga/cyclone-10.html

Xilinx FPGA 產品名稱(以製程分類)

FPGA Leadership across Multiple Process Nodes

Xilinx offers a comprehensive multi-node portfolio to address requirements across a wide set of applications. Whether you are designing a state-of-the art, high-performance networking application requiring the highest capacity, bandwidth, and performance, or looking for a low-cost, small footprint FPGA to take your software-defined technology to the next level, Xilinx FPGAs and 3D ICs provide you with system integration while optimizing for performance/watt.

Xilinx Multi-Node Product Portfolio Offering

45nm	28nm	20nm	16nm
SPARTAN-6	VIRTEX-7 KINTEX-7 ARTIX-7 SPARTAN-7	VIRTEX UltraSCALE KINTEX UltraSCALE	VIRTEX UltraSCALE+ KINTEX UltraSCALE+

https://www.xilinx.com/products/silicon-devices/fpga.html

Xilinx FPGA 產品表

Product Tables and Product Selection Guides

Cost-Optimized Portfolio

Spartan-7	Spartan-6
Artix-7	Zynq-7000

7 Series

Spartan-7	Artix-7
Kintex-7	Virtex-7

UltraScale

Kintex UltraScale	Virtex UltraScale

UltraScale+

Kintex UltraScale+	Virtex UltraScale+

https://www.xilinx.com/products/silicon-devices/fpga.html

FPGA 入門開發板 – Basys3

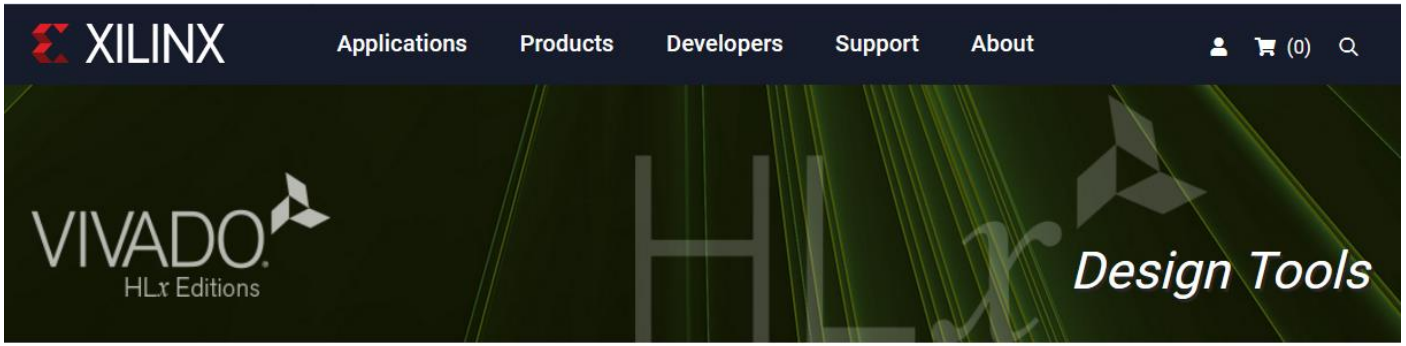
* Basys3 為入門級的FPGA實驗板
  - 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
* 針對Vivado Design Suite而設計
* 具有Xilinx Artix-7 FPGA架構

Xilinx Artix7 Basys3 FPGA 開發板

約 TWD 5,000

Basys3 教學資源

* Vivado Design Suite教學網
  - https://www.xilinx.com/products/design-tools/vivado.html?_ga=2.130255247.1825609563.1595573298-60285558.1595573298



/ Developer Tools / Vivado Design Suite

Vivado Design Suite - HLx Editions

productivity multiplied

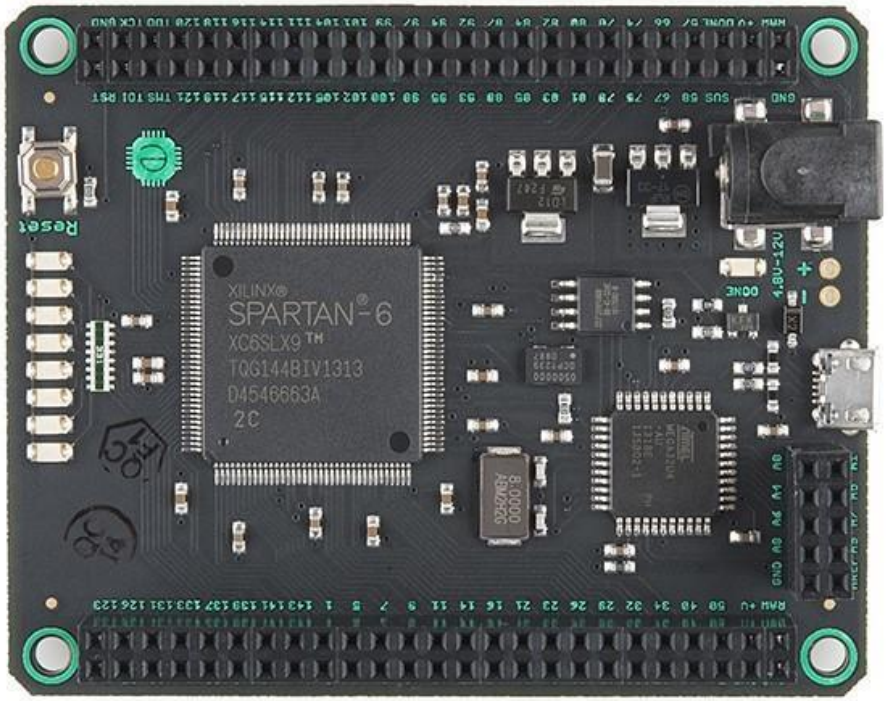
What's New in Vivado

Download Vivado

FPGA 入門開發板 – Mojo v3

- Spartan 6 XC6SLX9 FPGA
- 84 digital IO pins
- 8 analog inputs
- 8 general purpose LEDs
- 1 LED to show when the FPGA is correctly configured
- On board voltage regulation that can handle 4.8-12V
- ATmega32U4 used for configuring the FPGA, USB communications, and reading the analog pins
- On board flash memory to store the FPGA configuration file.

Mojo v3 FPGA-compatible FPGA Development Board Spartan6

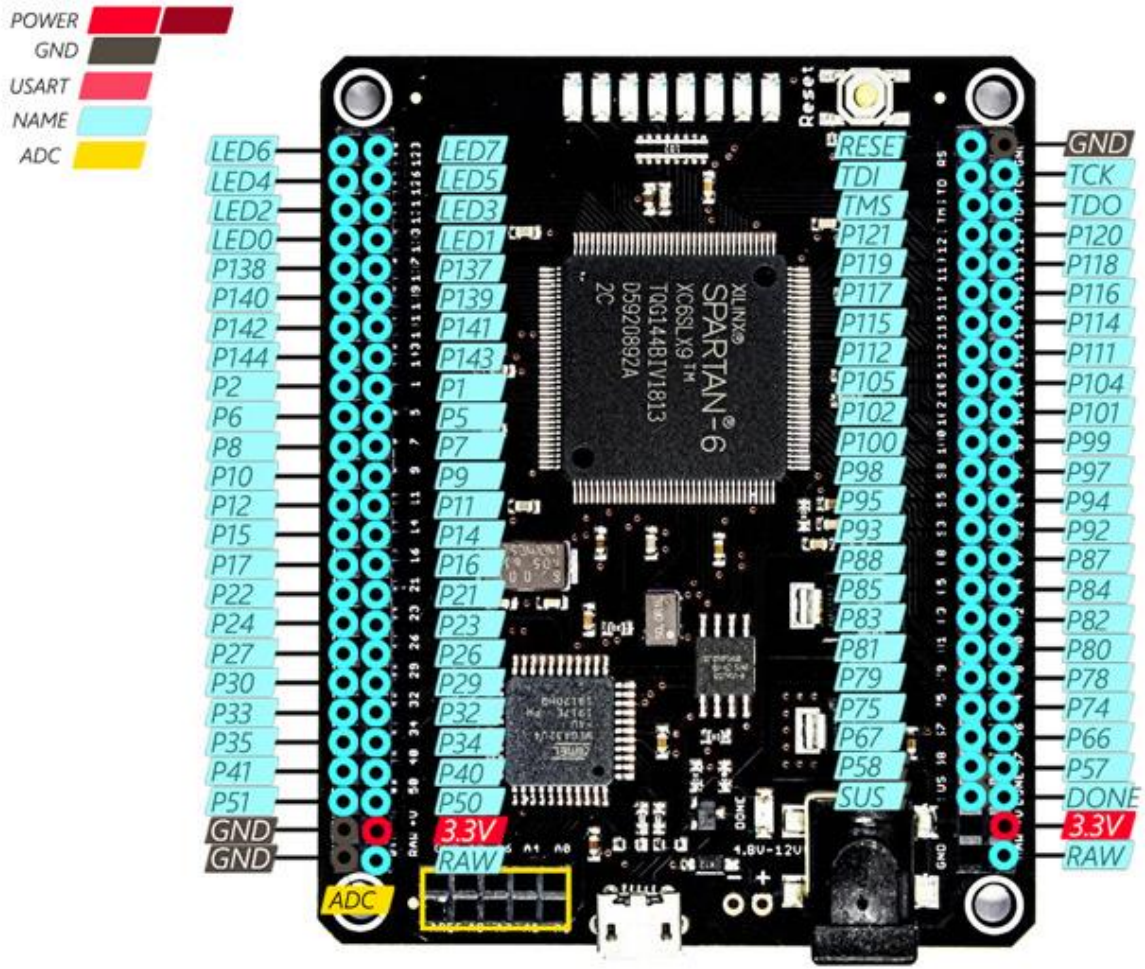


https://www.sparkfun.com/products/retired/11953

約 TWD 1,500

FPGA 入門開發板 – Mojo v3

Mojo V3 FPGA Development Board Pinout



www.Electropeak.com

https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/

Mojo v3 教學資源

* **Mojo v3 教學網**
  - https://alchitry.com/pages/verilog-fpga-tutorials

Verilog - FPGA Tutorials

Welcome to the Verilog tutorials page! This page has all the information you need to get started using the Au, Cu, or Mojo with Verilog.

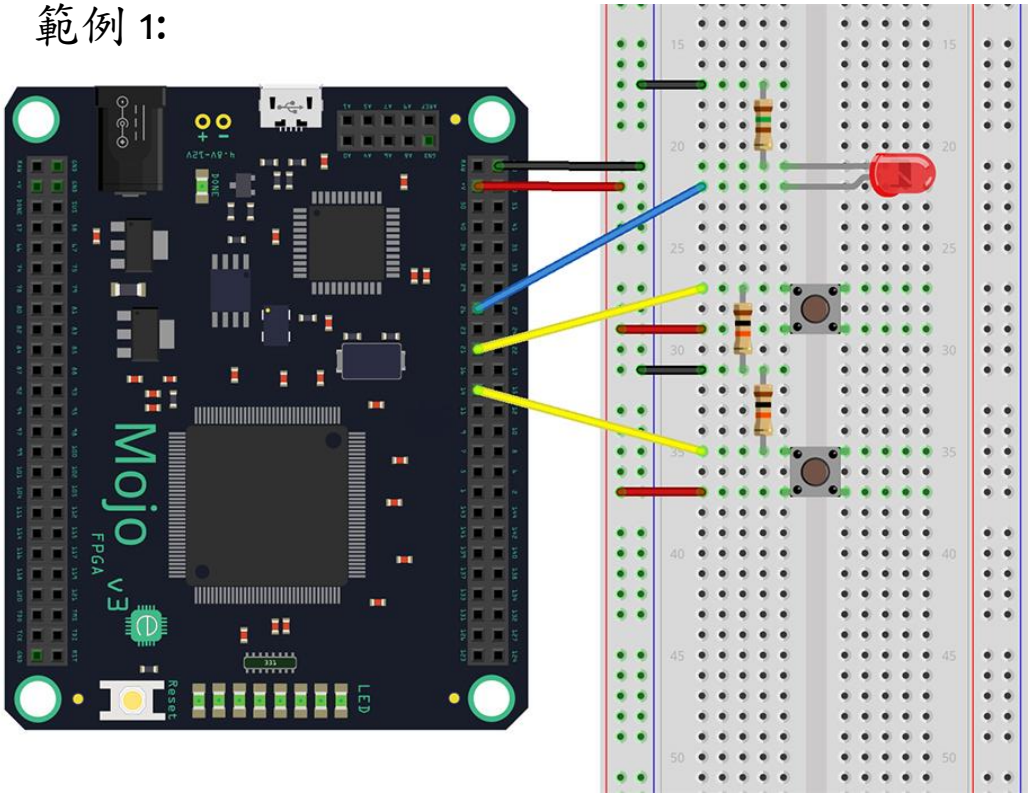
If you are a beginner, we recommend following the Lucid tutorials instead. Lucid is a language we developed to make working with FPGAs easier. We believe this is the best place for a beginner to start.

Mojo v3 教學資源

Mojo v3 教學網

- https://micro.rohm.com/tw/deviceplus/how-tos/fpga-tutorial-intro-to-fpgas-with-the-mojo-part-1/

範例 1:



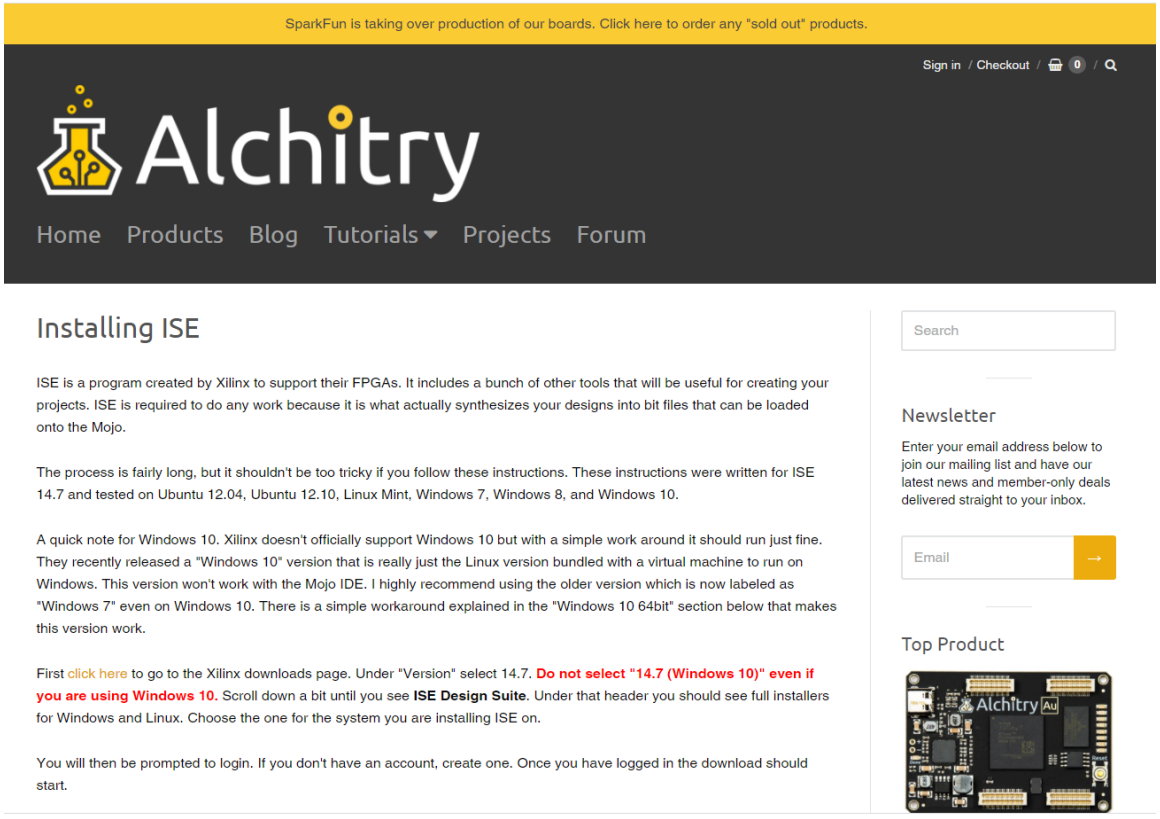
硬體:
- Mojo V3 研發板
- Micro USB 電纜
- Windows 或 Linux 電腦（不幸的是，目前不支援Mac作業系統）
- 紅色 LED
- 1 x 150Ω 或類似電阻（用於LED限流）
- 2 x 10kΩ 電阻（用於按鈕下拉）
- 麵包板和跳線

軟體:
- Mojo IDE （將設計實現上傳至FPGA）
- ISE Design Suite（編寫和綜合Verilog代碼的IDE）
  - 在下載頁面上向下滾動，直到看到標有 “ISE Design Suite”的部分
- Mojo Base Project（所有項目都將以Mojo提供的該項目骨架為基礎）

Mojo v3 教學資源

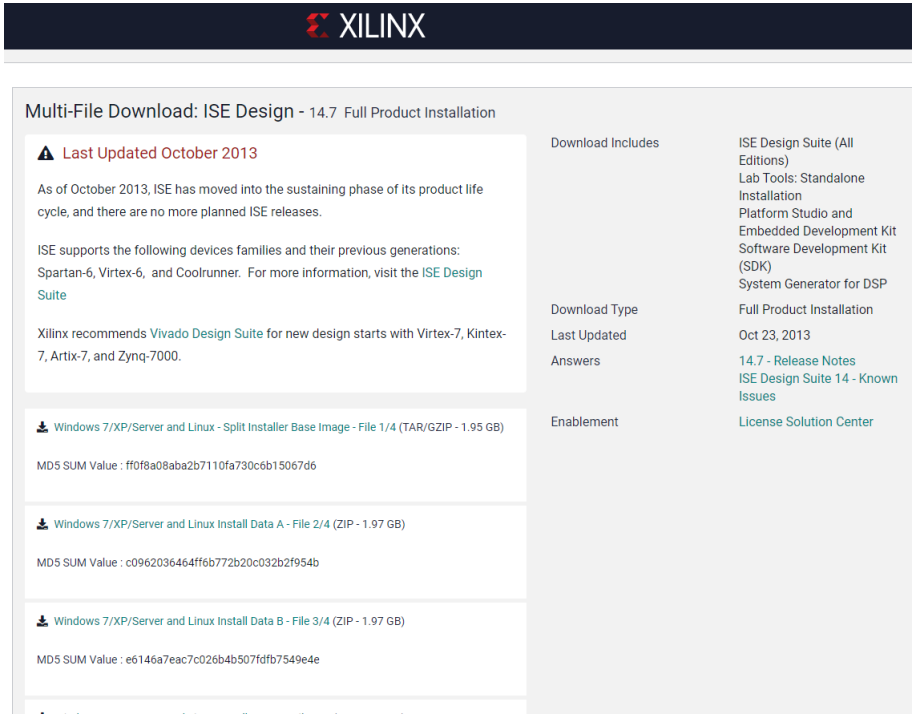
Mojo v3 教學網

- STEP 1: Installing ISE https://alchitry.com/pages/installing-ise



Mojo v3 教學資源

* **Mojo v3 教學網**
  - STEP 1: Installing ISE
    https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive-ise.html

