Avalon 傳輸

定義客制化指令

銘傳電通　陳慶逸

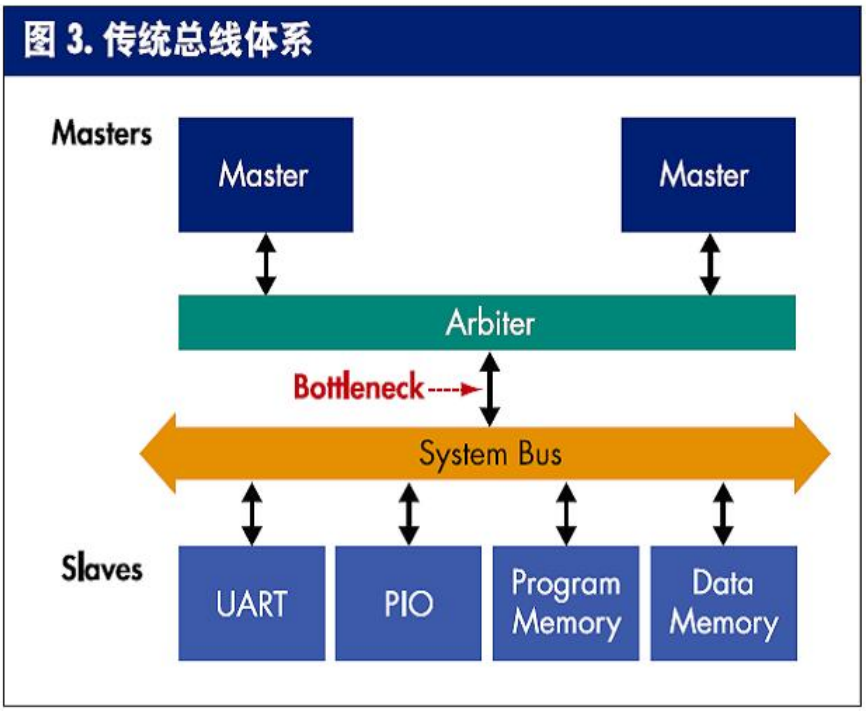
Avalon 傳輸

一、背景知識

Avalon 匯流排介面可分為兩類：Slave 和 Master。Slave 和 Master 的主要區別是對於 Avalon 匯流排控制權的把握。Master 介面具有相接的 Avalon 匯流排的控制權，而 Slave 介面是被動的。

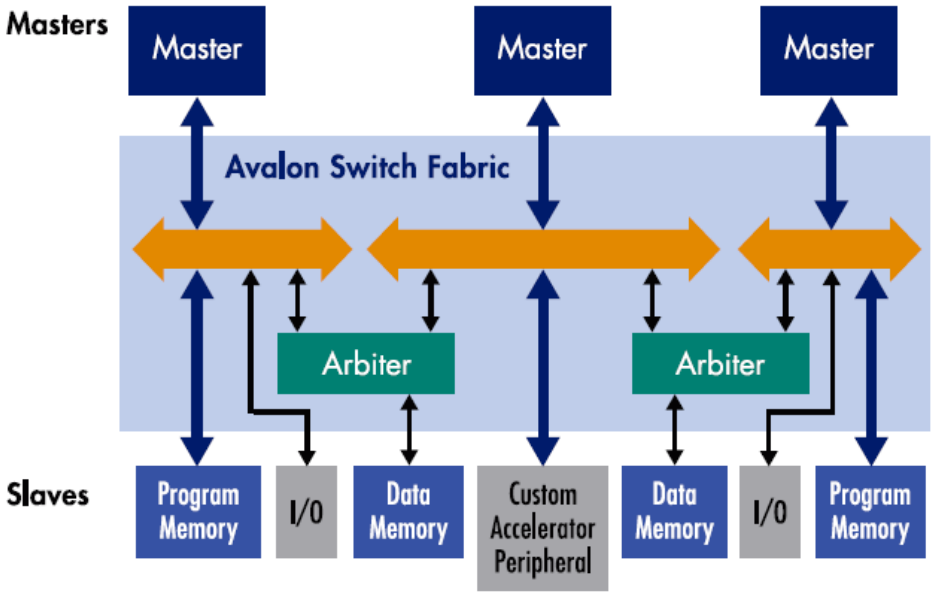
Introduction Avalon Bus

匯流排仲裁器根據固定規則分發匯流排資源給某個主機(如 CPU 和 DMA)。每次只有一個主機能夠接入匯流排、使用匯流排資源，因此會導致帶寬瓶頸。



每個匯流排主機均有自己的專用互聯，匯流排主機只需搶占共享從機，而不是匯流排本身。Avalon 交換架構的同時多主機體系架構提升了系統帶寬，消除了帶寬瓶頸。

图 4. Avalon 交换架构体系



Fundamental Slave Read & Write Transfer

在 Avalon 匯流排上的 Slave 設備進行讀操作時，就會啟動 Avalon 匯流排的從讀傳輸過程，其相關信號：

- clk: 時脈
- address: 位址信號。用於選中相對應的 Avalon Slave 設備的對應暫存器或記憶體單元。
- byteenable_n: 位元致能信號。對於 32 位元的週邊設備應該有 4 位元的 byteenable_n 信號，對於每一個 8 位元，對應一個 byteenable_n，使用者可以直接理解為位址。
- read_n: 讀信號。
- chipselect: 晶片選擇信號。這個信號在 address, byteenable_n 有效後，在匯流排上出現。
- readdata: 讀資料信號，可以是 1~32 位元的信號。

Figure 4. Fundamental Slave Read Transfer

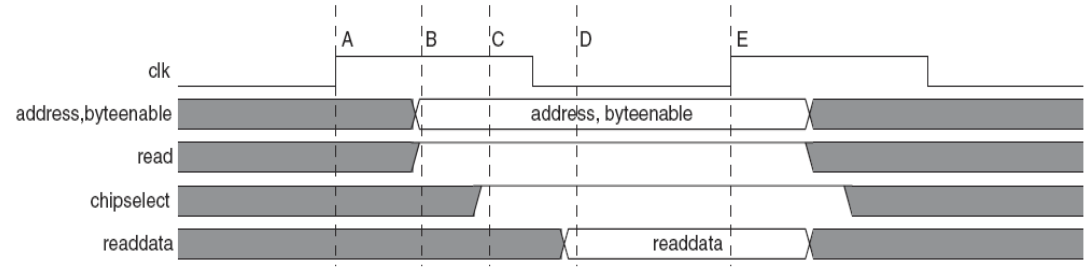
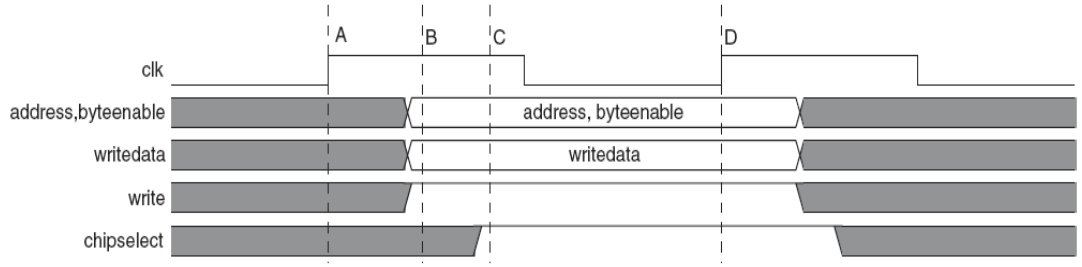