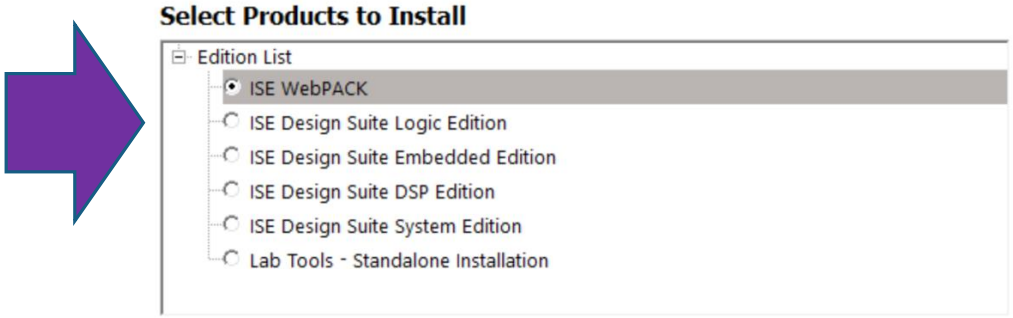


Mojo v3 教學資源

Mojo v3 教學網

- STEP 1: Installing ISE
  https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive-ise.html

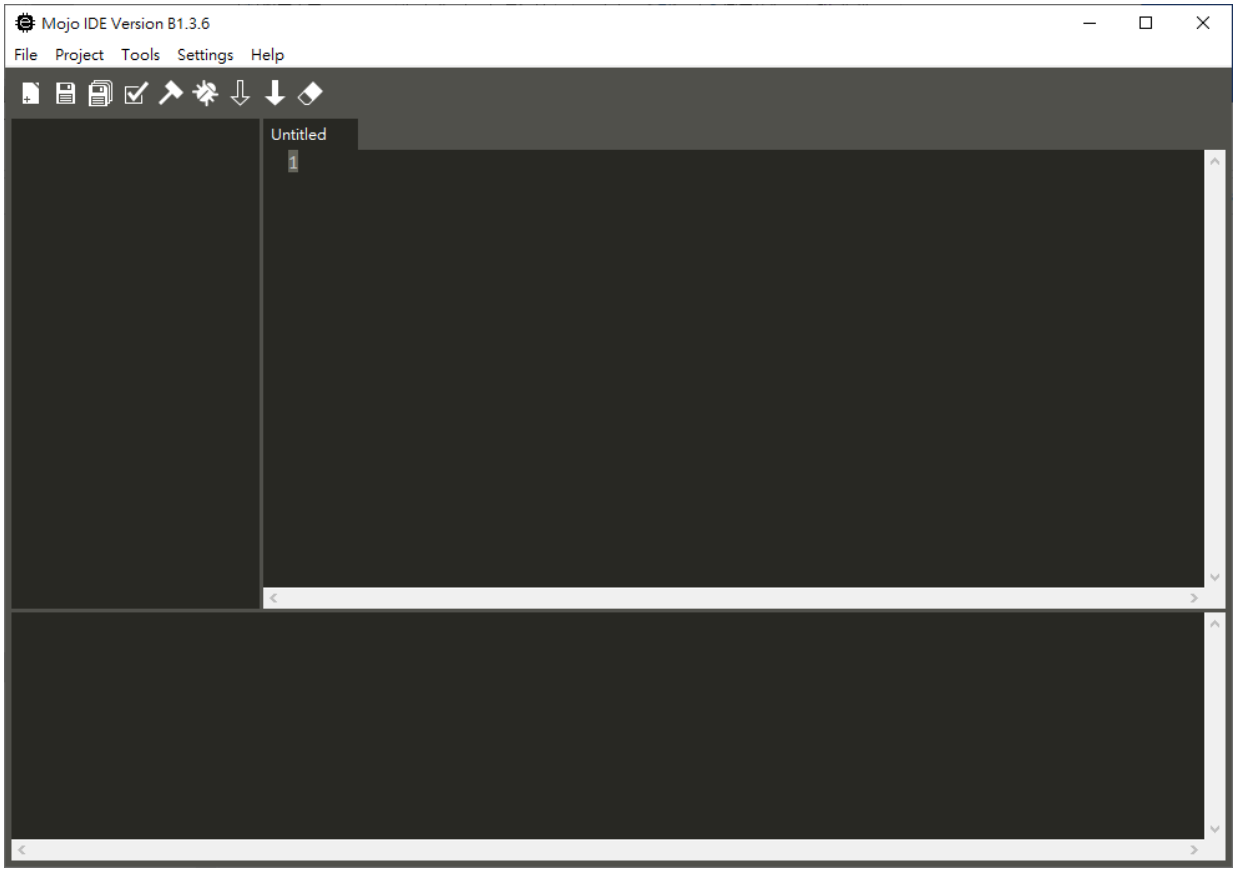
- Xilinx_ISE_DS_14.7_1015_1-1.tar
- Xilinx_ISE_DS_14.7_1015_1-2.zip.xz
- Xilinx_ISE_DS_14.7_1015_1-3.zip.xz
- Xilinx_ISE_DS_14.7_1015_1-4.zip.xz



 https://myz80.wordpress.com/2020/10/11/getting-started-with-mojo-v3-fpga-in-windows-10/

Mojo v3 教學資源

* **Mojo v3 教學網**
  - Step 2: Mojo IDE https://alchitry.com/pages/mojo-ide



Mojo IDE vB1.3.6

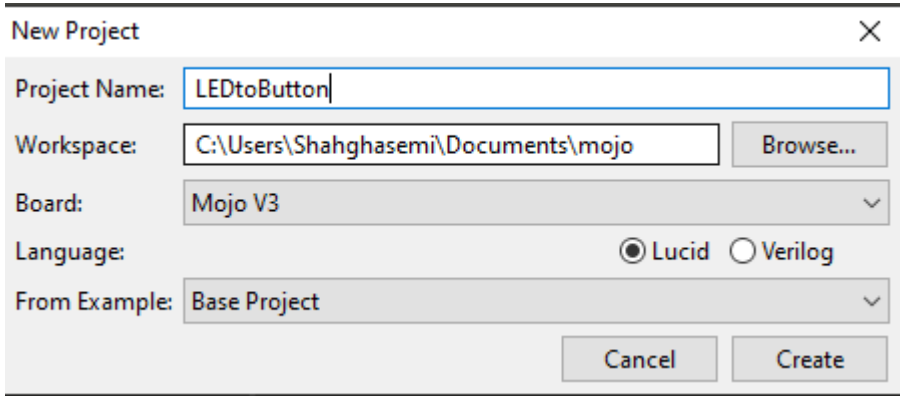
After finishing the **Xilinx ISE Design Suite** software installations , open the Mojo IDE software.

https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/

Mojo v3 教學資源

Mojo v3 教學網

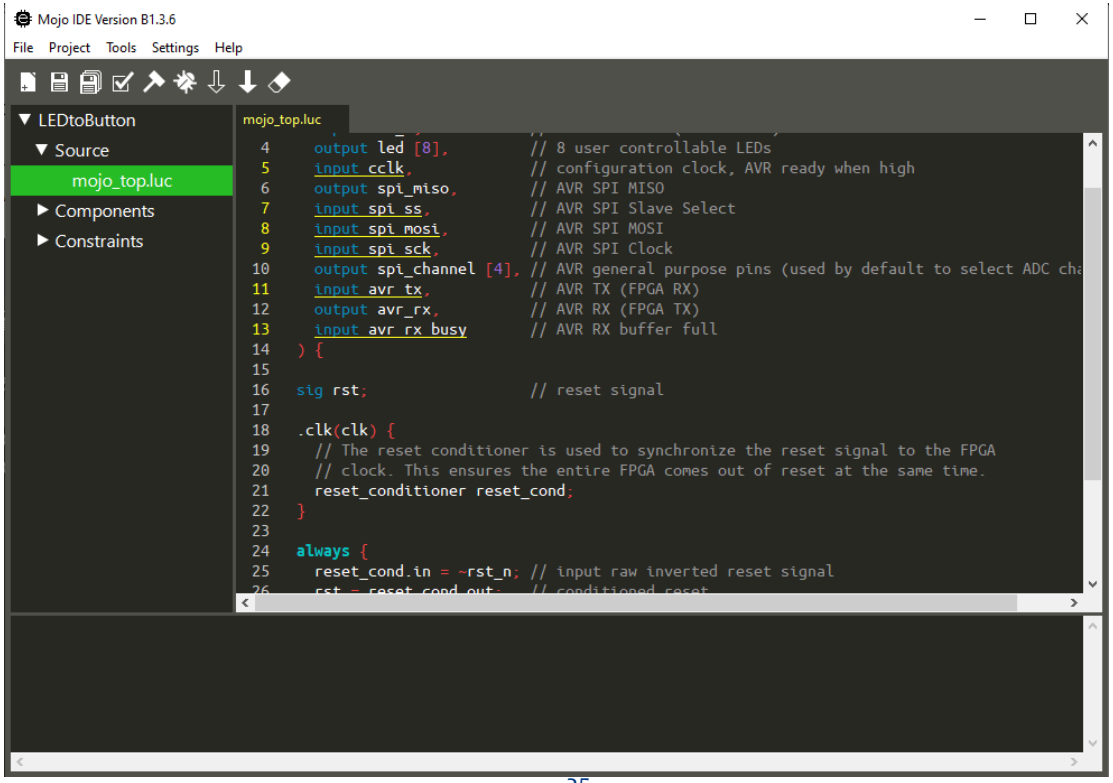
- Step 3: Code https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/
  - Click on **New Project** from the **File** menu of **Mojo IDE**, and fill in the blanks as following.



Mojo v3 教學資源

Mojo v3 教學網 (範例 2)

- Step 3: Code https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/
  - Now enter the **Source**, and open **mojo_top.luc** file.



Mojo v3 教學資源

Mojo v3 教學網

Step 3: Code https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/

- Now enter the **Source**, and open **mojo_top.luc** file.

```
/*
 Modify on March 17, 2021
 Modify by MohammedDamirchi base of Example
 https://electropeak.com/learn/
*/

module mojo_top (
  input clk,           // 50MHz clock
  input rst_n,         // reset button (active low)
  output led [8],      // 8 user controllable LEDs
  input cclk,          // configuration clock, AVR ready when high
  output spi_miso,     // AVR SPI MISO
  input spi_ss,        // AVR SPI Slave Select
  input spi_mosi,      // AVR SPI MOSI
  input spi_sck,       // AVR SPI Clock
  output spi_channel [4], // AVR general purpose pins (used by default to select ADC channel)
  input avr_tx,        // AVR TX (FPGA RX)
  output avr_rx,       // AVR RX (FPGA TX)
  input avr_rx_busy    // AVR RX buffer full
 ) {
```

Mojo v3 教學資源

Mojo v3 教學網

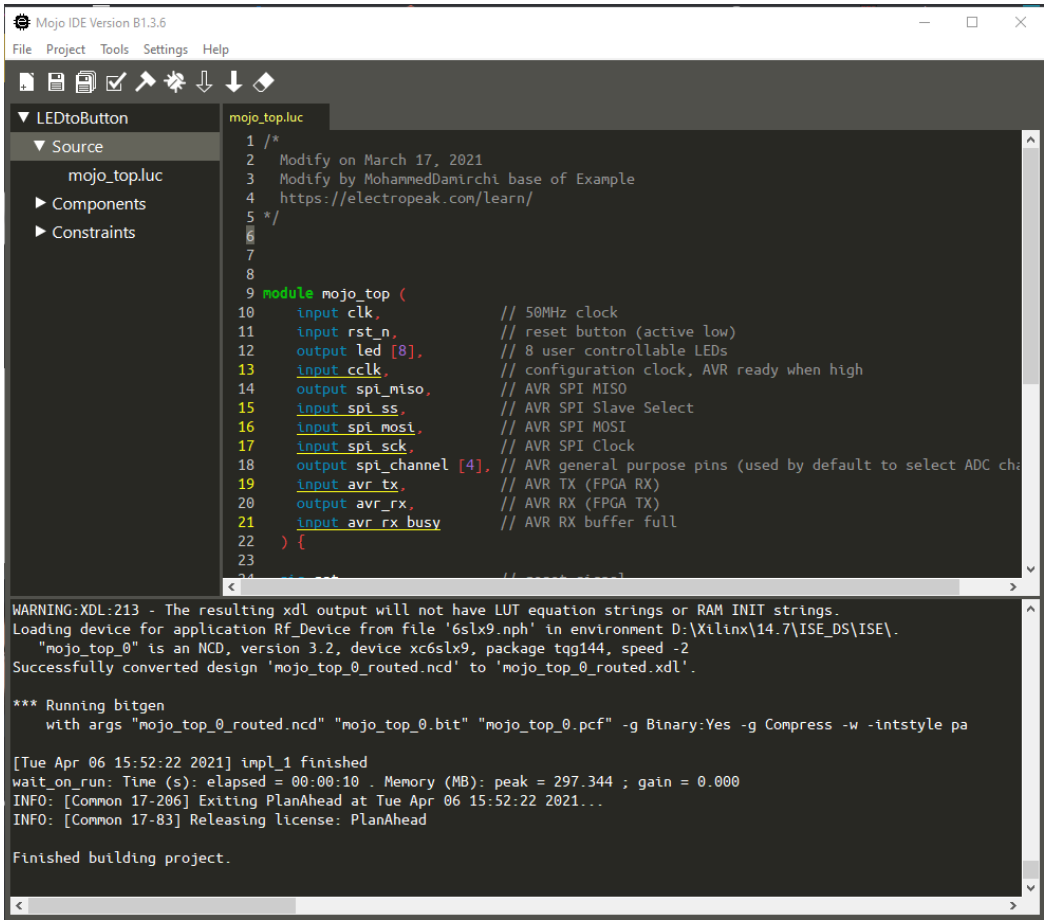
- Step 3: Code https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/
  - Now enter the **Source**, and open **mojo_top.luc** file.

```
sig rst;              // reset signal

.clk(clk) {
  // The reset conditioner is used to synchronize the reset signal to the FPGA
  // clock. This ensures the entire FPGA comes out of reset at the same time.
  reset_conditioner reset_cond;
}

always {
  reset_cond.in = ~rst_n; // input raw inverted reset signal
  rst = reset_cond.out;   // conditioned reset

  led= c{7b0,rst};          // turn LEDs off
  spi_miso = bz;         // not using SPI
  spi_channel = bzzzz;    // not using flags
  avr_rx = bz;          // not using serial port
 }
}
```

Mojo v3 教學資源

Mojo v3 教學網

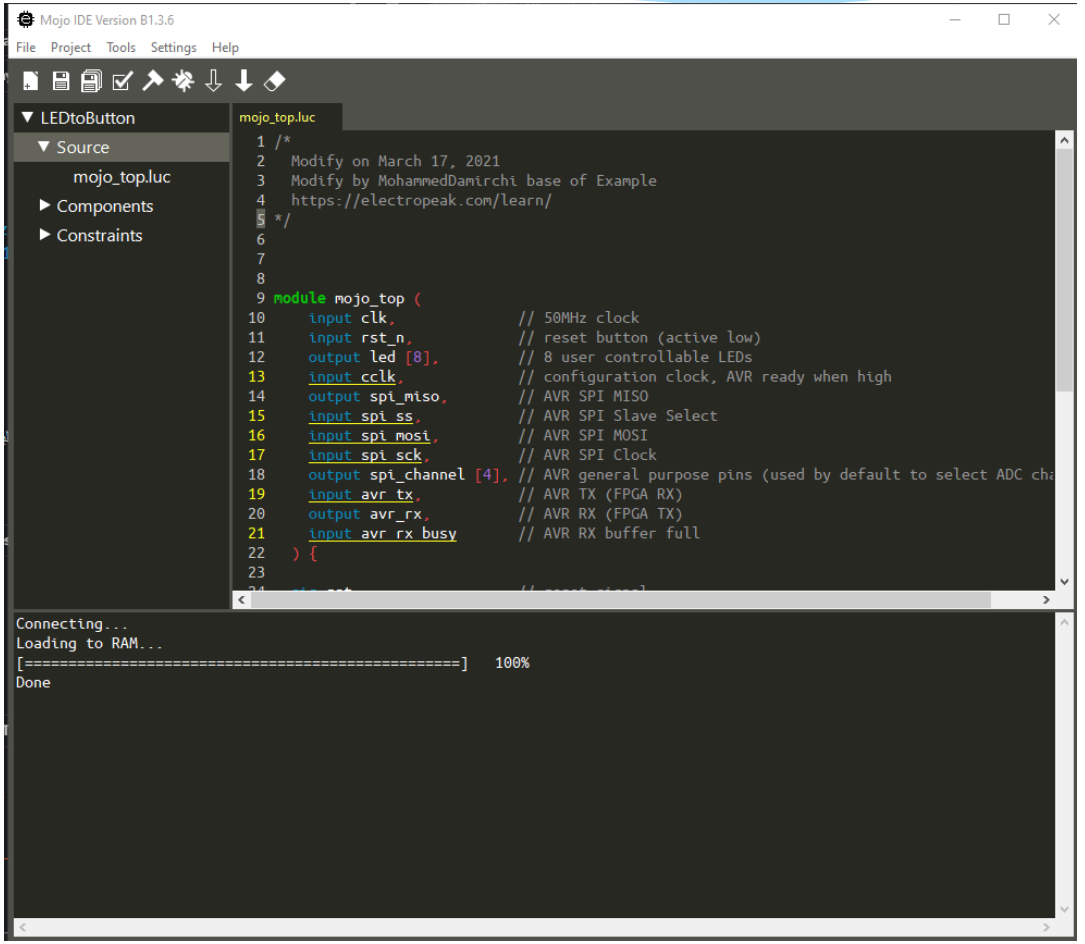
- Step 3: Code
  - Now click on **Build Project** and wait until the project is built.



https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/

Mojo v3 教學資源

* **Mojo v3 教學網**
  - Step 3: Code
    - Now click on **Program Mojo (Temporary)** to program your board.
    - This sample code is temporarily on the **FPGA** and the board will return to the original program by resetting.
    - To program the board permanently, click on **Program Mojo** (**Flash**) so that your program will run it even after if the board is reset.
    - This program is a sample code for testing the FPGA Mojo 3 development board, which by pressing the reset button, the embedded LED on the board turns on, and by releasing the reset button, the LED turns off again



https://electropeak.com/learn/a-complete-beginners-guide-to-mojo-v3-spartan-6-fpga-module/

Hardware Description Language

* 硬體描述語言（HDL Hardware Description Language）

  - HDL 是描述數位電路和設計數位系統的語言
  - 設計者利用HDL語言來描述自己的設計想法，利用電子設計自動化(EDA)工具進行模擬。
  - 再用ASIC或FPGA實現其功能，常見有Verilog和VHDL。

VHDL

```
library IEEE;
use IEEE.STD_Logic_1164, all;

entity LATCH_IF_ELSEIF is
  port (En1, En2, En3, A1, A2, A3: in std_logic;
        Y: out std_logic);
end entity LATCH_IF_ELSEIF;

architecture RTL of LATCH_IF_ELSEIF is
begin
   process (En1, En2, En3, A1, A2, A3)
   begin
      if (En1 = '1') then
        Y <= A1;
      elseif (En2 = '1') then
        Y <= A2;
      elseif (En3 = '1') then
        Y <= A3;
      end if;
    end process;

end architecture RTL;
```