Figure 9. Fundamental Slave Write Transfer



Slave Read and Write Transfer with One Fixed Wait-State

Avalon 帶一個延遲狀態從讀傳輸過程與 Avalon 從讀傳輸的差別，只是整個傳輸過程延遲一個週期完成。該操作需要兩個 clk 週期。

Figure 5. Slave Read Transfer with One Fixed Wait-State

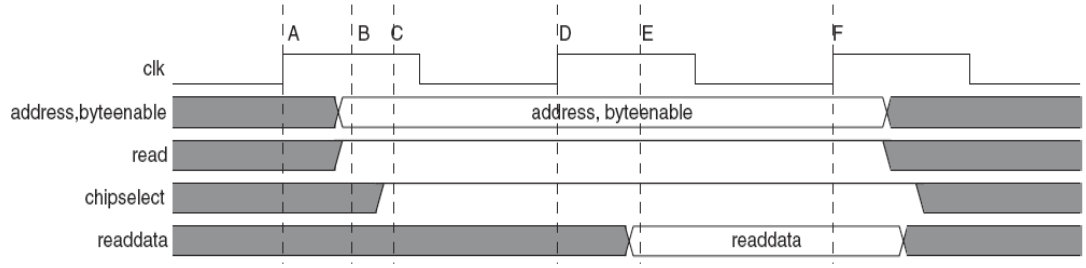
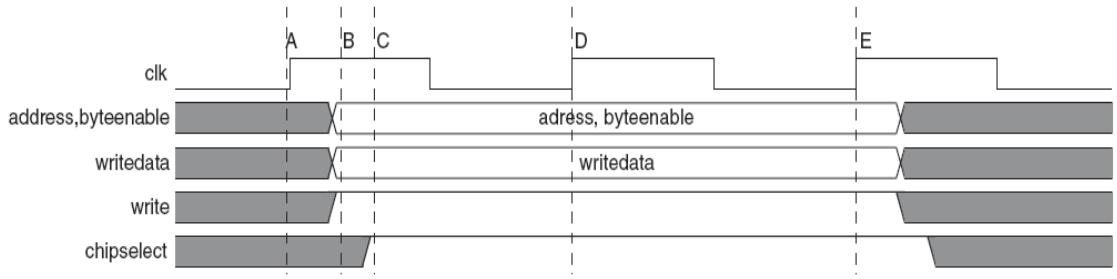
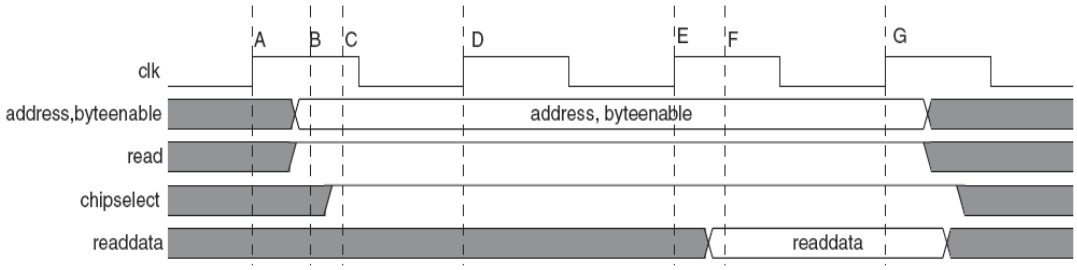


Figure 10. Slave Write Transfer with One Fixed Wait-State



Slave Read Transfer with Multiple Fixed Wait-States

Figure 6. Slave Read Transfer with Multiple Fixed Wait-States



Slave Read & Writer Transfer with Variable Wait-States

Figure 7. Slave Read Transfer with Variable Wait-States

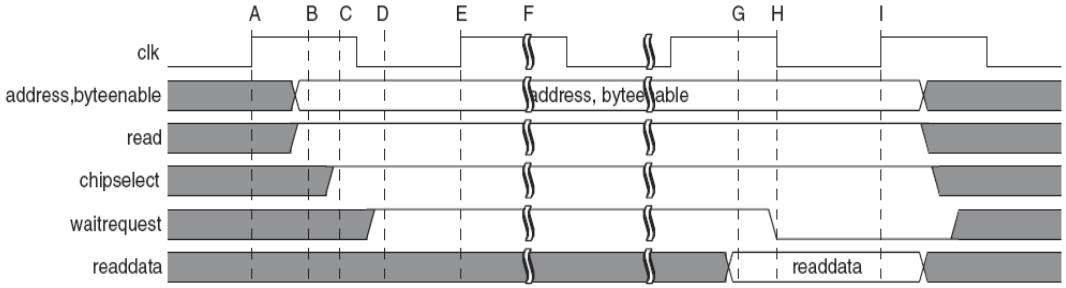
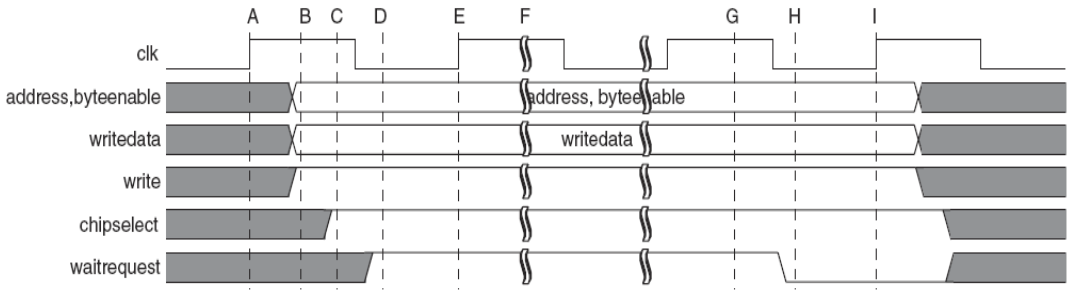


Figure 11. Slave Write Transfer with Variable Wait-States



二、硬體設計

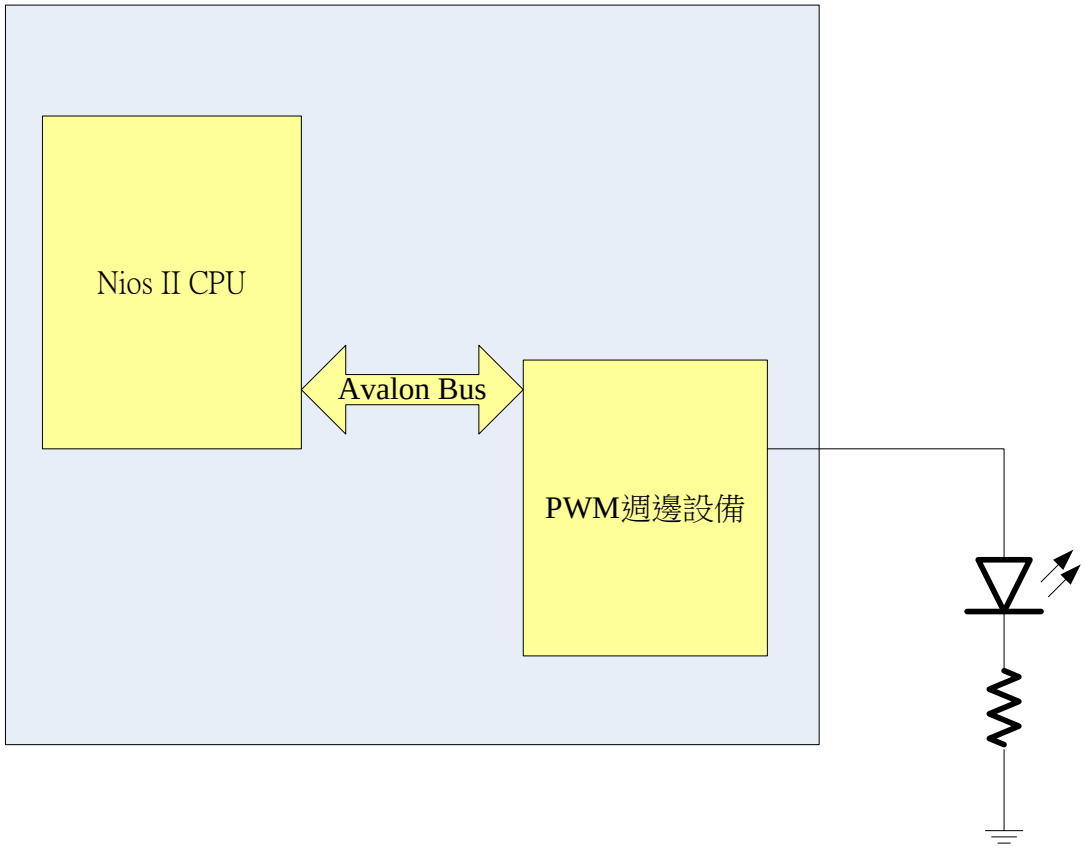
1. 在工作的 project 下建立一個資料夾 ip，並將 avs_IF_pwm.v 複製到 ip 資料夾內。

```
`default_nettype none

module avs_IF_pwm    (
                                // clock & reset IF
                                csi_clockreset_clk,
                                csi_clockreset_reset_n,

                                // Slave IF of Avalon bus
                                avs_s1_address,
                                avs_s1_chipselect,
                                avs_s1_write_n,
                                avs_s1_writedata,

                                  // Conduit IF of Avalon bus
//                                avs_s1_export_data_out,
                                avs_s1_export_pwm_out
                            );


// declaring input and output interface

input               csi_clockreset_clk;          // global clock signal
input               csi_clockreset_reset_n;    // global reset_n signal, active at low
input               avs_s1_address;                 // address bus signal
input               avs_s1_chipselect;            // control signal of chipselect
input               avs_s1_write_n;                 // control signal of write active
input    [31:0] avs_s1_writedata;              // data bus of write
//output [31:0] avs_s1_export_data_out;    // export register data
output [7:0]    avs_s1_export_pwm_out;      // export pwm signal to LED

////////////////////////////////////////////////
// declaring internal registers or wires
```