Verilog

```
module LATCH_IF_ELSEIF  (En1, En2, En3, A1, A2, A3,Y);
    input En1, En2, En3, A1, A2, A3;
    output Y;

    reg Y;

    always @(En1 or En2 or En3 or A1 or A2 or A3)
       if (En1 == 1)
          Y =A1;
     else if (En2 == 1)
          Y = A2;
     else if (En3 == 1)
          Y = A3;

end module
```

https://encyclopedia2.thefreedictionary.com/VHSIC+hardware+description+language

Verilog 介紹

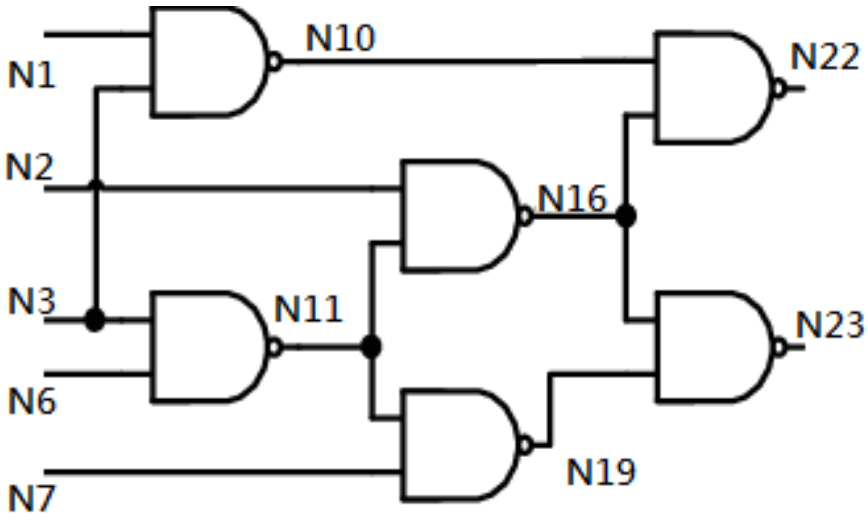
* Verilog HDL是一種硬體描述語言(HDL: Hardware Description Language)
* Verilog HDL和VHDL是最受歡迎的兩種硬體描述語言
* 2009年，IEEE 1364-2005和IEEE 1800-2005兩個部分合併為IEEE 1800-2009，成為了一個新的、統一的SystemVerilog硬體描述驗證語言（hardware description and verification language, HDVL）

https://zh.wikipedia.org/wiki/Verilog

Verilog 介紹

* 軟體是循序的，而硬體是並行的

電路



Verilog

```
module c17(N1, N2, N3, N6, N7, N22, N23);
input  N1, N2, N3, N6, N7;
output N22, N23;
wire N10, N11, N16, N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

由上圖得知，N1、 N2、 N3、 N6和N7並沒有先後之分，是並行的。

https://www.cnblogs.com/oomusou/archive/2008/06/17/c_verilog_mental_thinking.html
https://www.researchgate.net/figure/C17-schematic-and-structural-verilog-netlist_fig3_220405407

Verilog 介紹

* **硬體要循序，要靠clock和FSM**
  - 『我的演算法就是要循序一步一步的做，如C語言那樣，那怎麼辦?』，若Verilog要這樣，就得靠clock並且搭配FSM，當一個state完成後，進入下一個state，這樣就能依照clock的進行，而達成循序的要求。

* **Verilog 程式碼沒有先後之分**
  - 除了blocking assignment有先後執行順序，nonblocking assignment同時執行外，Verilog的程式沒有前後順序之分

https://www.cnblogs.com/oomusou/archive/2008/06/17/c_verilog_mental_thinking.html

Verilog 介紹

* Verilog是一種大小寫敏感的硬體描述語言，其中它的所有系統關鍵字都是小寫

* 設計人員可以設計一個頂層模組，通過實例呼叫模組的方式來進行測試。這個頂層模組常被稱為「測試平台（Testbench）」

https://www.cnblogs.com/oomusou/archive/2008/06/17/c_verilog_mental_thinking.html
https://zh.wikipedia.org/wiki/Verilog

Verilog介紹

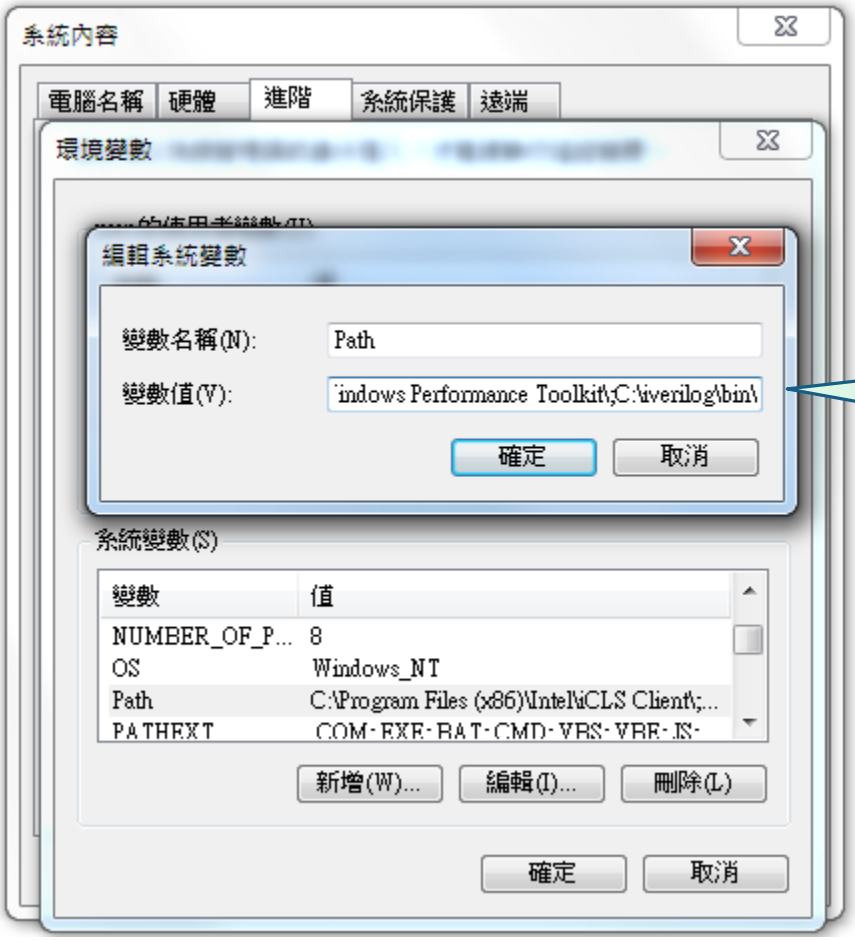
* 多用模擬器觀察自己寫的code，以下為常見模擬器
  - 免費
    - Icarus Verilog (Windows) : http://bleyer.org/icarus/
    - VeriWell https://sourceforge.net/projects/veriwell/
  - 商用
    - ModeSim是一個能編輯及看模擬圖形的軟體，學生版填寫資料可下載License。

Verilog模擬器 - Icarus Verilog

* 設定PATH 系統變數，請設定環境變數在變數Path加入 ";C:\iverilog\bin\;C:\iverilog\gtkwave\bin\"
  - Windows 10 和 Windows 8
    1. 在「搜尋」中，搜尋並選取：系統 (控制台)
    2. 按一下**進階系統設定**連結。
    3. 按一下**環境變數**。在**系統變數**區段中，找到 PATH 環境變數並加以選取。按一下**編輯**。如果 PATH 環境變數不存在，請按一下新增。
    4. 在**編輯系統變數** (或**新增系統變數**) 視窗中，指定 PATH 環境變數的值。按一下**確定**。按一下**確定**，將其餘的視窗全都關閉。
  - Windows 7
    1. 在桌面的**電腦**圖示按一下滑鼠右鍵。
    2. 從內容功能表中選擇**內容**。
    3. 按一下**進階系統設定**連結。
    4. 按一下**環境變數**。在**系統變數**區段中，找到 PATH 環境變數並加以選取。按一下**編輯**。如果 PATH 環境變數不存在，請按一下新增。
    5. 在**編輯系統變數** (或**新增系統變數**) 視窗中，指定 PATH 環境變數的值。按一下**確定**。按一下**確定**，將其餘的視窗全都關閉。

Verilog模擬器 - Icarus Verilog

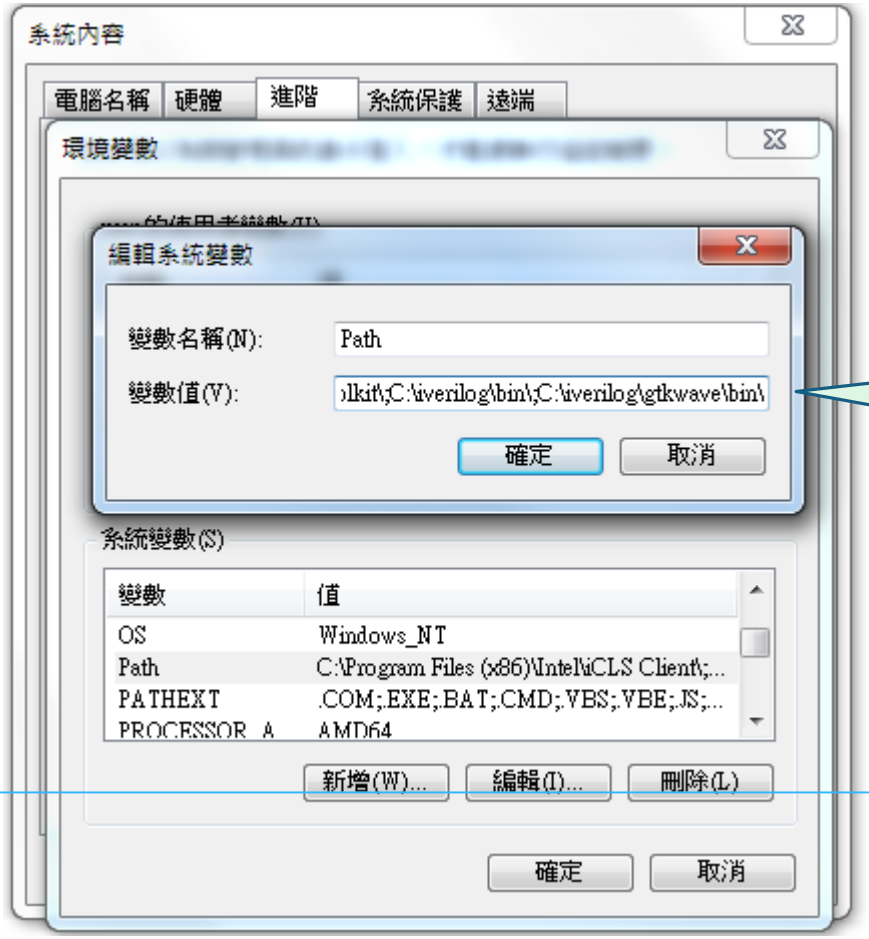
- **Windows 7**



在變數名稱 Path 的變數值最後加上 ;C:\iverilog\bin\;C:\iverilog\gtkwave\bin\

Verilog模擬器 - Icarus Verilog

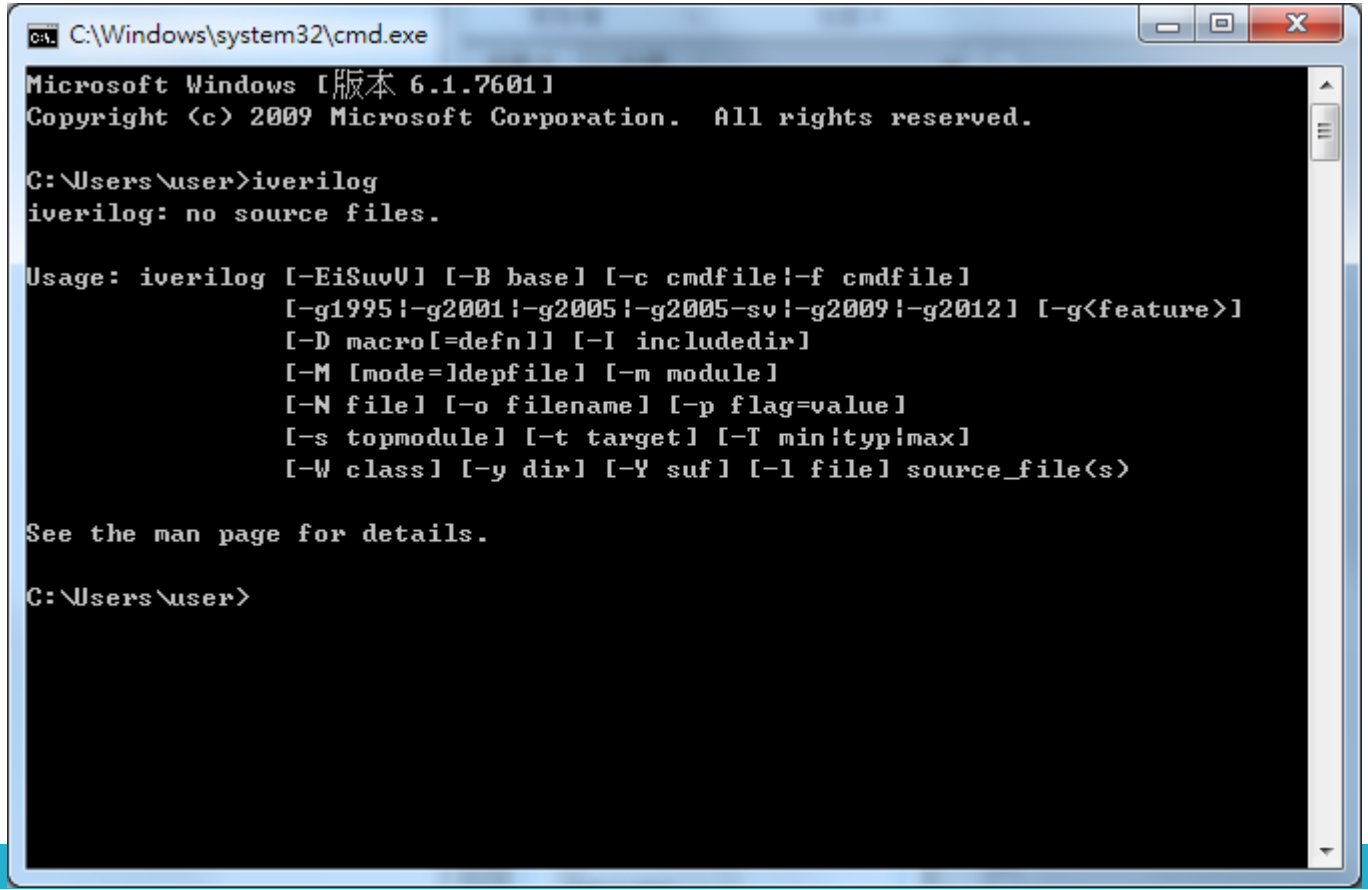
- Windows 7



在變數名稱 Path 的變數值最後加上 ;C:\iverilog\bin\;C:\iverilog\gtkwave\bin\

Verilog模擬器 - Icarus Verilog

* 執行 Icarus Verilog
  - 安裝成功後，cmd 執行命令提示字元，鍵入"iverilog"會跑出類似下面的資訊。



Verilog模擬器 - Icarus Verilog

* 在目錄"C:\iverilog\test"加入二個.v檔。

simple.v

```
module simple(A, B,CIN,COUT,SUM);

  input   A,B,CIN;
  output  COUT,SUM;


  assign {COUT,SUM}=A+B+CIN;

endmodule
```

https://sites.google.com/site/verilog710/xiang-guan-gong-ju/icarus-verilog

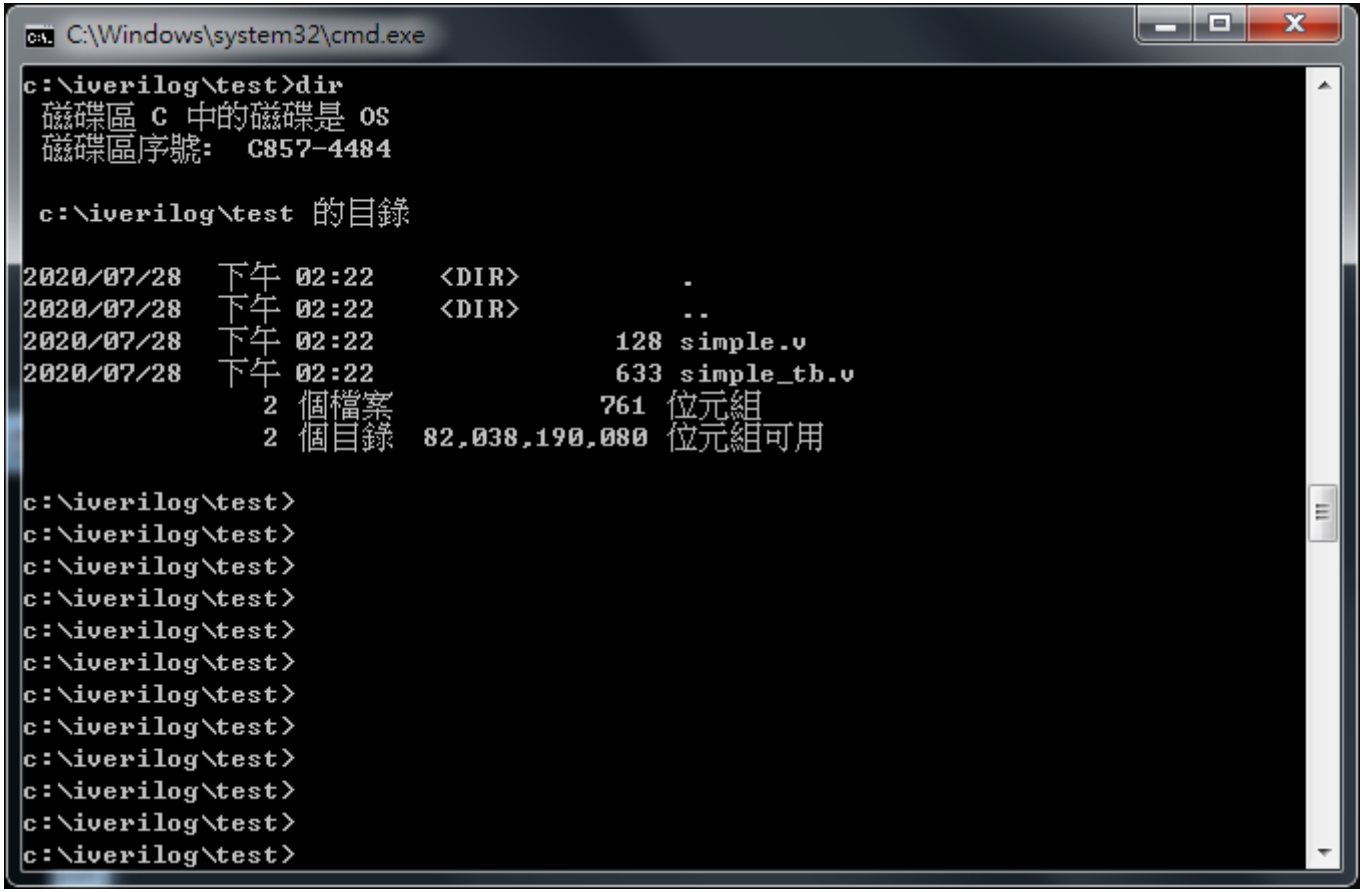
電路測試碼(test bench)

simple_tb.v