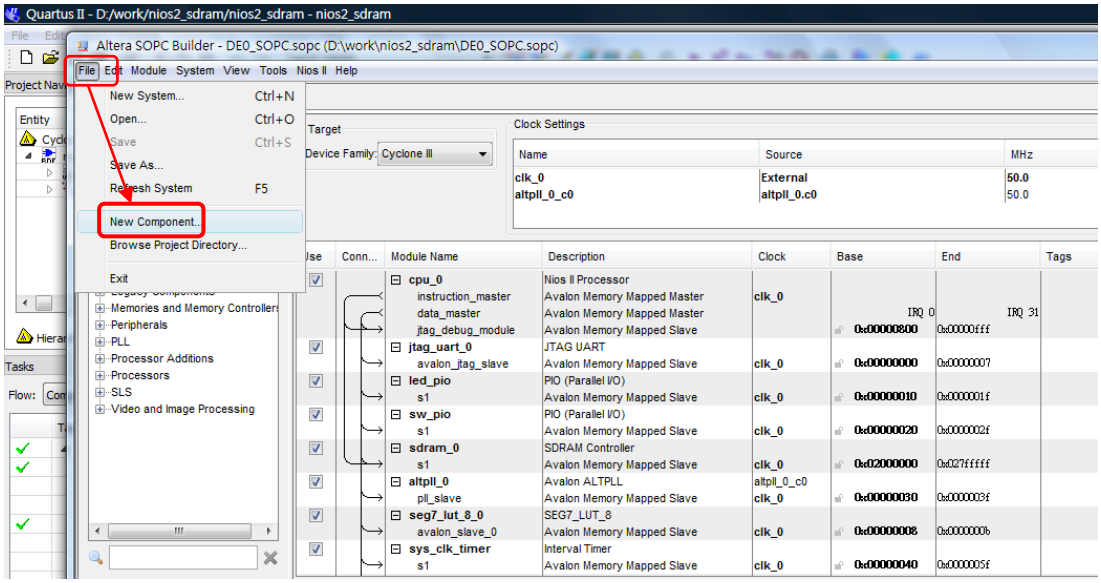
```
////////////////////////////////////////////////
reg [31:0]       rDiv, rDuty, rCounter;
reg                 pwm_on;

////////////////////////////////////////////////
// describing the circuit of dehavior
////////////////////////////////////////////////

//slave write: process
always@(posedge csi_clockreset_clk or negedge csi_clockreset_reset_n) begin
      if (!csi_clockreset_reset_n) begin //reset
            rDiv <= 0;
            rDuty <= 0;
      end
      else begin
         if (avs_s1_chipselect && (avs_s1_write_n ==0)) begin
               if (avs_s1_address == 0) begin
                     rDiv    <= avs_s1_writedata;
                     rDuty <= rDuty;
               end
               else begin
                     rDiv    <= rDiv;
                     rDuty <= avs_s1_writedata;
               end
         end
   end
end

//clock divider: process
always@(posedge csi_clockreset_clk or negedge csi_clockreset_reset_n) begin
      if (!csi_clockreset_reset_n)
            rCounter <= 0;
      else begin
            if (rCounter >= rDiv)
                  rCounter <= 0;
            else
                  rCounter <= rCounter + 1;
      end
```

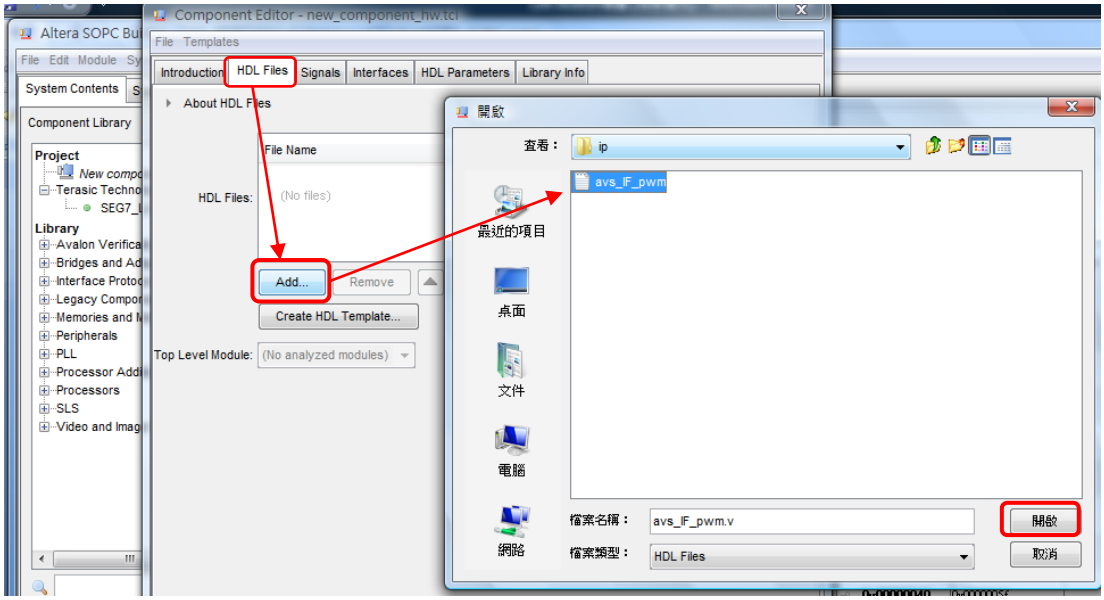
```
end

//Generate pwm signal:
always@(posedge csi_clockreset_clk or negedge csi_clockreset_reset_n) begin
	if (!csi_clockreset_reset_n)
		pwm_on <= 1;
	else begin
		if (rCounter >= rDuty)
			pwm_on <= 0;
		else if (rCounter == 0)
			pwm_on <= 1;
		else
			pwm_on <= pwm_on;
	end
end

// export data
assign avs_s1_export_pwm_out = {pwm_on, pwm_on, pwm_on, pwm_on, pwm_on, pwm_on,
pwm_on, pwm_on};
//assign avs_s1_export_data_out = (avs_s1_address == 1'b0) ? rDiv : rDuty;

endmodule
```