```
module simple_tb;

  reg   A,B,CIN;
  wire  COUT,SUM;

  initial
   begin
     $dumpfile("simple.vcd");
     $dumpvars(0, s);
     $monitor("A = %b, B = %b CIN = %b | COUT = %b SUM = %b ", A, B,CIN,COUT,SUM);
     #50 A = 1'b0; B=1'b0;CIN=1'b0;
     #50 A = 1'b0; B=1'b1; CIN=1'b0;
     #50 A = 1'b1;  B=1'b0;CIN=1'b0;
     #50 A = 1'b1;  B=1'b1; CIN=1'b0;
     #50 A = 1'b0; B=1'b0;CIN=1'b1;
     #50 A = 1'b0; B=1'b1; CIN=1'b1;
     #50 A = 1'b1;  B=1'b0;CIN=1'b1;
     #50 A = 1'b1;  B=1'b1; CIN=1'b1;
     #50 $finish;
   end

  simple s(A, B,CIN,COUT,SUM);
endmodule
```

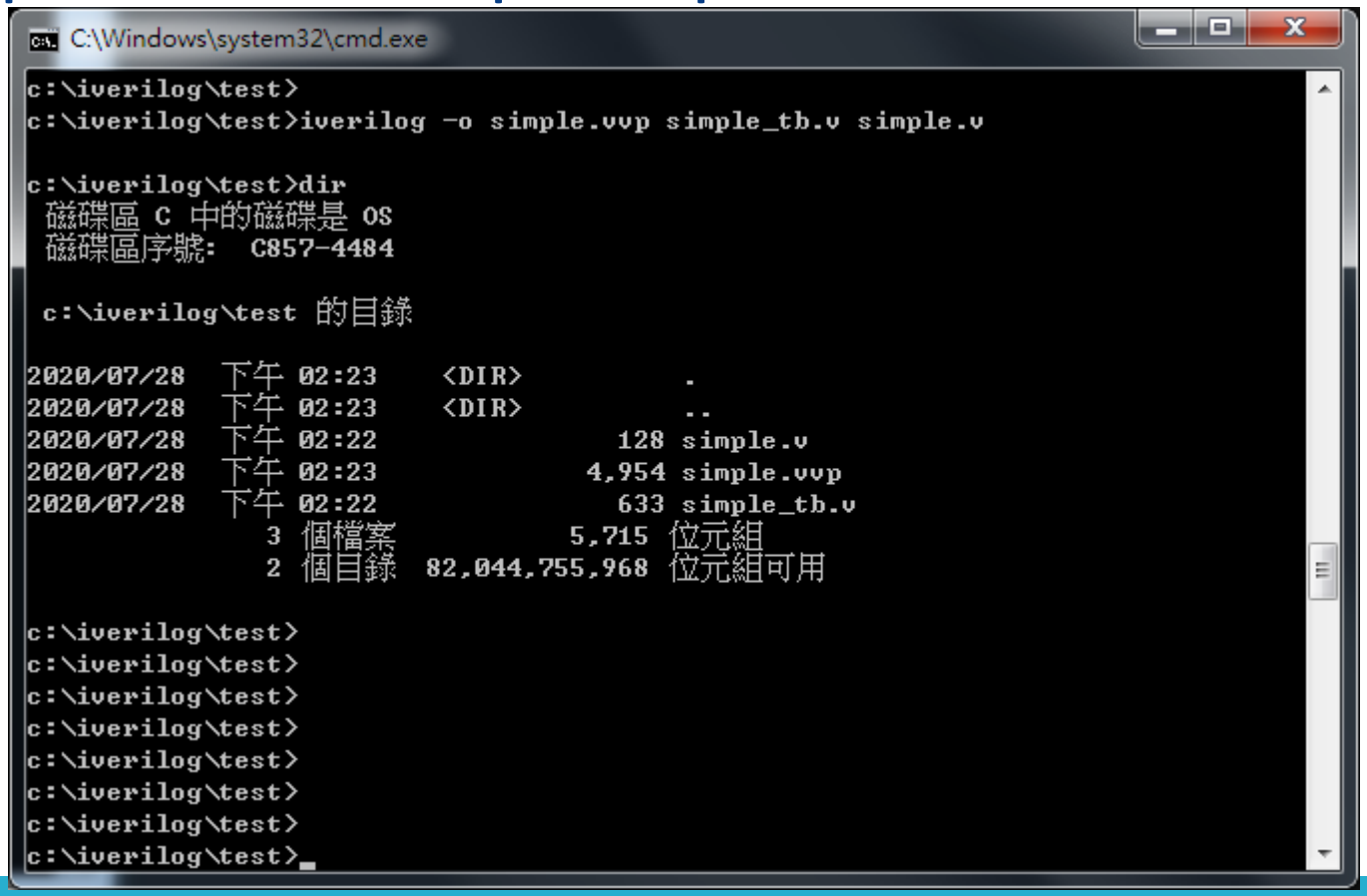
Verilog模擬器 - Icarus Verilog

* 在 c:\iverilog\test 鍵入“dir“ 有二個檔案 simple.v及simple_tb.v



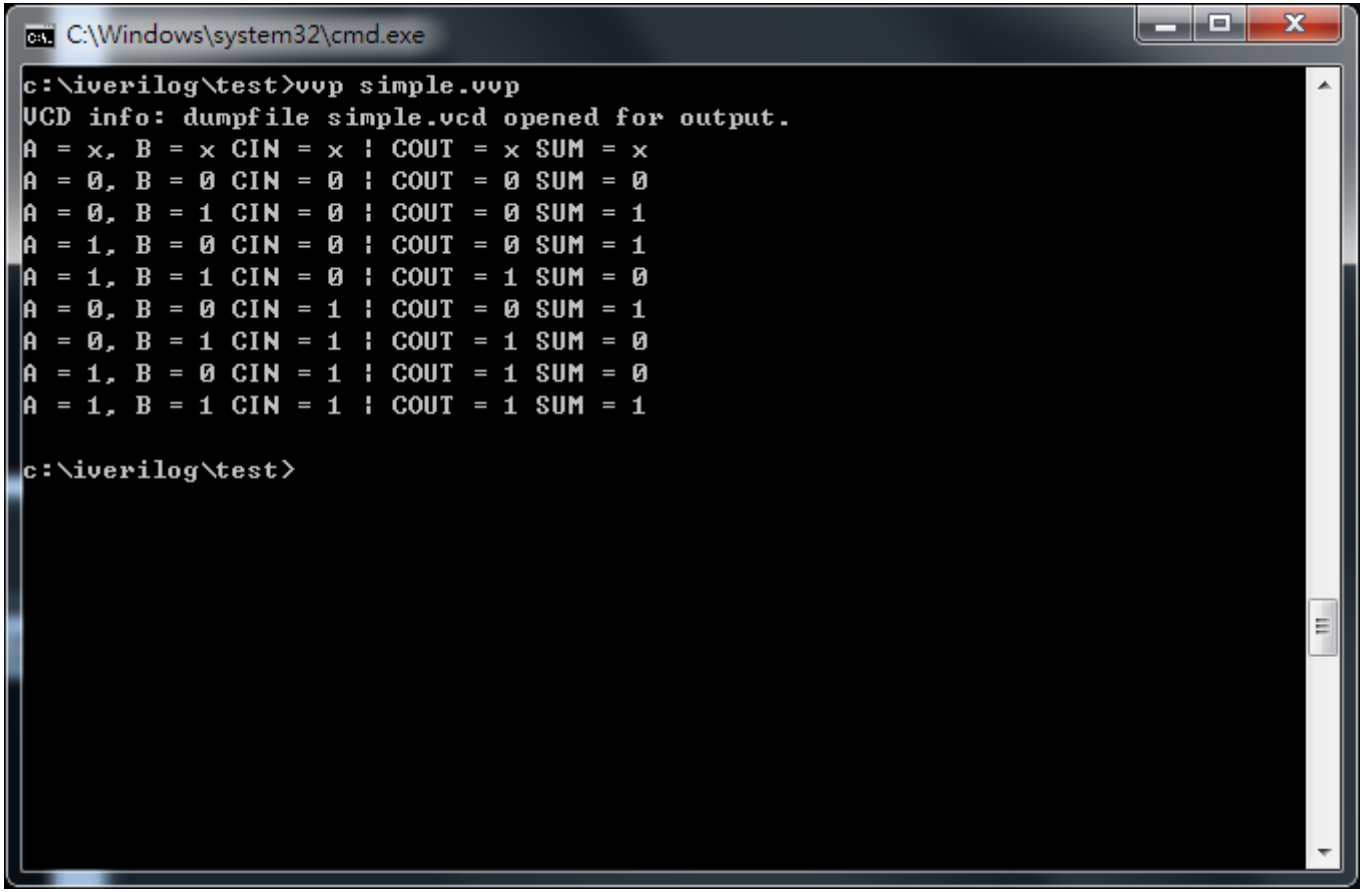
Verilog模擬器 - Icarus Verilog

* 鍵入"iverilog -o simple.vvp simple_tb.v simple.v"產生simple.vvp



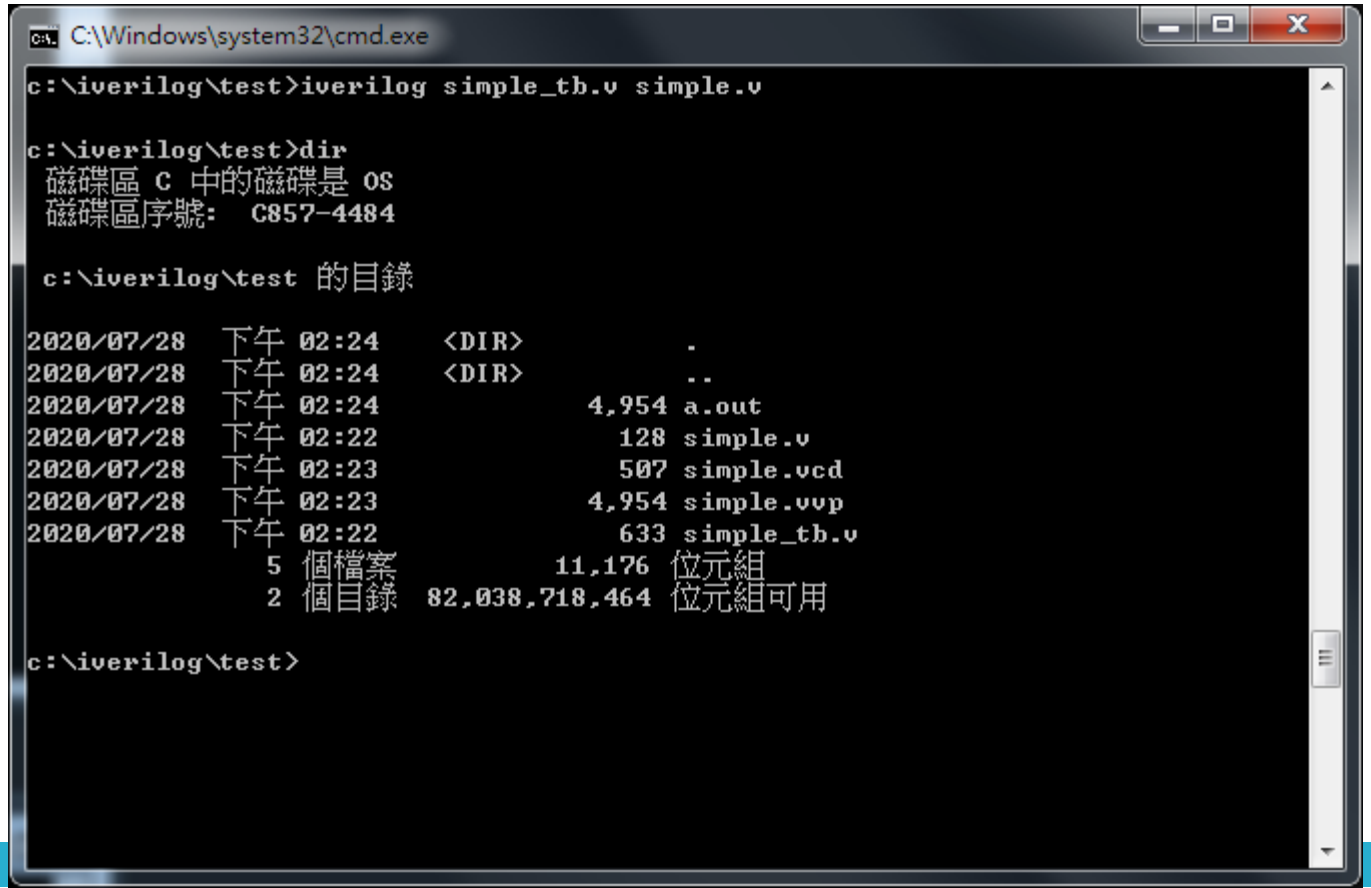
Verilog模擬器 - Icarus Verilog

* 鍵入vvp simple.vvp 即可執行



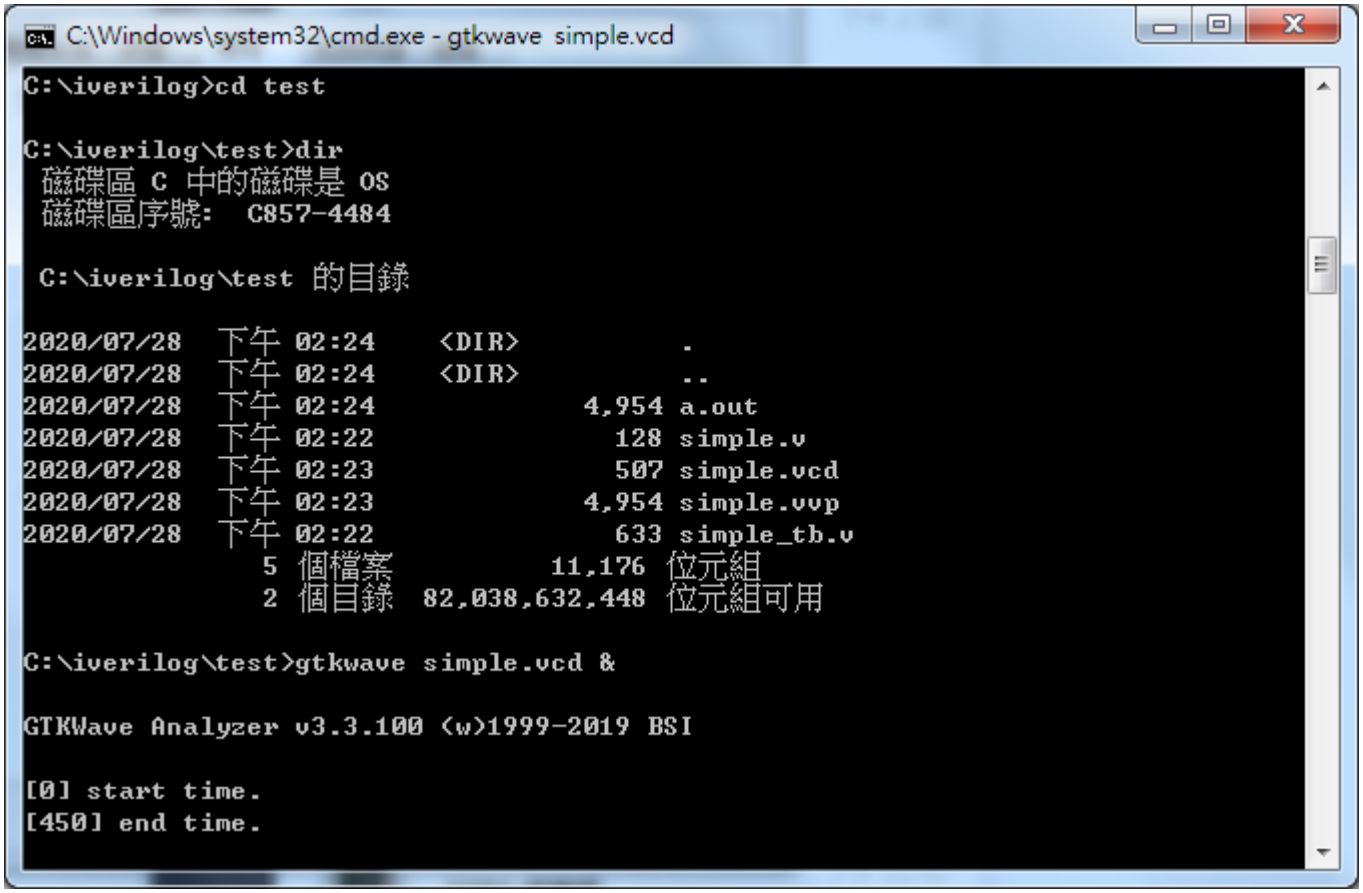
Verilog模擬器 - Icarus Verilog

* 觀察模擬圖形鍵入"iverilog simple_tb.v simple.v"產生.vcd檔



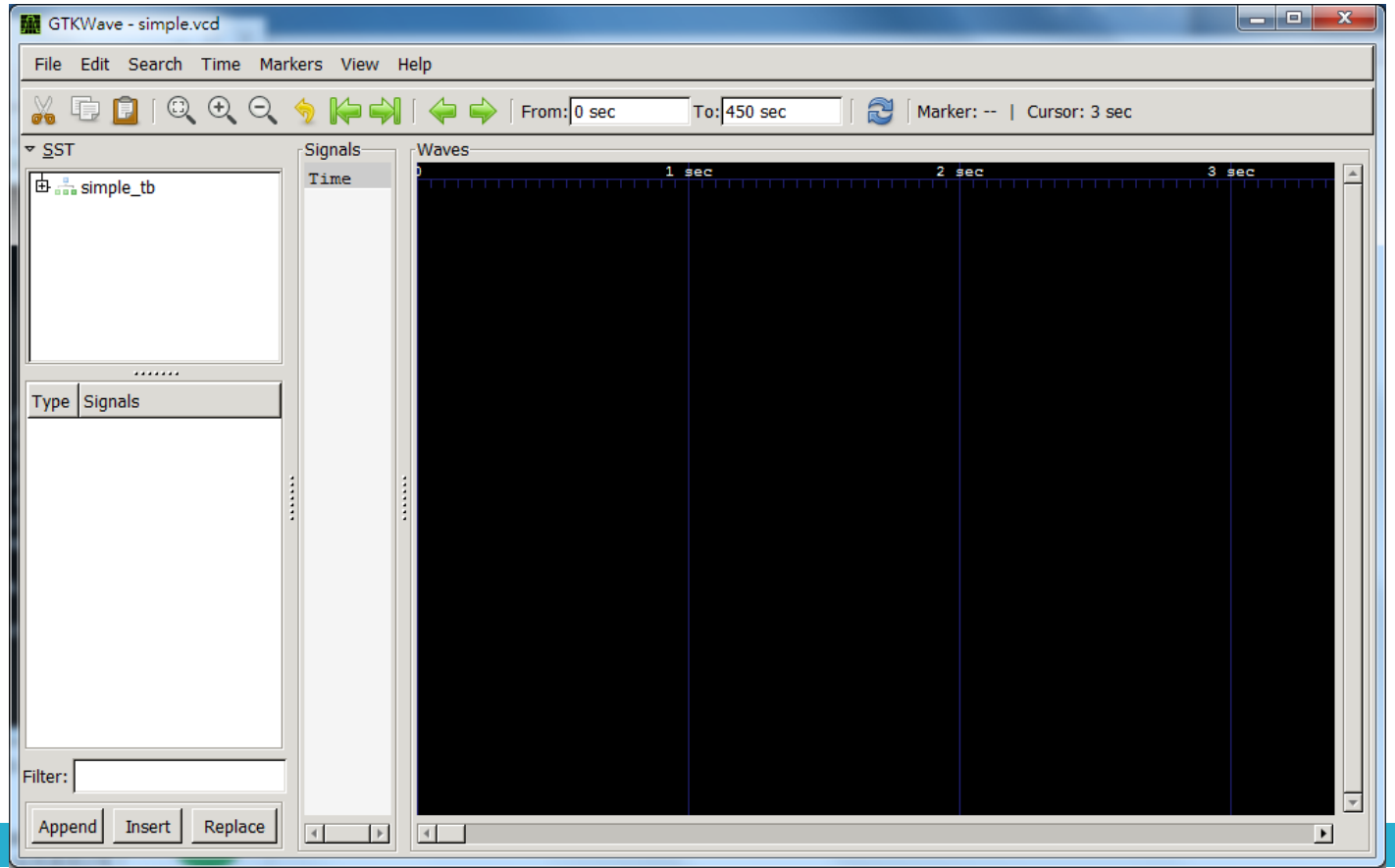
Verilog模擬器 - Icarus Verilog

* 開啟觀看波形軟體鍵入gtkwave simple.vcd &



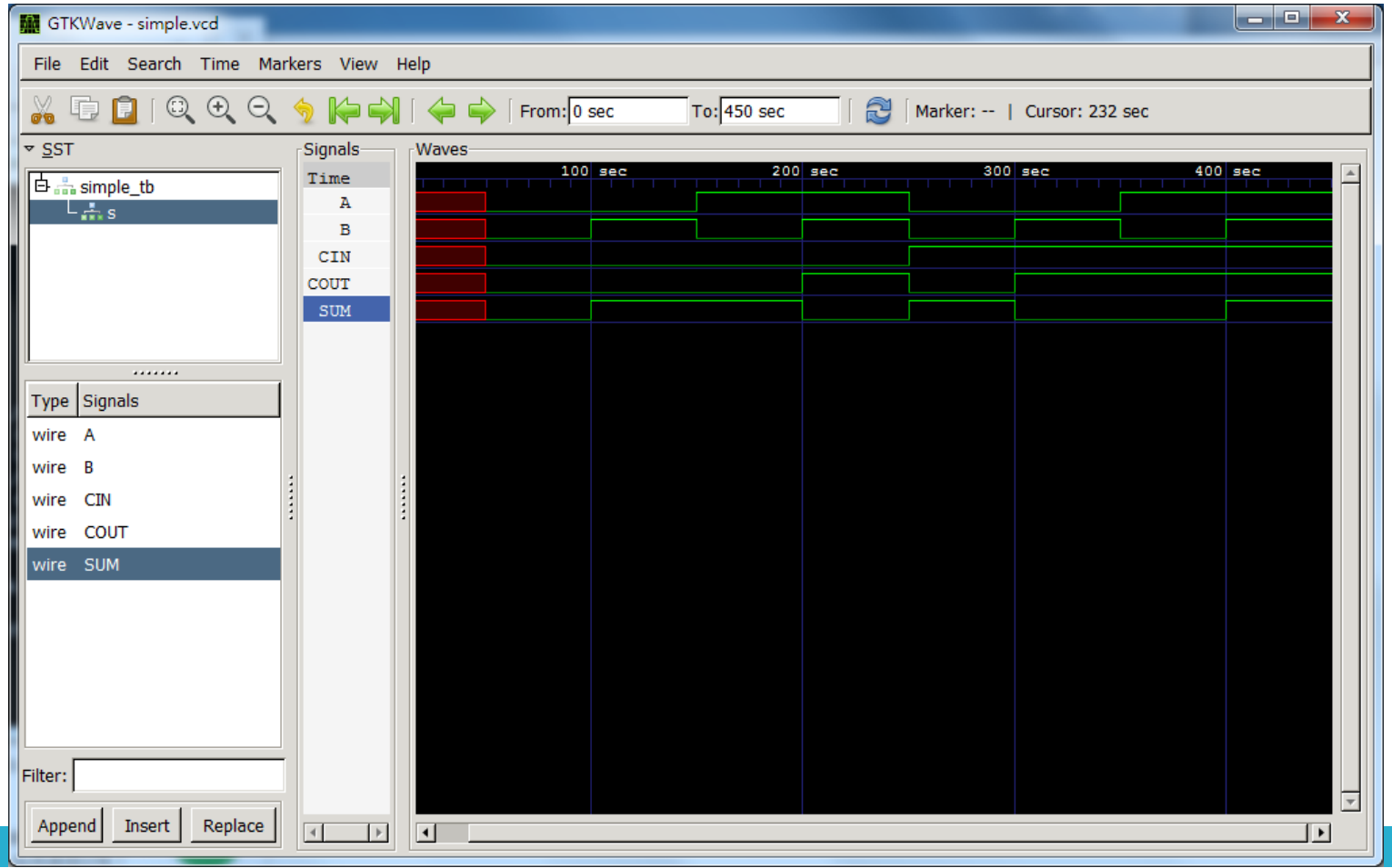
Verilog模擬器 - Icarus Verilog

* *GTKWave* 採用GTK+ 函式庫設計出來的一個開放原始碼的波形顯示工具



Verilog模擬器 - Icarus Verilog

* *GTKWave* 可以讀取FST, LXT, LXT2, VZT, GHW, VCD/EVCD 等檔案格式



Verilog模擬器 - ModelSim

* *http://www.oldfriend.url.tw/page52.htm*

Verilog 基本語法

* module
  - Verilog 主要的架構就是模組（module）
  - 模組被包含在關鍵字module、endmodule之內
  - 每一個 Verilog 檔案，必須包含一個模組module

```
module 模組名稱( 輸出入埠名稱 );
   … … … …
endmodule
```

```
module 模組名稱( In1, In2, Out1, Out2, InOut1 );
   … … … …
endmodul
```

* 與 C 語言相同，有 // 及 /**/ 兩種註解方式

Verilog 基本語法

module 模組名稱(輸出入埠名稱);

輸出入埠 敘述

資料型態 敘述

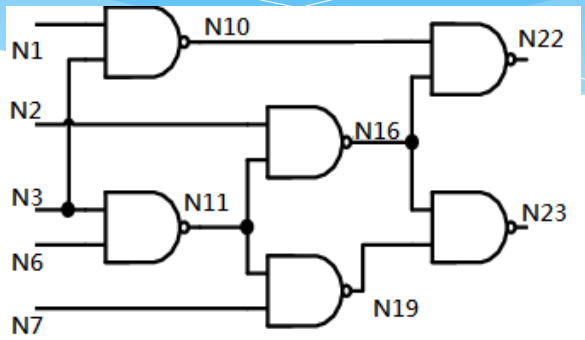
內部電路 敘述

endmodule

```
module c17(N1, N2, N3, N6, N7, N22, N23);
input   N1, N2, N3, N6, N7;
output N22, N23;
wire N10, N11, N16, N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

Verilog 基本語法

```
module <name> // 模組名稱
 parameter ...    // 參數宣告
 port ...         // 腳位宣告
 wire ...         // 線宣告
 reg  ...         // 暫存器宣告

 initial begin    // 初始化設定區塊
 end

 assign ...       // 資料處理層級之描述

 ...              // 引用較低階模組別名

 always begin // 行為層級之描述區塊
              // 資料處理與指定等描述
              // task與function的使用
 end

 function       // 函數宣告
 task           // 作業宣告
endmodule
```




```
module c17(N1, N2, N3, N6, N7, N22, N23);
input  N1, N2, N3, N6, N7;
output N22, N23;
wire N10, N11, N16, N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

Primitives - Verilog 資料型態

* 資料狀態 (四值邏輯（four-valued logic）)

值	意義	說明
0	邏輯0/低電位	布林代數 - 假值
1	邏輯1/高電位	布林代數- 真值
z或Z	高阻抗(High Impendence)	三態緩衝器的輸出，高阻抗斷線
x或X	未知的值 (Unknow)或浮接(Floating)	像是線路未初始化之前，以及有0,1 兩者衝突的線路值，或者是輸入為 Z 的輸出值

https://www.hdlworks.com/hdl_corner/verilog_ref/

Primitives - gate_type

元件	反元件	說明
and	nand	邏輯元件 AND/NAND
or	nor	邏輯元件 OR/NOR
xor	xnor	邏輯元件 XOR/XNOR
buf	not	緩衝器(buffer)/反相器 (inverter)
bufif0	notif0	三態緩衝器 (0 代表連通)
bifif1	notif1	三態緩衝器 (1 代表連通)
pullup	pulldown	上拉（Pull Up）或下拉（Pull Down）電阻（兩者統稱爲“拉電阻”）

Primitives - gate_type

Syntax:

gate_type [(strength)] [#(delay)] [instance_name] [instance_range] (terminal, terminal, ...);

Name	Gate Type	Terminals
Logic	**and, nand, or, nor, xor, xnor**	Output, Input(s)
Buffer and inverter	**buf, not**	Output(s), Input
Tristate logic	**bufif0, bufif1, notif0, notif1**	Output, Input, Enable
Pullup and pulldown	**pullup, pulldown**	Output

Example:

```
and u1 (Q, A, B);
and #(2.1, 2.8) u2 (Q, A, B);
and (pull0, strong1) (Q, A, B);
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Primitives - gate_type 包含延遲的語法

* **Example:**
  - buf a(x, y); // 無延遲
  - buf #1 b(x, y); // 延遲 1 單位時間
  - buf #(2,3) c(x, y); // 時間 2 時升起 (rise)，3 時落下 (fall)
  - buf #(1:2:3) d(x, y); // 1:最短延遲 (min), 2:典型延遲 (typical), 3:最長延遲 (max)

Primitives - switch_type

Syntax:

switch_type [#(delay)] [instance_name] [instance_range] (terminal, terminal, ...);

Name	Switch Type	Terminals
MOS	**nmos, pmos, rnmos, rpmos**	Output, Input, Enable
CMOS	**cmos, rcmos**	Output, Input, N-Enable, P-Enable
Bidirectional pass	**tran, rtran**	Inout1, Inout2
Bidirectional pass with control	**tranif0, tranif1, rtranif0, rtranif1**	Inout1, Inout2, Control

https://www.hdlworks.com/hdl_corner/verilog_ref/

Lexical Conventions -Verilog Numbers

* 數值

<正負號><位元長度>'<進制表示><數值資料>

- 位元長度：以十進制表示位元長度(二進制長度)
- 進制表示： 二進制(b)、八進制(o)、十進制(d)、十六進制(h)，預設為十進制
- 數值資料：可用底線'_'來增加可讀性，數值內也可以混用X和Z

Lexical Conventions - Verilog Numbers

* 數值

Integer: [sign] [size] ['base] value
Real: [sign] value.value | [sign] baseEexponent

sign = + | -

Base	Symbol	Legal values
Binary	b or B	0, 1, x, X, z, Z, ?, _
Octal	o or O	0 - 7, x, X, z, Z, ?, _
Hexadecimal	h or H	0 - 9, a - f, A - F, x, X, Z, Z, ?, _
Decimal (default)	d or D	0 - 9, _

Lexical Conventions - Verilog Numbers

* 數值

<位元長度>’<進制表示><數值資料>

- Num = 5’b01101 // 二進制
- Num = 1'b1 // 二進制
- Num = 22; // 十進制
- Num = 12’b0000_1111_0000; // 增加可讀性
- Num = 4’hf; // 十六進制(二進制的1111)
- Num = 4’bxxx1; // 前三位為未知，最後為1
- Num = 4’bz01; // 前兩位為z，後兩位為01
- Num = -0.5 // 十進制， 負數
- Num = 5.8E3 // 十進制，實數
- Num = 2e-4 // 十進制，實數

Lexical Conventions - Verilog String

* 字串

Character	Meaning
\n	New line
\t	Tab
\\	Backslash
\"	Double quote
\ddd	Octal code
%%	Percent sign

Example:

"This is the first line\n This is the second line"
reg [11*8:1] R1;
R1 = "Hello World";

Lexical Conventions - Verilog訊號強度

* 訊號強度（從強到弱）
  - 如果兩個具有不同強度的信號驅動同一個線網，則競爭結果為高強度信號的值。
  - 如果兩個強度相同的信號之間發生競爭，則結果為不確定值。
  - **cap_strength = large | medium | small , The cap_strength is for trireg nets only.**

Strength level	Name	Keyword
7	Supply drive	**supply0 , supply1**
6	Strong drive	**strong0 , strong1**
5	Pull drive	**pull0 , pull1**
4	Large capacitive	**large**
3	Weak drive	**weak0 , weak1**
2	Medium capacitive	**medium**
1	Small capacitive	**small**
0	High impedance	**highz0 , highz1**

https://read01.com/aAoEkO.html

https://www.hdlworks.com/hdl_corner/verilog_ref/items/Strengths.htm

Lexical Conventions - Verilog訊號強度範例

* 訊號強度範例（從強到弱）
  - 兩個簡單的模塊，兩個模塊都是輸出，然後輸出接在一起。
  - 第一個模塊是輸出強1，弱0.
  - 第二個模塊輸出弱1，強0。

```
module circuit1(a);

output a;

assign (suppy1, weako)a = 1;

endmodule
```

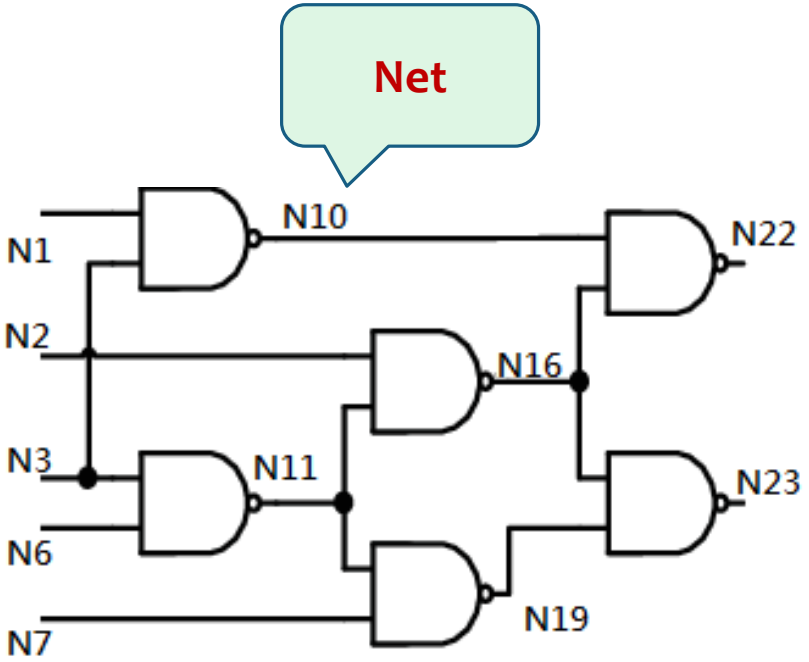
```
module circuit2(b);

output b;

assign (weak1, weako)b = 0;

endmodule
```

Data Types - Verilog 變數

* Verilog所用到的所有變數都屬於兩個基本的類型：
  - 線網類型 (Net)
    - 線網與我們實際使用的電線類似
    - 線網的數值一般只能通過連續賦值（continuous assignment），由賦值符右側連接的驅動源決定
  - 暫存器類型 (Register)
    - 儲存當前的數值，直到另一個數值被賦值給它

Data Types - Verilog 線網Net

* 線網 Net (wire、wand、wor 、tri、trior、triand、tri0、tri1、supply0、supply1、trireg)
  - Physical wire between devices
  - 組合邏輯
  - 沒有記憶性
  - 無驅動能力
  - 只能在assign左側賦值，不能在always @ 中賦值
  - 能用於assign與always @模組表示式的右側
  - 預設值為 z



* wire → 不能將多條訊號同時接到wire
* wand → a/b輸入訊號做狀態 AND
* wor → a/b輸入訊號做狀態 OR

Data Types - Verilog 線網Net

wire or **tri**	simple interconnecting wire
wor or **trior**	wired outputs OR together
wand or **triand**	wired outputs AND together
tri0	pulls down when tri-stated
tri1	pulls up when tri-stated
supply0	constant logic 0 (supply strength)
supply1	constant logic 1 (supply strength)
trireg	stores last value when tri-stated (capacitance strength)

Example:

```
wire [7:0] Data;
trireg (large) C1;
wire Q = A || B;                                  // continuous assignment
wire [7:0] Array [0:255][0:255][0:255];           // Verilog-2001 Multidimensional array
```


https://www.hdlworks.com/hdl_corner/verilog_ref/

```
module 模組名稱( a, b, c, d, e );

  input a, b;
  output c, d, e;

  wire c;
  wand d;
  wor e;

  // wire接一起 → 錯誤
  assign c = a;
  assign c = b;

  // wire-and → d = a&b
  assign d = a;
  assign d = b;

  // wire-or → e = a|b
  assign e = a;
  assign e = b;

endmodule
```

Data Types - Verilog 暫存器

* 暫存器Register
  - 暫存器類型的變數有以下幾種：reg（普通暫存器）、integer（整數）、time（時間）、real（實數），其中reg作為一般的暫存器使用最為普遍
  - 可以用於組合邏輯或者時序邏輯
  - 有記憶性，能儲存資料
  - 有驅動能力
  - 預設值為x (最好要初始化)
  - 在always @模組表示式左側被賦值
  - 能用於assign與always @模組表示式的右側

https://hom-wang.gitbooks.io/verilog-hdl/content/Chapter_02.html
https://zh.wikipedia.org/wiki/Verilog

Data Types - Verilog 暫存器

reg	unsigned variable of any bit size
integer	signed 32-bit variable
time	unsigned 64-bit variable
real or **realtime**	double-precision floating point variable

https://www.hdlworks.com/hdl_corner/verilog_ref/

Data Types - Verilog 暫存器

Syntax:

```
register_type [ size ] variable_name, variable_name, ...;
register_type [ size ] variable_name = initial_value;
register_type [ size ] memory_name [ array_size ];

// Multi-dimensional array
register_type [ size ] memory_name [ array_size ] [ array_size ] ...;
```

Example

```
reg [7:0] Data;
integer Int;
time Now;
reg [15:0] Memory [0:1023];
reg [7:0] A = 8'h3C;                    // Verilog-2001
reg [7:0] Array [0:255][0:255][0:255];    // Verilog-2001
```

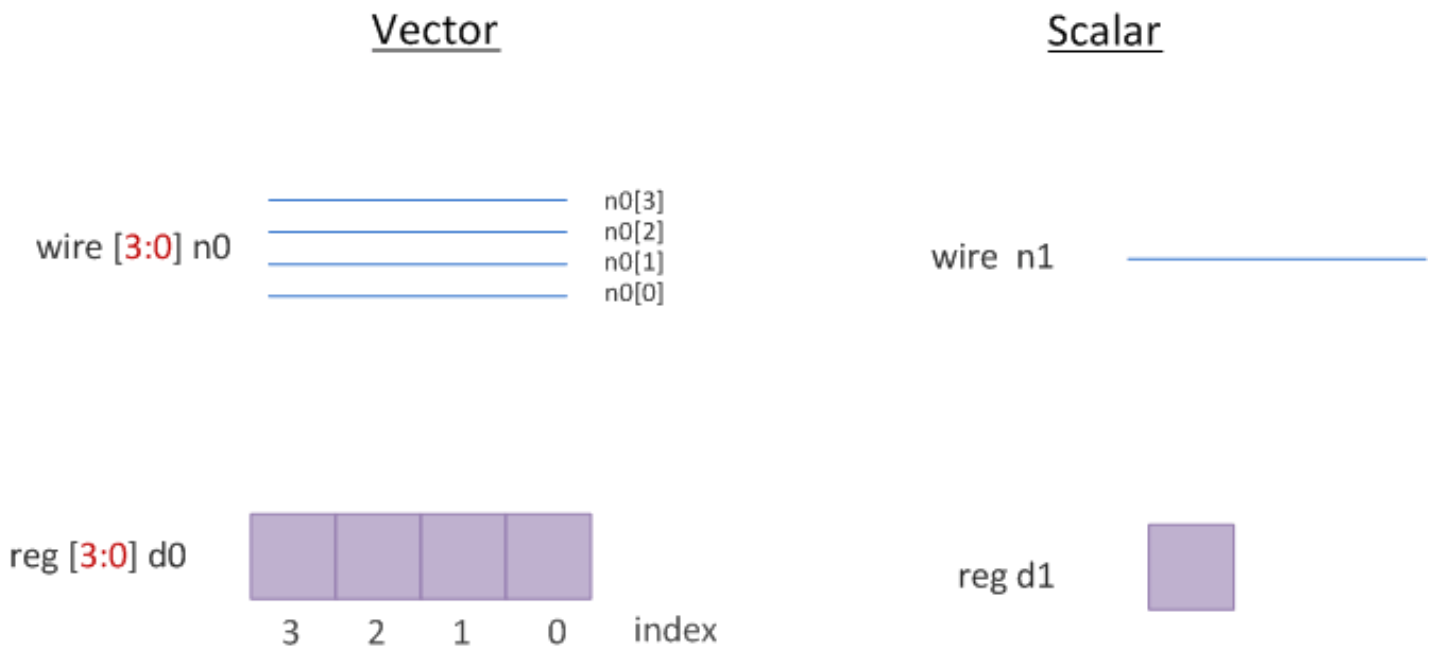
https://www.hdlworks.com/hdl_corner/verilog_ref/