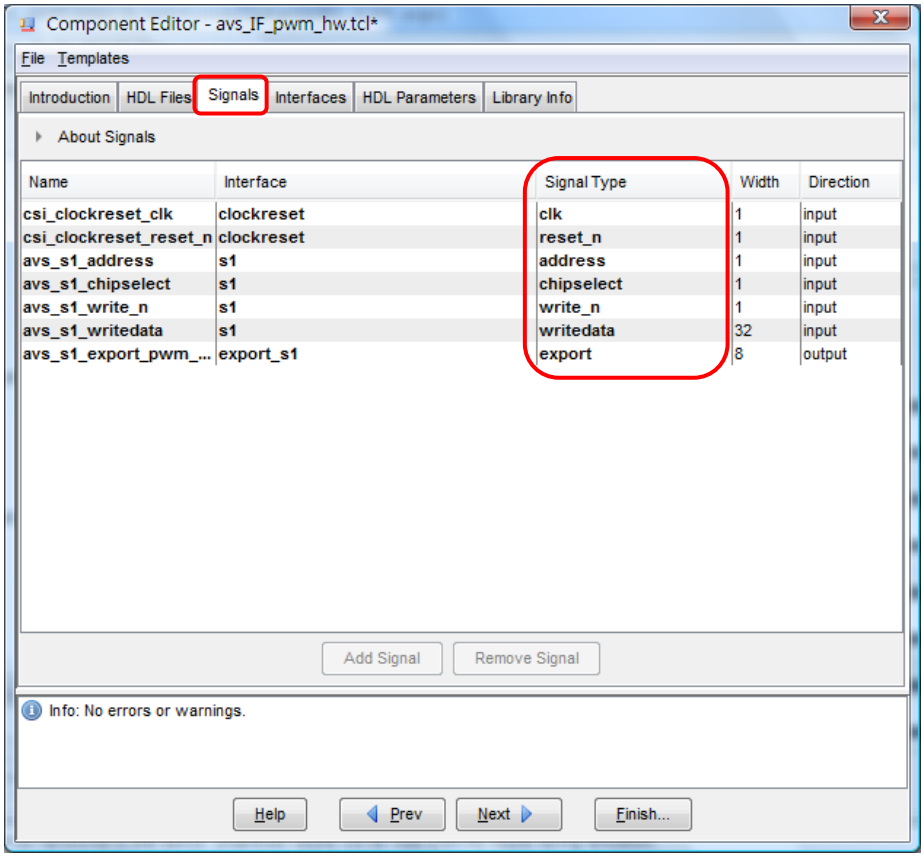
本範例為一個脈衝寬度調變電路，PWM 的輸出將連接到 FPGA 外的 LED 或直流馬達上，透過控制 PWM 週邊設備暫存器可以對 LED(或直流馬達)進行控制。

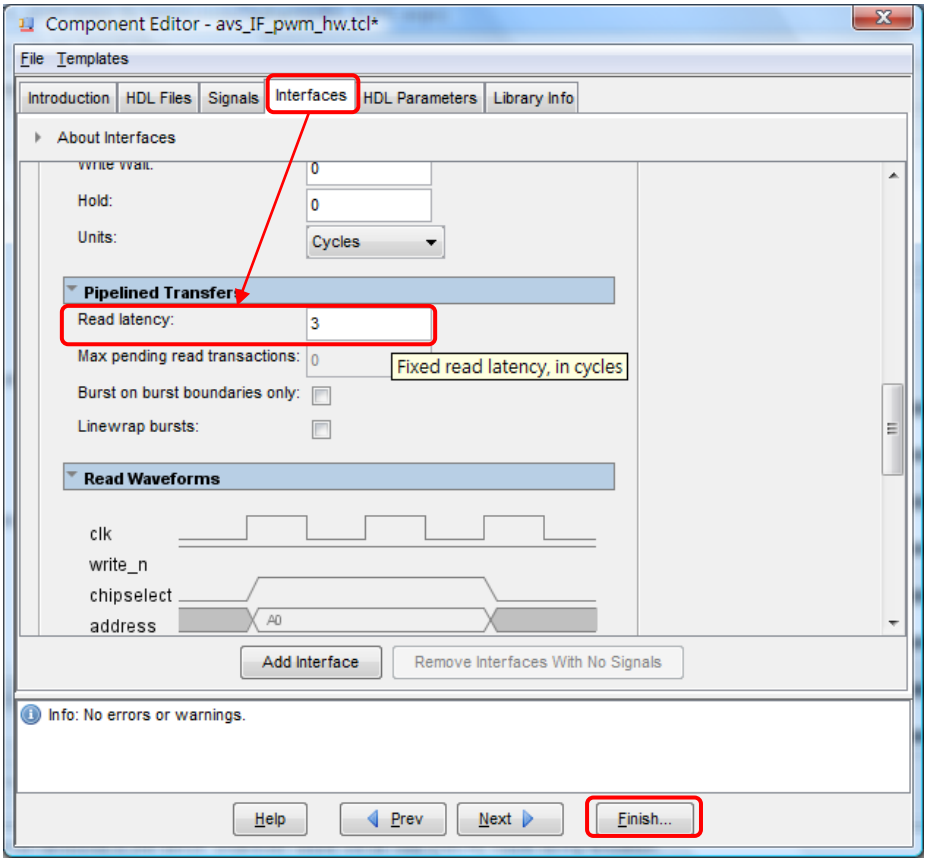


2. File/ New Component。

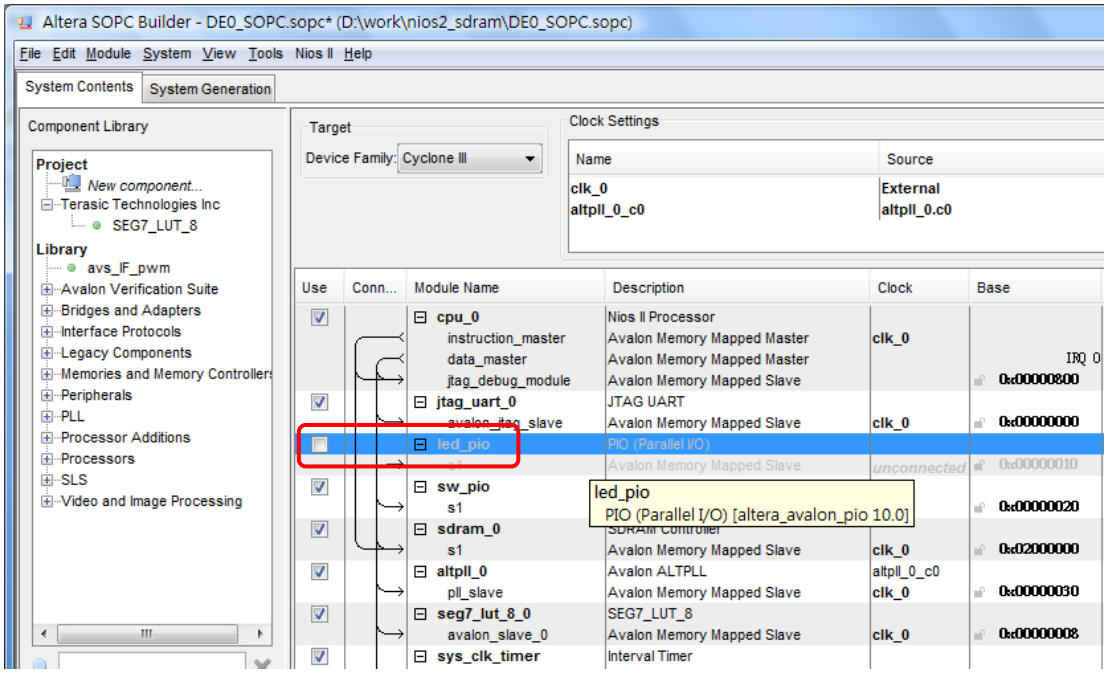



Component Editor - new_component_hw.tcl
File Templates
Introduction
HDL Files
Signals
Interfaces
HDL Parameters
Library Info
About HDL Files
File Name
HDL Files:
(No files)
Add...
Remove
Create HDL Template...
Top Level Module:
(No analyzed modules)
開啟
查看：
ip
avs_IF_pwm
最近的項目
桌面
文件
電腦
網路
檔案名稱：
avs_IF_pwm.v
檔案類型：
HDL Files
開啟
取消

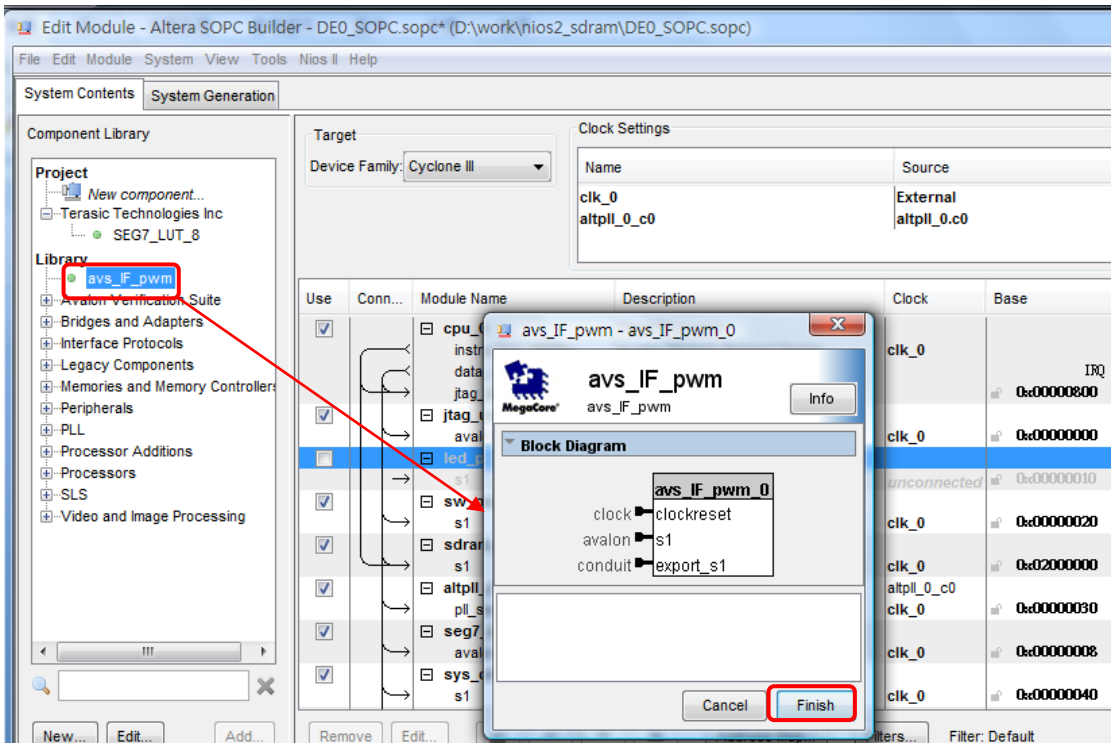

Component Editor - avs_IF_pwm_hw.tcl*
File Templates
Introduction
HDL Files
Signals
Interfaces
HDL Parameters
Library Info
About Signals
Name
Interface
Signal Type
Width
Direction
csi_clockreset_clk
clockreset
clk
1
input
csi_clockreset_reset_n
clockreset
reset_n
1
input
avs_s1_address
s1
address
1
input
avs_s1_chipselect
s1
chipselect
1
input
avs_s1_write_n
s1
write_n
1
input
avs_s1_writedata
s1
writedata
32
input
avs_s1_export_pwm_...
export_s1
export
8
output
Add Signal
Remove Signal
Info: No errors or warnings.
Help
Prev
Next
Finish...