Data Types - Verilog 暫存器

```
module 模組名稱( a, b, c );

   input a;
   output b, c;

   reg b, rTmp;

   // 範例1
   always @(*) begin
      b = a;
   end

   // 範例2
   assign c = rTmp;

endmodule
```

Verilog向量

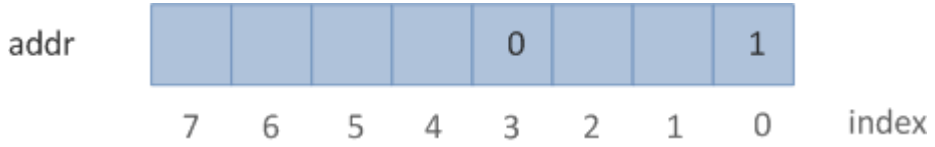
* Verilog中，純量的意思是只具有一個位元的變數，而向量表示具有多個位元的變數



https://hom-wang.gitbooks.io/verilog-hdl/content/Chapter_02.html
https://www.chipverify.com/verilog/verilog-scalar-vector

Verilog向量

* Verilog向量 - **Bit-selects**

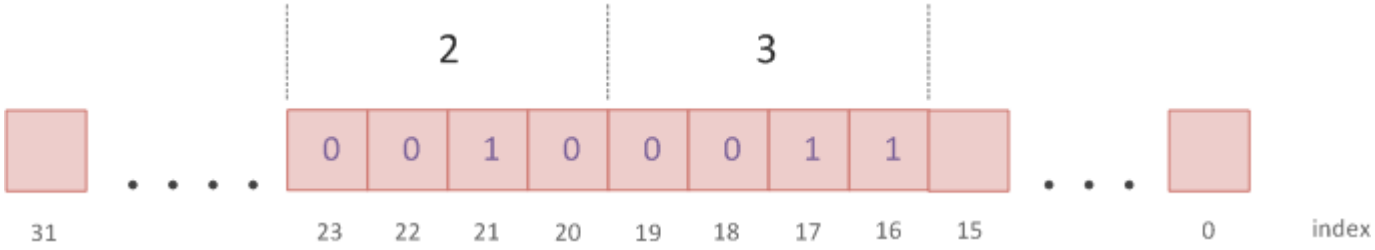



```
reg [7:0] addr;          // 8-bit reg variable [7, 6, 5, 4, 3, 2, 1, 0]

addr [0] = 1;            // assign 1 to bit 0 of addr
addr [3] = 0;            // assign 0 to bit 3 of addr
addr [8] = 1;            // illegal : bit8 does not exist in addr
```

https://hom-wang.gitbooks.io/verilog-hdl/content/Chapter_02.html
https://www.chipverify.com/verilog/verilog-scalar-vector

Verilog向量

* Verilog向量 - **Part-selects**




```
reg [31:0]    addr;

addr [23:16] = 8'h23;        // bits 23 to 16 will be replaced by the new value 'h23 -> constant part-select
```

https://hom-wang.gitbooks.io/verilog-hdl/content/Chapter_02.html
https://www.chipverify.com/verilog/verilog-scalar-vector

Verilog陣列

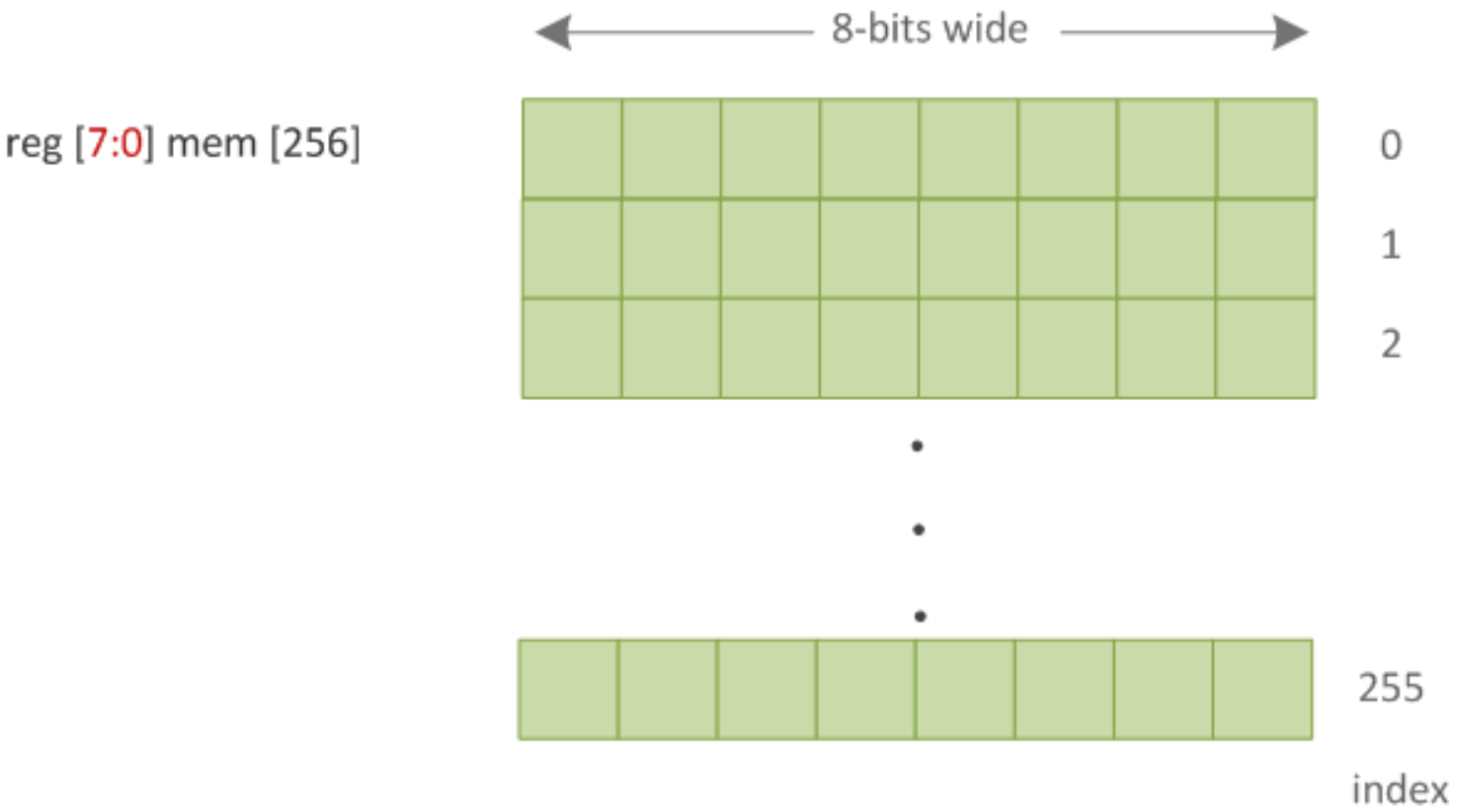
* Verilog陣列
  - 陣列的data types: reg, wire, integer, real
  - HDL只能用於描述一維陣列的表示法，不能描述多維陣列
  - 陣列是多個1位元或若干個位元的元件

```
reg y1 [11:0];          // y is an scalar reg array of depth=12, each 1-bit wide
wire [0:7] y2 [3:0];    // y is an 8-bit vector net with a depth of 4
reg [7:0] y3 [0:1][0:3]; // y is a 2D array rows=2,cols=4 each 8-bit wide

y1 = 0;                 // Illegal - All elements can't be assigned in a single go

y2[0] = 8'ha2;          // Assign 0xa2 to index=0
y2[2] = 8'h1c;          // Assign 0x1c to index=2
y3[1][2] = 8'hdd;       // Assign 0xdd to rows=1 cols=2
y3[0][0] = 8'haa;       // Assign 0xaa to rows=0 cols=0
```

https://www.chipverify.com/verilog/verilog-arrays-memories

Verilog陣列

* Verilog **Memories**



https://www.chipverify.com/verilog/verilog-arrays-memories

Verilog 基本語法

* intput（輸出埠）敘述
  - 模組內可接 net
  - 模組外可接 net、register
* output（輸出埠）敘述
  - 模組內可接 net、register
  - 模組外可接 net
* inout（雙向埠）敘述
  - 模組內可接 net
  - 模組外可接 net

```
module 模組名稱( In1, In2, Out1, Out2, InOut1 );

   input in1, in2;    // 敘述輸出入型態
   output Out1, Out2;
   inout InOut1;

   … … … …

endmodul
```

Expression -Verilog Operators

```
// operator operand
+  -                          // sign
!                             // logical negation
~                             // bitwise negation
&  ~&  |  ~|  ^  ~^  ^~       // reduction (~^ and ^~ are equivalent)

// operand operator operand
+  -  *  /  **                // arithmetic
%                             // modulus
>  >=  <  <=                  // comparison
&&  ||                        // logical
==  !=                        // logical equality
===  !===                     // case equality
&  |  ^  ^~  ~^               // bitwise (^~ and ~^ are equivalent)
<<  >>  <<<  >>>              // shift

// miscellaneous operators
?  :                          // Sel ? M : N;  if Sel is true, M: else N
{}                            // {M, N};  concatenate M to N
{{}}                          // {N{M}};  replicate M N-times
->                            // ->M;  trigger an event on an event data type
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Expression -Verilog Operator Precedence

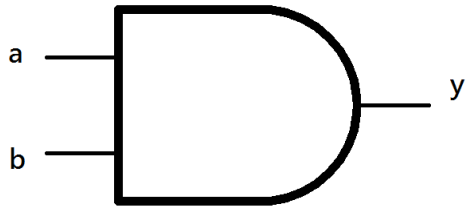
```
+ - ! ~ (unary)      // highest precedence
* / % **
+ - (binary)
<< >> <<< >>>
< <= > >=
== != === !==
& ~&
^ ~^
| ~|
&&
||
?:                   // lowest precedence
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog綜合練習 - Icarus Verilog

* 邏輯 AND

and.v

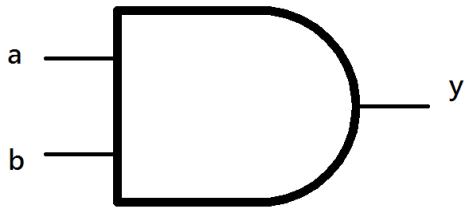


a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

```
module and_example( a, b, y);
  input a, b;
  output y;

  assign y = a & b;

endmodule
```

Verilog綜合練習 - Icarus Verilog

* 邏輯 AND



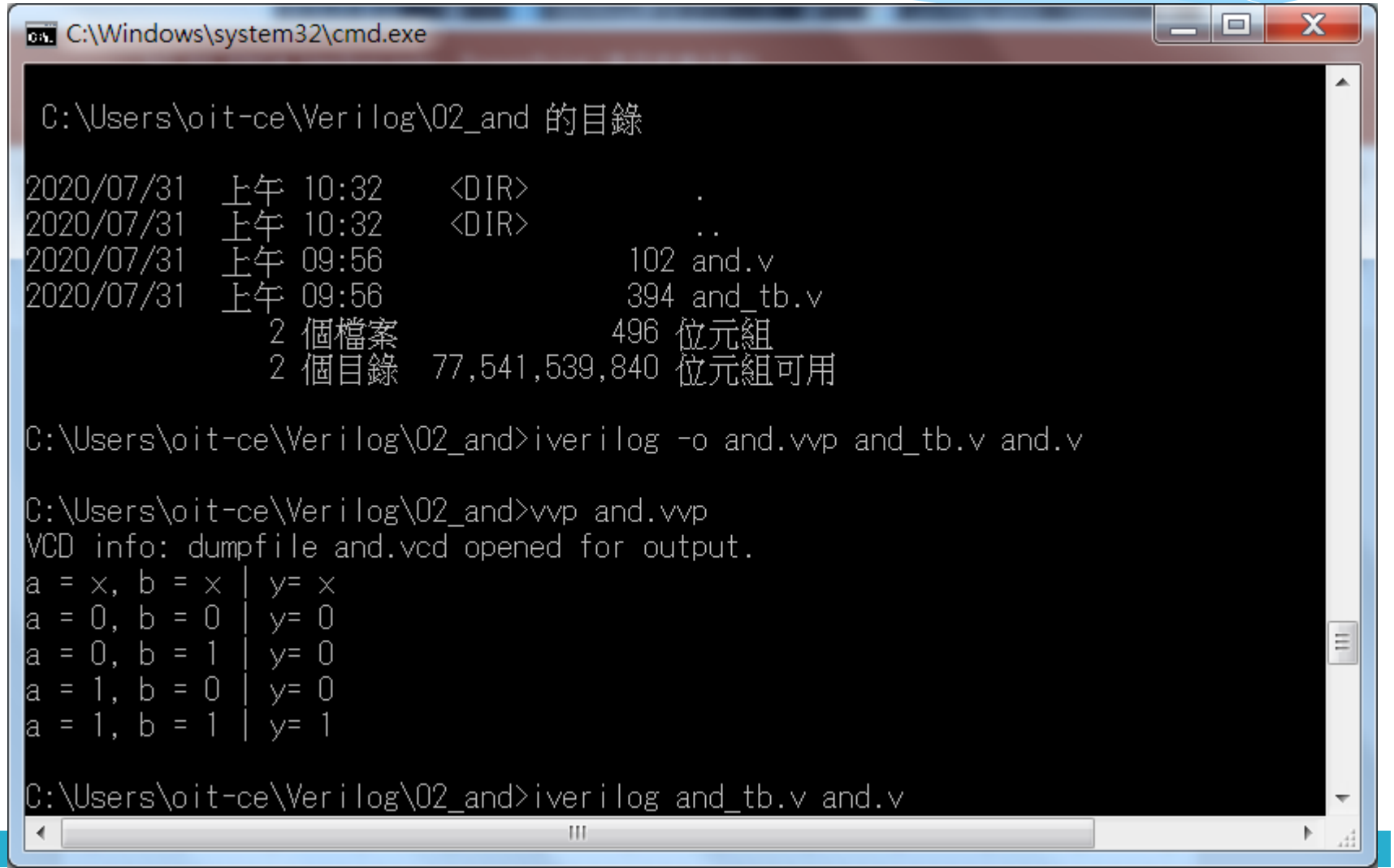
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

and_tb.v

```
module and_example_tb;

  reg  a, b;
  wire y;

  initial
   begin
     $dumpfile("and.vcd");
     $dumpvars(0, s);
     $monitor("a = %b, b = %b | y= %b ", a, b, y);
     #50 a = 1'b0; b=1'b0;
     #50 a = 1'b0; b=1'b1;
     #50 a = 1'b1;  b=1'b0;
     #50 a = 1'b1;  b=1'b1;
     #50 $finish;
   end

  and_example s(a, b, y);
endmodule
```

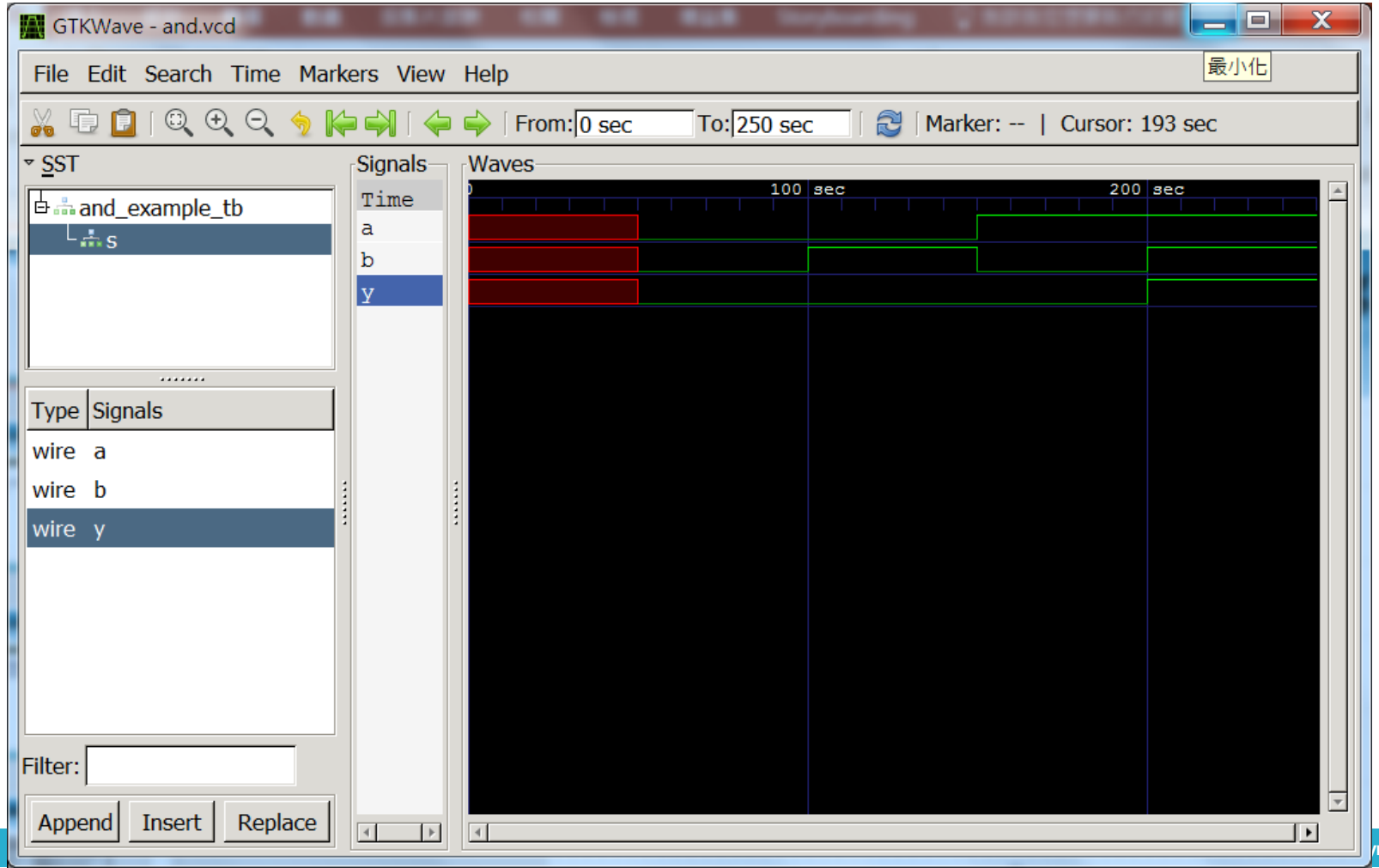
Verilog綜合練習 - Icarus Verilog

* 邏輯 AND



Verilog綜合練習 - Icarus Verilog

* 邏輯 AND
  - gtkwave and.vcd &



Verilog 全域變數

* Verilog 中所有的變數都是全域變數，只要加上模組名稱就可直接存取。

```
module circuit1;

integer myglobalvar;

endmodule
```

```
module circuit2;

initial

$display(circuit1.myglobalvar);

endmodule
```