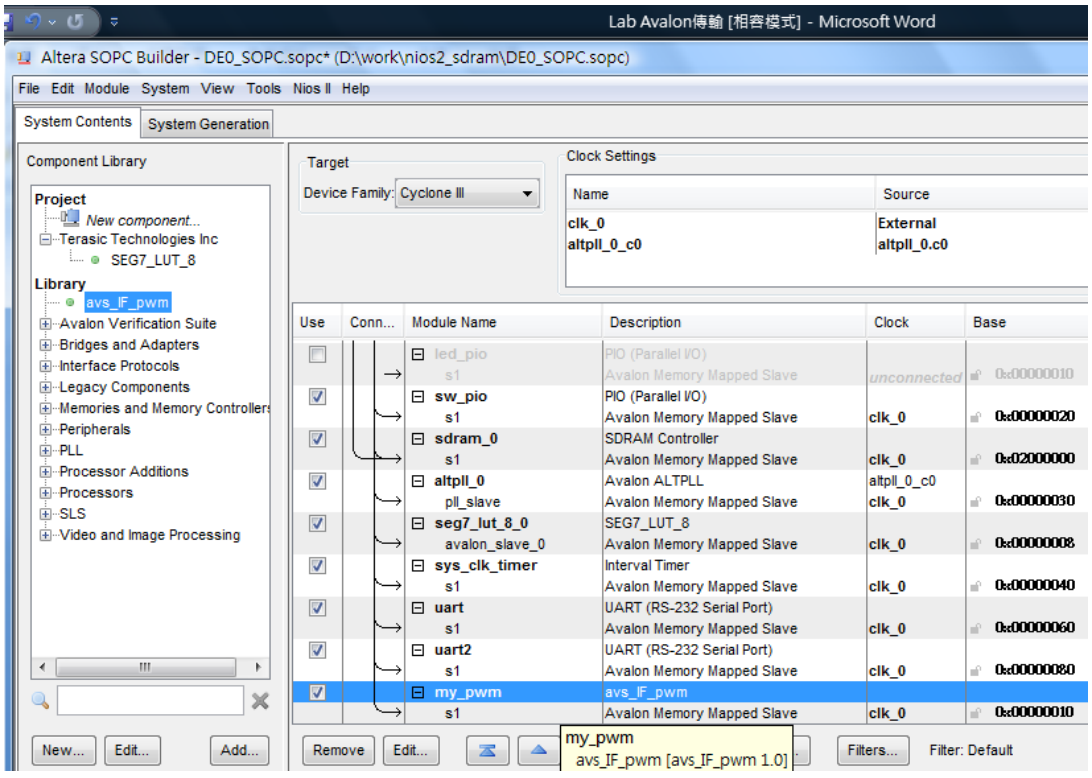


3. Uncheck the **Use** box next to **led_pio**.。

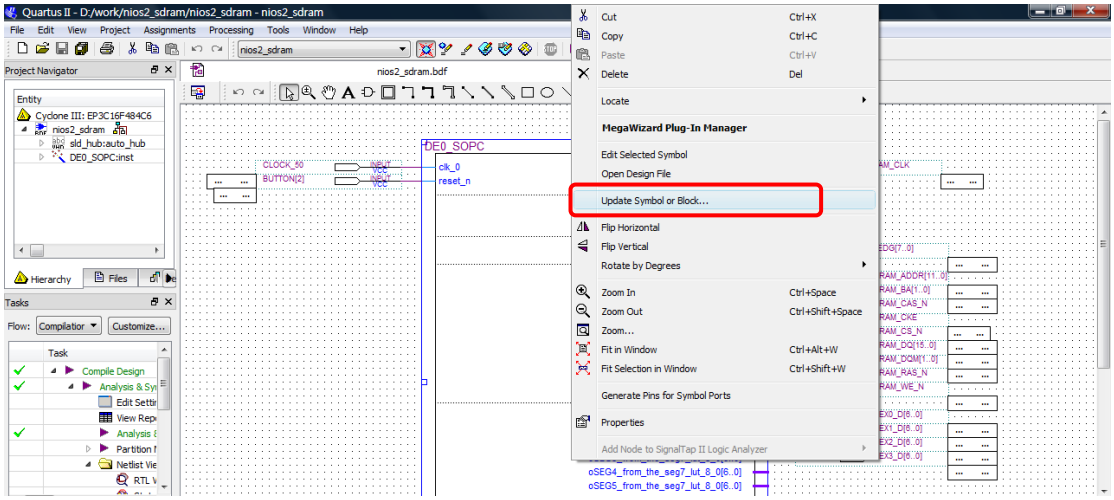


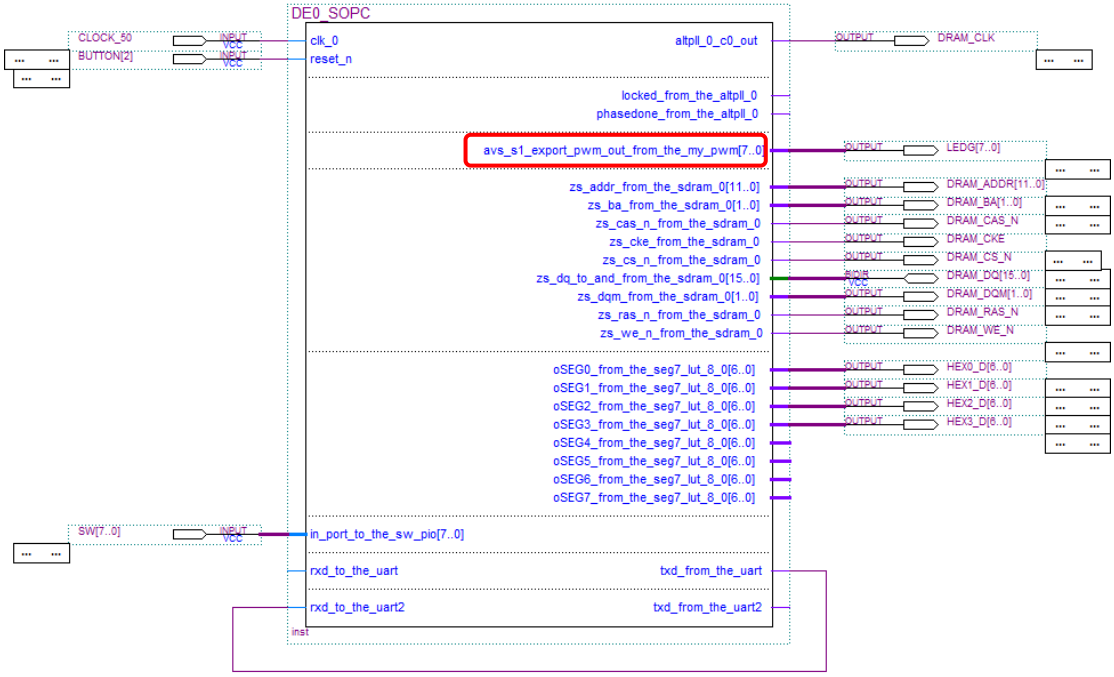
4. Add “avs_IF_pwm” component，Rename the peripheral to my_pwm.





5. 按滑鼠右鍵選擇 Update Symbol or Block 更新 Nios II Symbol。重新整理連線後點選 Start Compilation 編譯電路。

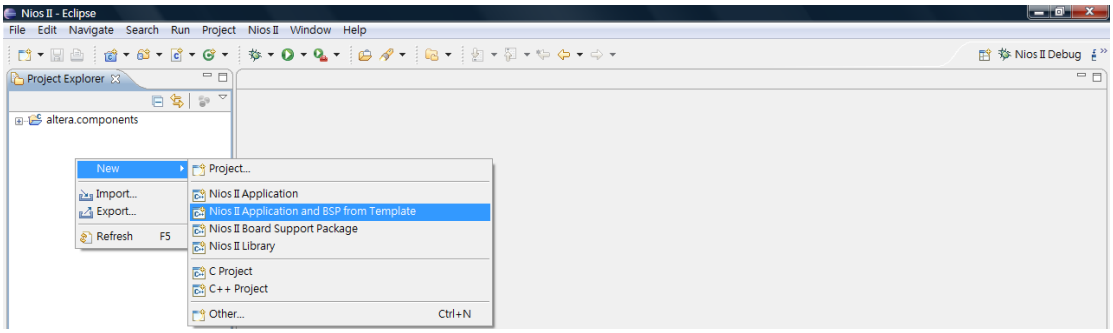

Quartus II - D:/work/nios2_sdram/nios2_sdram - nios2_sdram
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Cyclone III: EP3C16F484C6
nios2_sdram
sld_hub:auto_hub
DE0_SOPC:inst
Hierarchy
Files
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Syn
Edit Setti
View Rep
Analysis &
Partition
Netlist Vie
RTL V
nios2_sdram.bdf
DE0_SOPC
CLOCK_50
BUTTON[2]
clk_0
reset_n
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Locate
MegaWizard Plug-In Manager
Edit Selected Symbol
Open Design File
Update Symbol or Block...
Flip Horizontal
Flip Vertical
Rotate by Degrees
Zoom In Ctrl+Space
Zoom Out Ctrl+Shift+Space
Zoom...
Fit in Window Ctrl+Alt+W
Fit Selection in Window Ctrl+Shift+W
Generate Pins for Symbol Ports
Properties
Add Node to SignalTap II Logic Analyzer
oSEG4_from_the_seg7_lut_8_0[6..0]
oSEG5_from_the_seg7_lut_8_0[6..0]

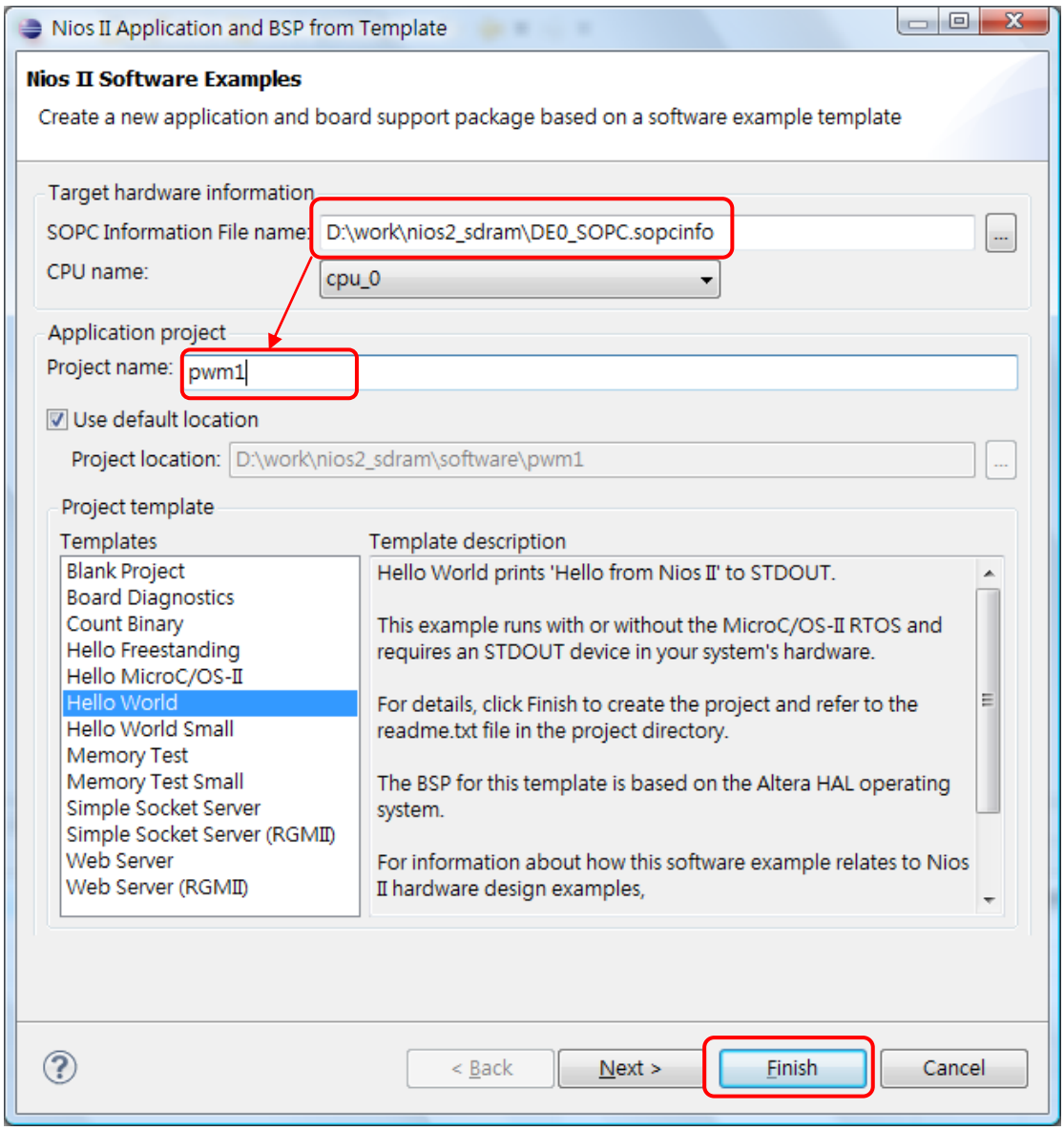

DE0_SOPC
CLOCK_50
BUTTON[2]
clk_0
reset_n
altpll_0_c0_out
DRAM_CLK
locked_from_the_altpll_0
phasedone_from_the_altpll_0
avs_s1_export_pwm_out_from_the_my_pwm[7..0]
LEDG[7..0]
zs_addr_from_the_sdram_0[11..0]
DRAM_ADDR[11..0]
zs_ba_from_the_sdram_0[1..0]
DRAM_BA[1..0]
zs_cas_n_from_the_sdram_0
DRAM_CAS_N
zs_cke_from_the_sdram_0
DRAM_CKE
zs_cs_n_from_the_sdram_0
DRAM_CS_N
zs_dq_to_and_from_the_sdram_0[15..0]
DRAM_DQ[15..0]
zs_dqm_from_the_sdram_0[1..0]
DRAM_DQM[1..0]
zs_ras_n_from_the_sdram_0
DRAM_RAS_N
zs_we_n_from_the_sdram_0
DRAM_WE_N
oSEG0_from_the_seg7_lut_8_0[6..0]
HEX0_D[6..0]
oSEG1_from_the_seg7_lut_8_0[6..0]
HEX1_D[6..0]
oSEG2_from_the_seg7_lut_8_0[6..0]
HEX2_D[6..0]
oSEG3_from_the_seg7_lut_8_0[6..0]
HEX3_D[6..0]
oSEG4_from_the_seg7_lut_8_0[6..0]
oSEG5_from_the_seg7_lut_8_0[6..0]
oSEG6_from_the_seg7_lut_8_0[6..0]
oSEG7_from_the_seg7_lut_8_0[6..0]
SW[7..0]
in_port_to_the_sw_pio[7..0]
rxd_to_the_uart
txd_from_the_uart
rxd_to_the_uart2
txd_from_the_uart2
inst

三、Nios II EDS 軟體設計

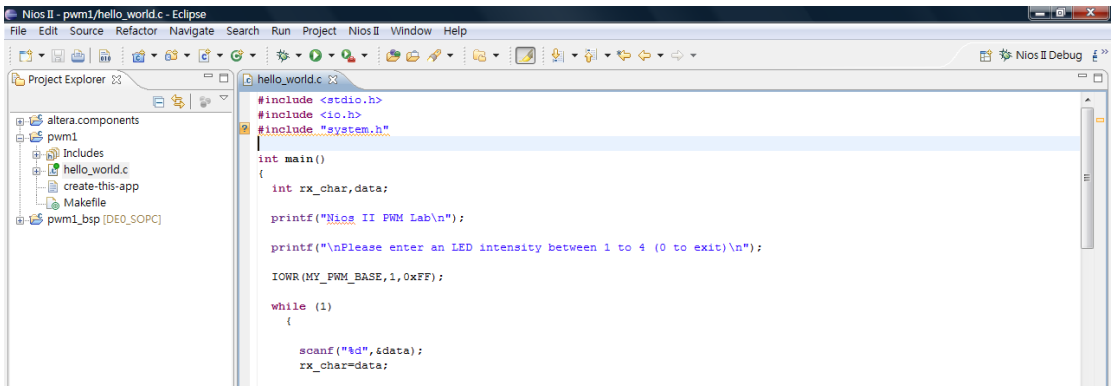
1. 在 Nios II DES 中建立新專案。

1-1 File/ New/ Nios II Application and BSP from Template

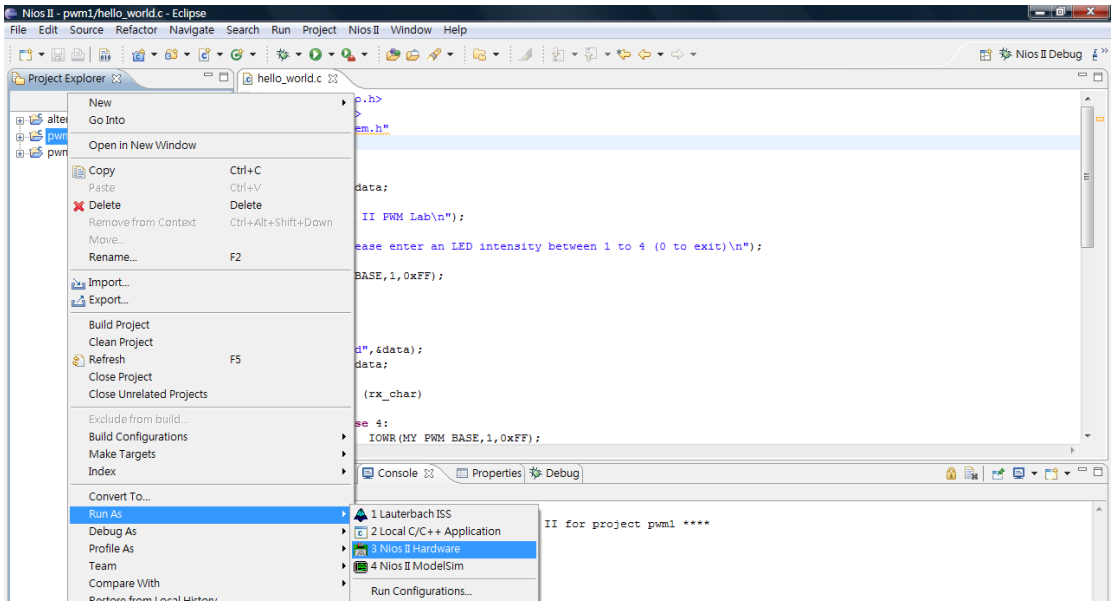



Nios II Application and BSP from Template
Nios II Software Examples
Create a new application and board support package based on a software example template
Target hardware information
SOPC Information File name:
D:\work\nios2_sdram\DE0_SOPC.sopcinfo
CPU name:
cpu_0
Application project
Project name:
pwm1
Use default location
Project location:
D:\work\nios2_sdram\software\pwm1
Project template
Templates
Blank Project
Board Diagnostics
Count Binary
Hello Freestanding
Hello MicroC/OS-II
Hello World
Hello World Small
Memory Test
Memory Test Small
Simple Socket Server
Simple Socket Server (RGMII)
Web Server
Web Server (RGMII)
Template description
Hello World prints 'Hello from Nios II' to STDOUT.
This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.
For details, click Finish to create the project and refer to the readme.txt file in the project directory.
The BSP for this template is based on the Altera HAL operating system.
For information about how this software example relates to Nios II hardware design examples,
< Back
Next >
Finish
Cancel