Verilog Behavioural Modelling

* **always**
  - 敘述的觀念如監督程式一般，隨時監看著輸出入埠訊號的變化，然後告知模組內部進行相關的處理

* **always@(...)**
  - 括弧內的運算式稱之為事件運算式 (event expression)

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

* **always**
  - **The always procedural block statement is executed repeatedly throughout the simulation.**

Syntax:

```
always [ sensitivity_list ]
  statement
```

Example:

```
always #10 Clk = !Clk;

always @(posedge Clk or negedge Reset)
begin
  if (!Reset)
    Q <= 0;
  else
    Q <= D;
end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **always - Sensitivity list**
  * **The sensitivity list controls when the statements in an always block are evaluated.**

Syntax:

@([edge] signal [**or** [edge] signal] …)
@([edge] signal [, [edge] signal] …)
@*

edge = **posedge** | **negedge**

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* always - Sensitivity list

Syntax:

@([edge] signal [**or** [edge] signal] …)
@([edge] signal [, [edge] signal] …)
@*

edge = **posedge** | **negedge**

Example:

```
always @(posedge Clk or negedge Reset)
always @(negedge Reset, Enable)
always @*                    // same as @(A or B or C or D)
begin
  Tmp1 = A & B;
  Tmp2 = C & D;
end
```


https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **initial**
  - **The initial procedural block statement is executed only once.**

Syntax:

```
initial
  statement
```

Example:

```
initial
 fork
   Q = 16'h0000;
   #20 Q = 16'hC10A;
   #40 Q = 16'hAFE0;
 join
```

Verilog Behavioural Modelling

- **begin-end**
  - **he begin-end groups statements together, so that they execute sequentially.**

Syntax:

```
begin [ : label
  [ local_declarations ] ]
  statement
end
```

Example:

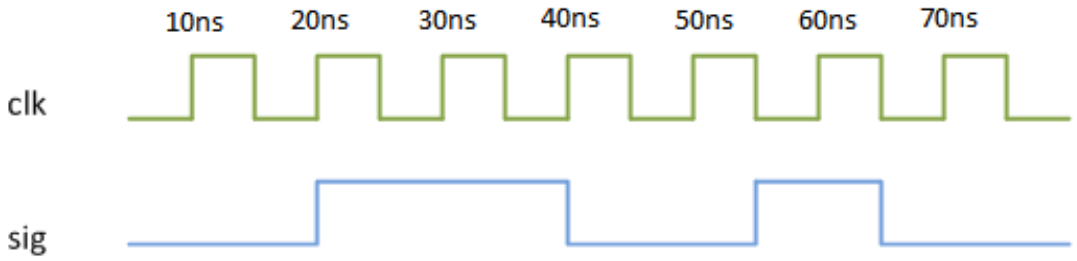
```
begin: l1
  integer I;
  for (I = 0; I < 4; I = I + 1)
    #Delay (A, B) = I;
end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **always, initial example**

Example: sig.v




```
module tb;
  reg clk;
  reg sig;

  // Clock generation
  // Process starts at time 0ns and loops after every 5ns
  always #5 clk = ~clk;

  // Initial block : Process starts at time 0ns
  initial begin
    // This system task will print out the signal values everytime they change
    $monitor("Time = %0t clk = %0d sig = %0d", $time, clk, sig);
```

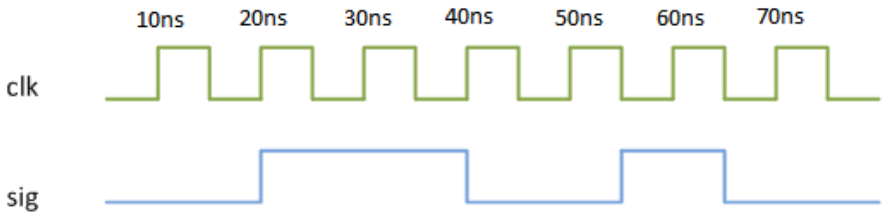
https://www.chipverify.com/verilog/verilog-simulation

Verilog Behavioural Modelling

* **always, initial example**

Example (cont.): sig.v

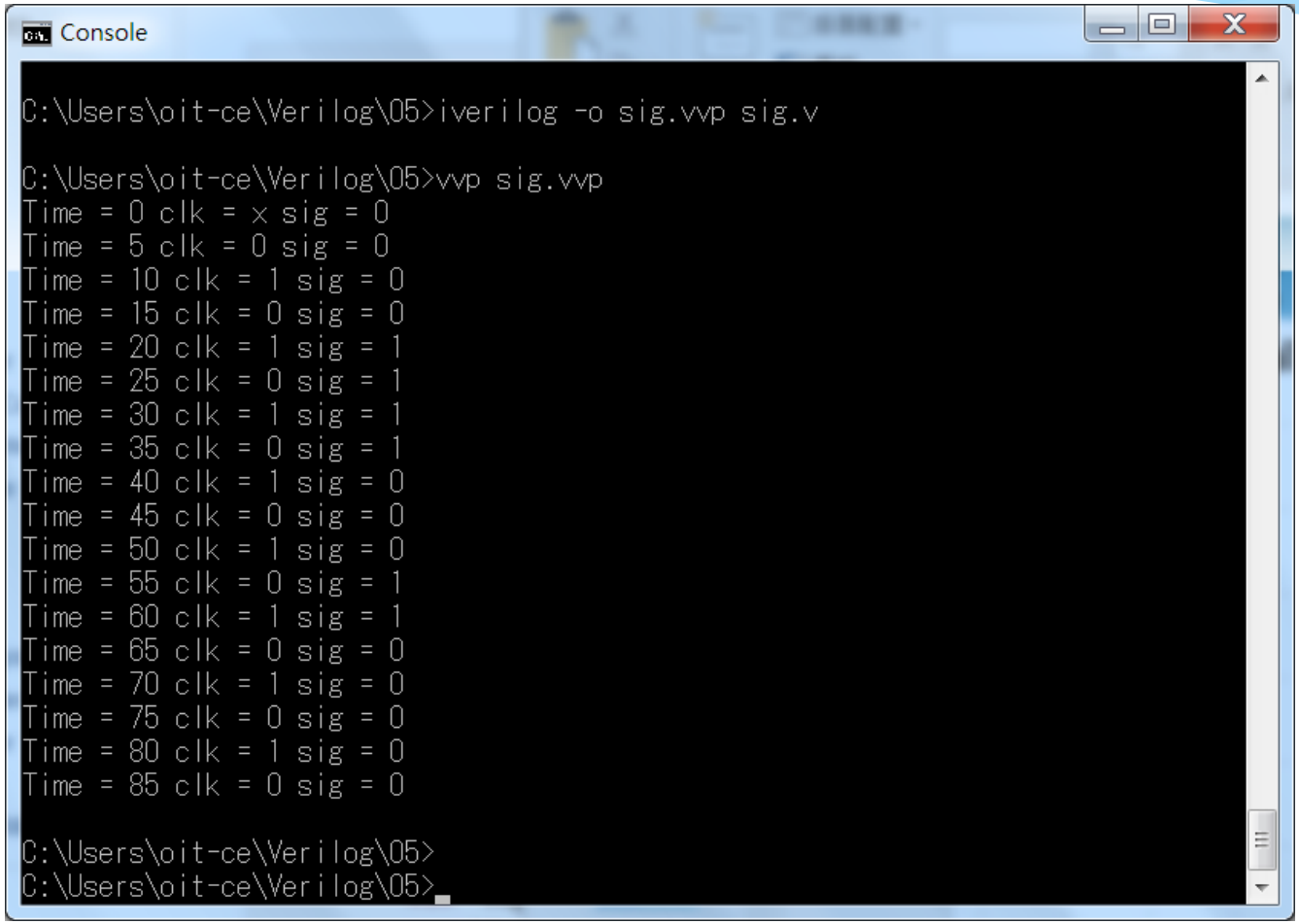
```
// Also called stimulus, we simply assign different values to the variables
// after some simulation "delay"
sig = 0;
#5   clk = 0;      // Assign clk to 0 at time 5ns
#15  sig = 1;      // Assign sig to 1 at time 20ns (#5 + #15)
#20  sig = 0;      // Assign sig to 0 at time 40ns (#5 + #15 + #20)
#15  sig = 1;      // Assign sig to 1 at time 55ns (#5 + #15 + #20 + #15)
#10  sig = 0;      // Assign sig to 0 at time 65ns (#5 + #15 + #20 + #15 + #10)
#20  $finish;      // Finish simulation at time 85ns
 end
endmodule
```



https://www.chipverify.com/verilog/verilog-simulation

Verilog Behavioural Modelling

* always, initial example



iverilog –o sig.vvp sig.v

vvp sig.vvp

https://www.chipverify.com/verilog/verilog-simulation

Verilog Behavioural Modelling

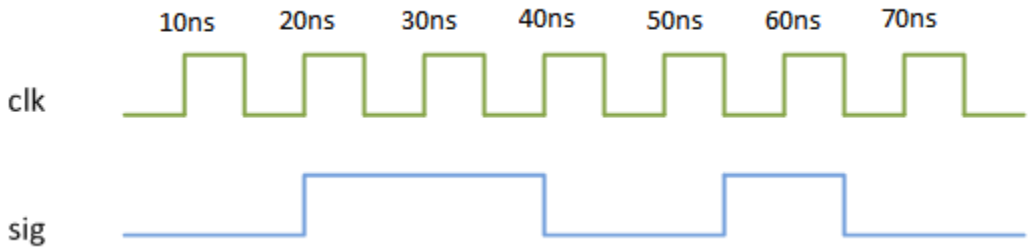
* always, initial example

```
Time = 0   clk = x   sig = 0
Time = 5   clk = 0   sig = 0
Time = 10  clk = 1   sig = 0
Time = 15  clk = 0   sig = 0
Time = 20  clk = 1   sig = 1
Time = 25  clk = 0   sig = 1
Time = 30  clk = 1   sig = 1
Time = 35  clk = 0   sig = 1
Time = 40  clk = 1   sig = 0
Time = 45  clk = 0   sig = 0
Time = 50  clk = 1   sig = 0
Time = 55  clk = 0   sig = 1
Time = 60  clk = 1   sig = 1
Time = 65  clk = 0   sig = 0
Time = 70  clk = 1   sig = 0
Time = 75  clk = 0   sig = 0
Time = 80  clk = 1   sig = 0
Time = 85  clk = 0   sig = 0
```

https://www.chipverify.com/verilog/verilog-simulation

```
iverilog –o sig.vvp sig.v
```

```
vvp sig.vvp
```



Verilog Behavioural Modelling

* **fork-Join**
  * **The fork-join groups statements together, so that they execute concurrently.**

Syntax:

```
fork [ : label
  [ local_declarations ] ]
  statements
join
```

Example:

```
fork: Stimuli
  #20 Data = 4'h0;
  #30 Data = 4'hF;   // This is executed last
  Reset = 0;         // This is executed first
  #10 Reset = 1;
join                 // It completes at time 30
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Procedural Assignment**
  * **A procedural assignment updates the value of register data types.**

Syntax:

[delay] register_name = [delay] expression; // blocking
[delay] register_name <= [delay] expression; // non-blocking

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Procedural Assignment

Example:

```
begin
  a = 0;
  #10 a = 1;
  #5 a = 2;
end          // time 0: a=0; time 10: a=1; time 15 (#10+#5): a=2;

begin
  a <= 0;
  #10 a <= 1;
  #5 a <= 2;
end          // time 0: a=0; time 5: a=2; time 10: a=1;

begin
  a <= b;
  b <= a;
end          // both assignments are evaluated before a or b changes
```

Verilog Behavioural Modelling

* **Procedural Continuous Assignment**
  * **A procedural continuous assignment assigns a value to a register.**

Syntax:

assign register_name = expression;
deassign register_name;

force net_or_register_name = expression;
release net_or_register_name;

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Procedural Continuous Assignment

Syntax:

assign register_name = expression;
deassign register_name;

force net_or_register_name = expression;
release net_or_register_name;

Example:

```
always @(posedge Clk)
  Cnt = Cnt + 1;

always @(Reset)
  if (Reset)
    // prevents counting until Reset is 0
    assign Cnt = 0; else
    // resume counting on next posedge Clk
    deassign Cnt;
initial
  begin
    #100 force Rst = 1'b0;
    #200 release Rst;
  end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

Procedural Timing Control

- **Procedural timing controls are used to schedule or delay procedural statements.**

Syntax:

#delay | event_control | **wait** (expression);

event_control = @name | @([edge] expression [or [edge] expression)
edge = **posedge** | **negedge**

Example:

#10 a = b;
@(**negedge** Reset or Enable) a = b;
wait (!Reset) #10 a = b;

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

- **Programming Statements**
  - **Verilog has the following programming statements:**
    - **Case**
      - **CaseX**
      - **CaseZ**
    - **If**
    - **Loop statements:**
      - **For**
      - **Forever**
      - **Repeat**
      - **While**
    - **Disable**

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – case**
  - **The case statement selects for execution one of several alternative statements; the alternative is chosen based on the value of the associated expression.**

Syntax:

```
case_word ( expression )
        case_match : statement
        [ default [ : ] statement ]
endcase

case_word = case | casex | casez
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

Syntax:

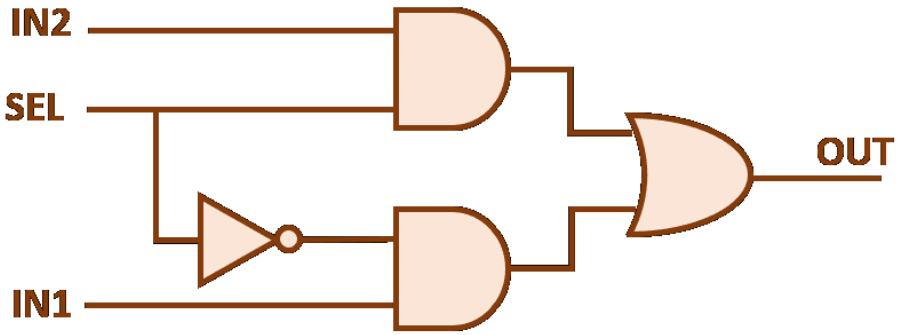
```
case_word ( expression )
        case_match : statement
        [ default [ : ] statement ]
endcase

case_word = case | casex | casez
```

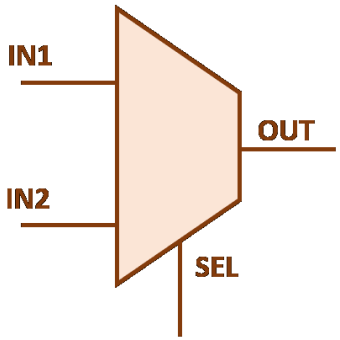
Example:

```
case (Addr)
  0 : Q <= 1;
  1 : begin
          Q <= 1;
          R <= 0;
      end
  2, 3 : R <= 1;
  default : $display("Illegal Addr value", Addr);
endcase

casez (Opcode)
      2'b1?? : Q <= 2'b01;
      2'b000 : Q <= 2'b00;
      2'b10? : Q <= 2'b10;
  default : Q <= 2'bxx;
endcase
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling





2X1 Multiplexor

Example:

```
module mux21(in1, in2, sl, out);
   input in1, in2, sl;
   output out;
   reg out;
   always@(in1 or in2 or sl)
   begin
      case (sl)
         1'b0: out = in1;
         1'b1: out = in2;
      endcase
   end
endmodule
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – casex**
  * **The casex is a special version of the case statement which uses X or Z logic values to represent don't care bits.**

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

- **Programming Statements – casez**
  - **The casez is a special version of the case statement which uses a Z logic value to represent don't care bits.**

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Programming Statements – casez

Example: casez.v

```
module casez_example();
reg [3:0] opcode;
reg [1:0] a,b,c;
reg [1:0] out;

always @ (opcode or a or b or c)
casez(opcode)
 4'b1zzx : begin // Don't care about lower 2:1 bit, bit 0 match with x
          out = a;
          $display("@%0dns 4'b1zzx is selected, opcode %b",$time,opcode);
         end
 4'b01?? : begin
          out = b; // bit 1:0 is don't care
          $display("@%0dns 4'b01?? is selected, opcode %b",$time,opcode);
         end
 4'b001? : begin  // bit 0 is don't care
          out = c;
          $display("@%0dns 4'b001? is selected, opcode %b",$time,opcode);
         end
 default : begin
          $display("@%0dns default is selected, opcode %b",$time,opcode);
         end
endcase
```

Treats z as don't care.

http://www.asic-world.com/verilog/vbehave2.html

Verilog Behavioural Modelling

* Programming Statements – casez

Example: (cont.) casez.v

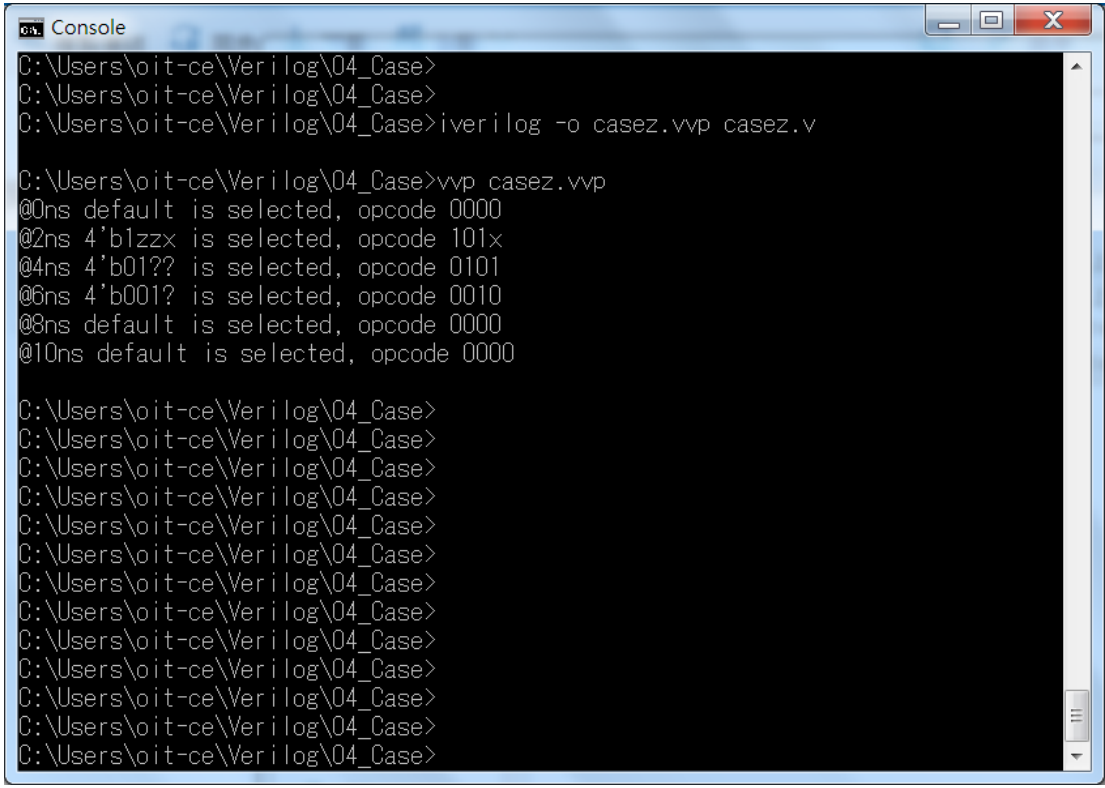
```
// Testbench code goes here
always #2 a = $random;
always #2 b = $random;
always #2 c = $random;

initial begin
  opcode = 0;
  #2 opcode = 4'b101x;
  #2 opcode = 4'b0101;
  #2 opcode = 4'b0010;
  #2 opcode = 4'b0000;
  #2 $finish;
end

endmodule
```

http://www.asic-world.com/verilog/vbehave2.html

Verilog Behavioural Modelling

* Programming Statements – casez

@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
@10ns default is selected, opcode 0000



http://www.asic-world.com/verilog/vbehave2.html

Verilog Behavioural Modelling

* **Programming Statements – if**
  - **if executes statements dependent on a condition.**
  - 進行訊號值的判斷，根據判斷結果執行相關處理
  - if 敘述能處理正準位與負準位觸發兩種訊號

Syntax:

```
if ( condition )
  statement
[ else if ( condition )
  statement ]
[ else
  statement ]
```

Example:

```
if (A)
  Q = 1;
else if (B == 6)
  Q = 2;
else
  Q = 0;
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – if**
  - 準位觸發範例

Example:

```
module test(reset, in, out);
   input reset, in;
   output out;
   reg out;
   always
   begin
      if (reset == 1'b1)    // 正
         out = 0;
      if (reset == 1'b0)    // 負
         out = in;
   end
endmodule
```


https://www.hdlworks.com/hdl_corner/verilog_ref/

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

* **Programming Statements – if範例**
  - **正反器(Flip-flop)，是一種具有兩種穩態的用於儲存的元件，可記錄二進位數位訊號「1」和「0」。**
  - **JK flip flop with Synchronous reset/set and clock enable 範例**

Inputs						Outputs
R	S	CE	J	K	Clk	Q
1	X	X	X	X	↑	0
0	1	X	X	X	↑	1
0	0	0	X	X	X	No Chg
0	0	1	0	0	X	No Chg
0	0	1	0	1	↑	0
0	0	1	1	1	↑	Toggle
0	0	1	1	0	↑	1

https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

Example: jk.v

```
//JK flip flop module
module FJKRSE(J,K,Clk,R,S,CE,Qout);

   input J,K;          //inputs
   input Clk;          //Clock
   input R;            //synchronous reset (R)
   input S;            //synchronous set (S)
   input CE;           //clock enable (CE)
   output Qout;        //data output (Q)

   //Internal variable
   reg Qout;

   always@ (posedge(Clk))  //Everything is synchronous to positive edge of clock
```

* JK flip flop with Synchronous reset/set and clock enable

https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

Example: (Cont.) jk.v

```
begin
    if(R == 1) //reset has highest priority.
      Qout = 0;
    else
      if(S == 1)  //set has next priority
        Qout = 1;
      else
        if (CE == 1) //J,K values are considered only when CE is ON.
          if (J == 0 && K == 0)
            Qout = Qout; //no change
          else if (J == 0 && K == 1)
            Qout = 0;  //reset
          else if (J == 1 && K == 0)
            Qout = 1;  //set
          else
            Qout = ~Qout;  //toggle
        else
          Qout = Qout; //no change
  end

endmodule
```

* JK flip flop with Synchronous reset/set and clock enable

https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

Example: (Cont.) Jk_tb.v

```
module tb_jkff;

  // Inputs
  reg J;
  reg K;
  reg Clk;
  reg R;
  reg S;
  reg CE;

  // Outputs
  wire Qout;

  // Instantiate the Unit Under Test (UUT)
  FJKRSE uut (
     .J(J),
     .K(K),
     .Clk(Clk),
     .R(R),
     .S(S),
     .CE(CE),
     .Qout(Qout)
  );
```

* JK flip flop with Synchronous reset/set and clock enable

https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

Example: (Cont.) Jk_tb.v

```
//Create 50 Mhz clock(20 ns clock period).
 initial Clk = 0;
 always #10 Clk = ~Clk;

 initial begin
    $monitor(R,S,CE,Clk,J,K, Qout);
    $dumpfile("jk.vcd");
    $dumpvars(0, uut);
    // Initialize Inputs
    J = 0;
    K = 0;
    R = 0;
    S = 0;
    CE = 0;
    #30;
    //Apply inputs
    R = 1;  #50;
    R = 0;
    S = 1;  #50;
    S = 0;
    J = 1;  K = 1;  #50;
```

* JK flip flop with Synchronous reset/set and clock enable

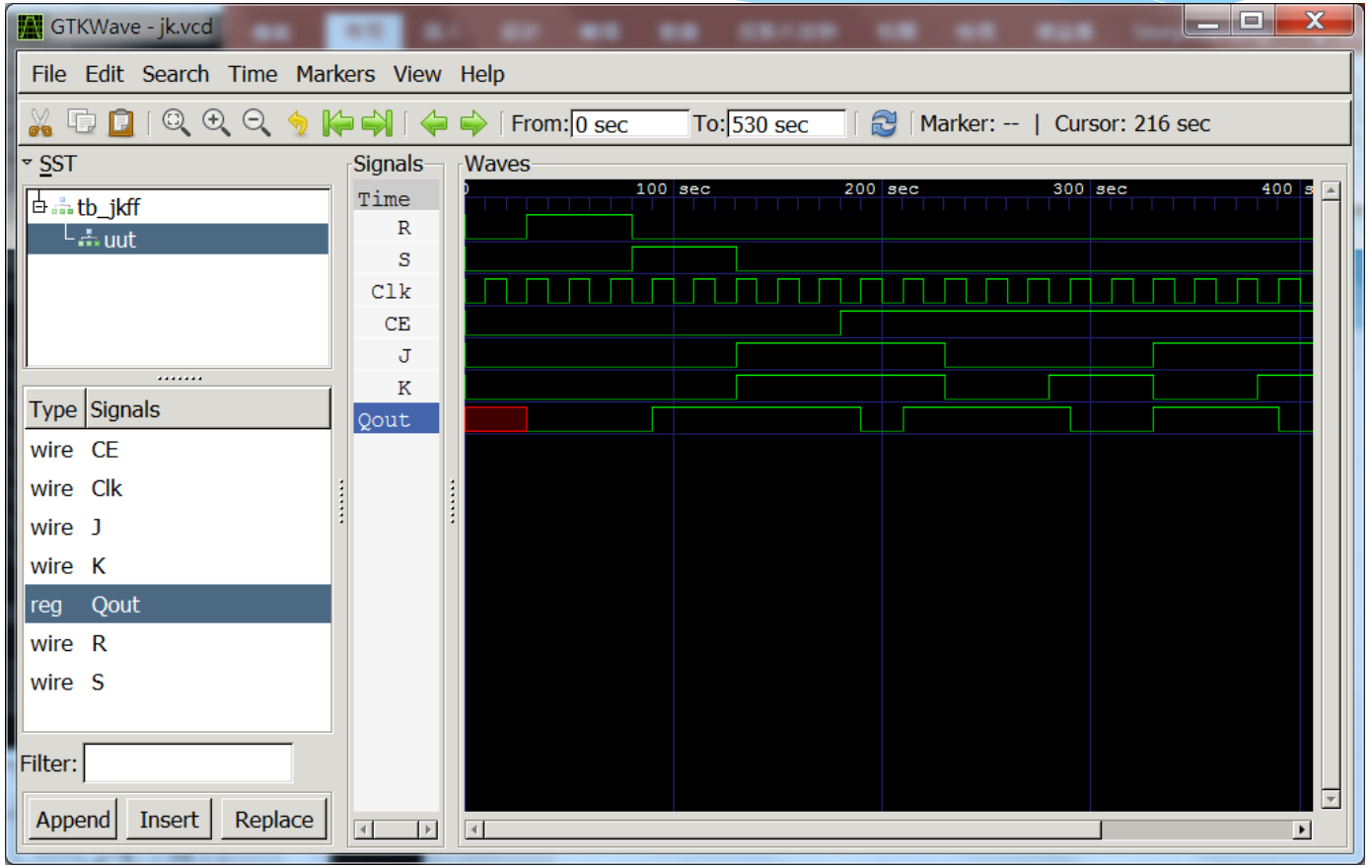
https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html
http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

Example: (Cont.) Jk_tb.v

```
    CE = 1; #50;
    J = 0;  K = 0;  #50;
    J = 0;  K = 1;  #50;
    J = 1;  K = 0;  #50;
    J = 1;  K = 1;  #50;
    CE = 0;
    #100 $finish;
  end

endmodule
```

* JK flip flop with Synchronous reset/set and clock enable



https://verilogcodes.blogspot.com/2015/11/verilog-code-for-jk-flip-flop-with.html

http://puremonkey2010.blogspot.com/2013/11/verilog-tutorial-always-ifelse-case-for.html

Verilog Behavioural Modelling

* **Programming Statements – for**
  * **For executes one or more statements iteratively until an expression is true.**

Syntax:

```
for ( initial_assignment; expression; step_assignment)
  statement
```

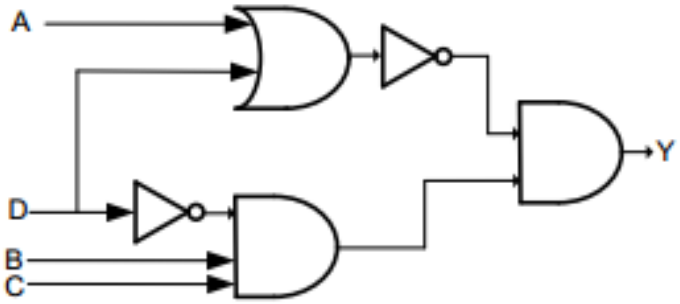
Example:

```
for (I = 0; I < 16; I = I + 2)
  Memory[I] = D;
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Programming Statements – for



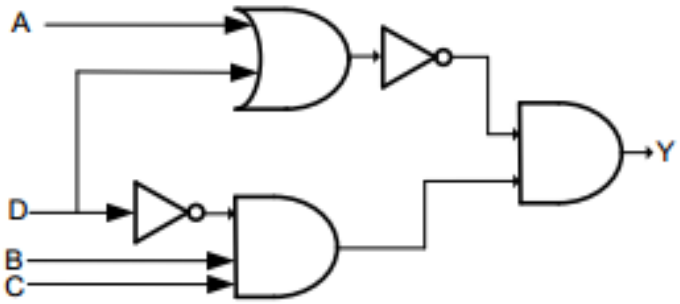
Example: for.v

```
module prob1(input wire a,b,c,d, output wire out);

   assign out = (a||d)&&(!d&&b&&c);

endmodule
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Programming Statements – for



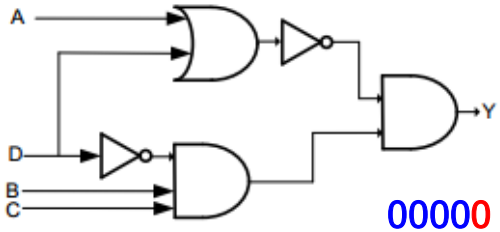
Example: for_tb.v

```
module prob1_tb();
   reg a,b,c,d;
   wire out;
   integer i;

   prob1 prob1_test(a,b,c,d, out);

   initial begin
      $monitor(a,b,c,d,out);
      for (i=0; i<16; i=i+1) begin
         {a,b,c,d} = i;
         #1;
      end
   end
endmodule
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* Programming Statements – for



00000
00010
00100
00110
01000
01010
01100
01110
10000
10010
10100
10110
11000
11010
11101
11110

```
Console

C:\Users\oit-ce\Verilog\03_For>iverilog -o for.vvp for_tb.v for.v

C:\Users\oit-ce\Verilog\03_For>vvp for.vvp
00000
00010
00100
00110
01000
01010
01100
01110
10000
10010
10100
10110
11000
11010
11101
11110

C:\Users\oit-ce\Verilog\03_For>
```

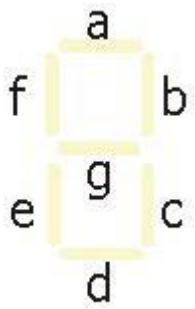
https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

Example: 7segment.v

```
//Verilog module.
module segment7(
    bcd,
    seg
   );

 //Declare inputs,outputs and internal variables.
    input [3:0] bcd;
    output [6:0] seg;
    reg [6:0] seg;

//always block for converting bcd digit into 7 segment format
   always @(bcd)
   begin
      case (bcd) //case statement
         0 : seg = 7'b0000001;
         1 : seg = 7'b1001111;
         2 : seg = 7'b0010010;
         3 : seg = 7'b0000110;
         4 : seg = 7'b1001100;
         5 : seg = 7'b0100100;
         6 : seg = 7'b0100000;
         7 : seg = 7'b0001111;
         8 : seg = 7'b0000000;
         9 : seg = 7'b0000100;
         //switch off 7 segment character when the bcd digit is not a decimal number.
         default : seg = 7'b1111111;
      endcase
   end
endmodule
```

* **Programming Statements – for**



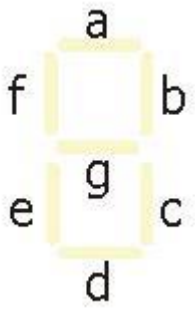
https://verilogcodes.blogspot.com/2015/10/verilog-code-for-bcd-to-7-segment.html

Verilog Behavioural Modelling

Example: 7segment_tb.v

```
module tb_segment7;

   reg [3:0] bcd;
   wire [6:0] seg;
   integer i;

   // Instantiate the Unit Under Test (UUT)
   segment7 uut (
      .bcd(bcd),
      .seg(seg)
   );

//Apply inputs
   initial begin
      $monitor(bcd,seg);
      $dumpfile("7segment.vcd");
      $dumpvars(0, uut);

      for(i = 0;i < 16;i = i+1) //run loop for 0 to 15.
      begin
         bcd = i;
         #10; //wait for 10 ns
      end
   end

endmodule
```

* **Programming Statements – for**



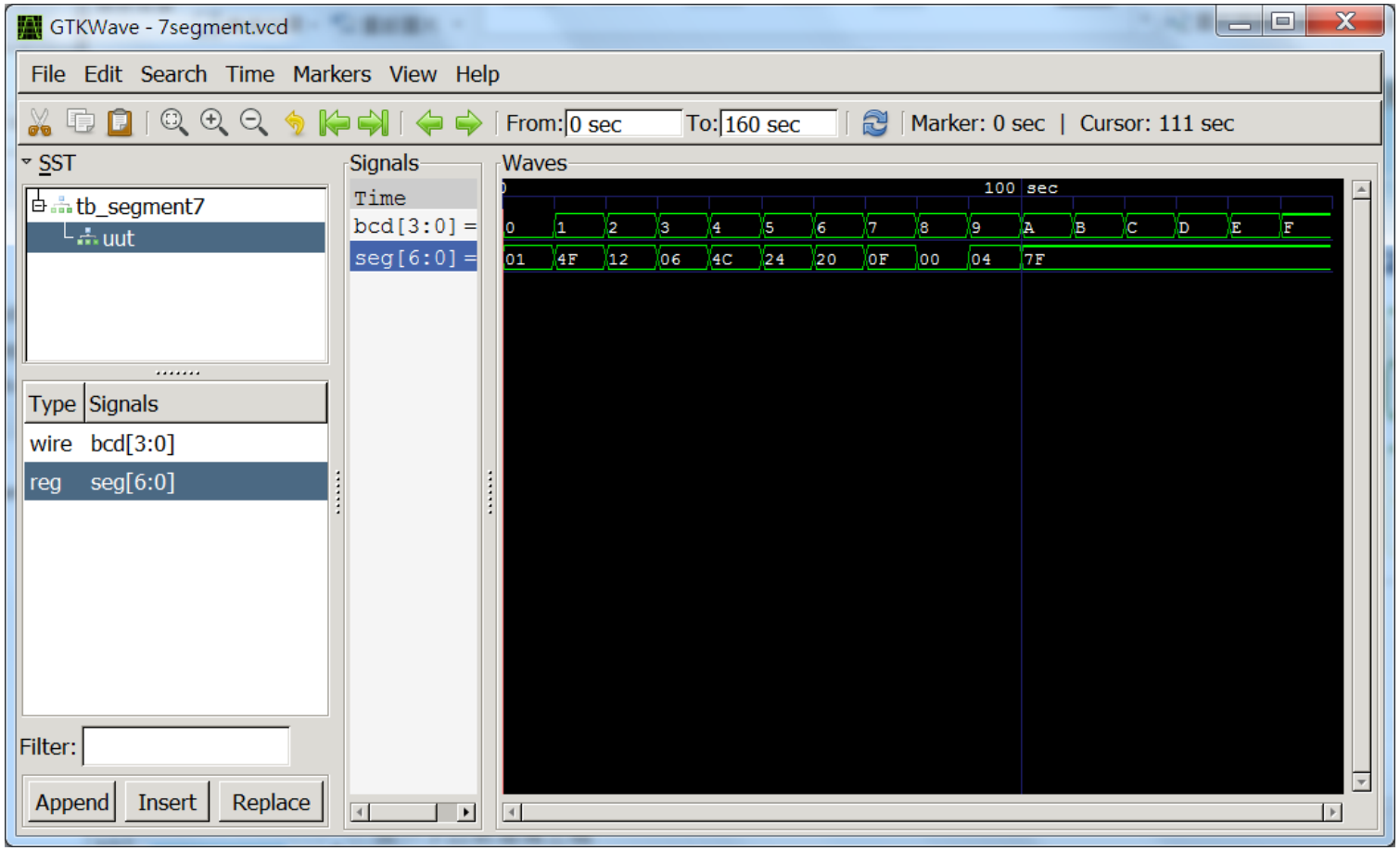
https://verilogcodes.blogspot.com/2015/10/verilog-code-for-bcd-to-7-segment.html

Verilog Behavioural Modelling

* Programming Statements – for

C:\> vvp 7segment.vvp
VCD info: dumpfile 7segment.vcd opened for output.

```
 0  1
 1 79
 2 18
 3  6
 4 76
 5 36
 6 32
 7 15
 8  0
 9  4
10 127
11 127
12 127
13 127
14 127
15 127
```



Verilog Behavioural Modelling

* **Programming Statements – forever**
  * **Forever executes one or more statements in an indefinite loop.**

Syntax:

```
forever
  statement
```

Example:

```
initial
begin : Clock
  Clk = 0;
  forever
      #50 Clk = !Clk;
end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – repeat**
  * **Repeat executes statements a specified number of times.**

Syntax:

repeat (expression)
  statement

Example:

repeat (LoopNumber)
  A = A + ~B;

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – while**
  * **While executes statements as long as an expression evaluates as true.**

Syntax:

```
while ( expression )
  statement
```

Example:

```
while (Nmbr)
 begin
   Cnt = Cnt + 1;
   Nmbr = Numbr >> 1;
 end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

* **Programming Statements – disable**
  - **Disable terminates the execution of a named group of statements before all the statements have been executed.**

Syntax:

```
disable group_name
```

Example:

```
begin: BreakBlock
 I = 0;
 forever begin
  if (I == A)
    disable BreakBlock;
  #1 I = I + 1;
 end
end
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog Behavioural Modelling

Continuous Assignment

- **A continuous assignment drives a value into a net.**

Syntax:

```
net_data_type [ strength ] [ delay ] [ size ] net_name = expression;    // implicit
assign [ strength ] [ #( delay ) ] net_name = expression;               // explicit
```

Example:

```
wire Out;
assign Out = A & B;
assign {COut, Sum} = A + B + CIn;
wire #50 Out = A & B;
```

https://www.hdlworks.com/hdl_corner/verilog_ref/

Verilog 範例

* http://www.asic-world.com/examples/verilog/index.html

- Decoder And Encoders
- Mux
- Flip Flop And Latches
- Counters
- Memories
- Parity And CRC
- Modeling With Switch Primitives
- Modeling With Gate Primitives
- User Defined Primitives
- PLI Examples
- Verilog UART Model
- Verilog Arbiter Model