1-2 改寫程式。


Nios II - pwm1/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help
Project Explorer
altera.components
pwm1
Includes
hello_world.c
create-this-app
Makefile
pwm1_bsp [DE0_SOPC]
hello_world.c
#include <stdio.h>
#include <io.h>
#include "system.h"
int main()
{
int rx_char,data;
printf("Nios II PWM Lab\n");
printf("\nPlease enter an LED intensity between 1 to 4 (0 to exit)\n");
IOWR(MY_PWM_BASE,1,0xFF);
while (1)
{
scanf("%d",&data);
rx_char=data;

1-3 Run As/ Nios II Hareware



程式 1 內容

```
#include <io.h>
#include "system.h"
#include "unistd.h"
// #define IOWR_AVALON_PWM_DIVIDER(base, data) IOWR(base, 0, data)
// #define IOWR_AVALON_PWM_DUTY(base, data) IOWR(base, 1, data)


int main()
{

  IOWR(MY_PWM_BASE,0,0xFFFF);

  while (1)
     {
         IOWR(MY_PWM_BASE,1,0xFFFF);
       usleep(1000000);

         IOWR(MY_PWM_BASE,1,0xFFF);
       usleep(1000000);
```

```
        IOWR(MY_PWM_BASE,1,0xFF);
       usleep(1000000);

        IOWR(MY_PWM_BASE,1,0xF);
       usleep(1000000);

        IOWR(MY_PWM_BASE,1,0x0);
       usleep(1000000);
    }
  return 0;
}
```

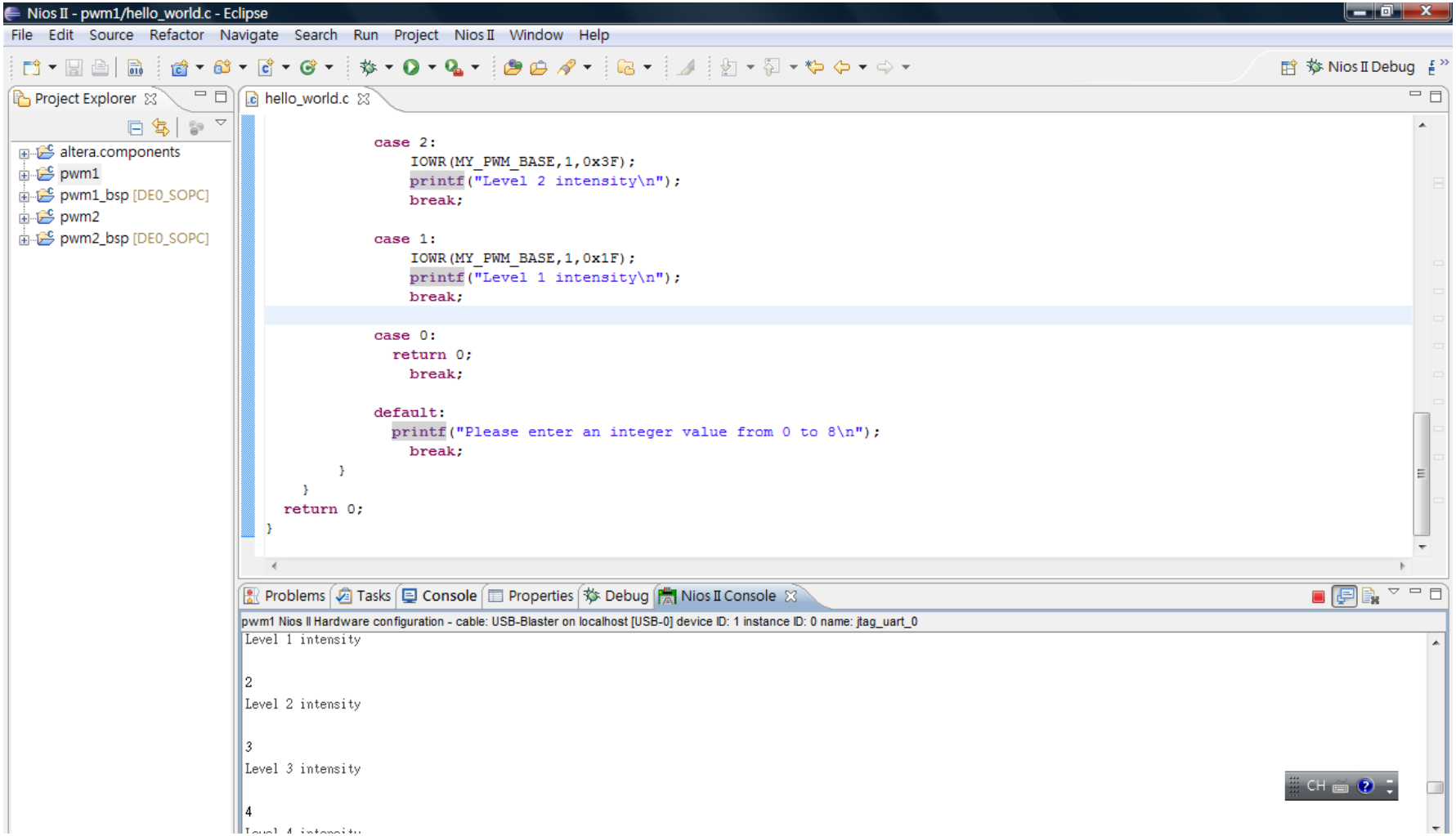
程式 2 內容

```
#include <stdio.h>
#include <io.h>
#include "system.h"

int main()
```

```
{
  int rx_char,data;

  printf("Nios II PWM Lab\n");
  printf("\nPlease enter an LED intensity between 1 to 8 (0 to exit)\n");

  IOWR(MY_PWM_BASE,0,0xFFFF);
//  IOWR(MY_PWM_BASE,1,0xFF);

  while (1)
     {
       scanf("%d",&data);
       rx_char = data;

         switch (rx_char)
         {
             case 8:
                 IOWR(MY_PWM_BASE,1,0xFFFF);
                 printf("Level 8 intensity\n");
                 break;

             case 7:
                 IOWR(MY_PWM_BASE,1,0x8FFF);
```

```
            printf("Level 7 intensity\n");
         break;

        case 6:
            IOWR(MY_PWM_BASE,1,0x3FFF);
            printf("Level 6 intensity\n");
            break;

        case 5:
            IOWR(MY_PWM_BASE,1,0x8FF);
            printf("Level 5 intensity\n");
            break;

         case 4:
             IOWR(MY_PWM_BASE,1,0x3FF);
             printf("Level 4 intensity\n");
             break;

         case 3:
             IOWR(MY_PWM_BASE,1,0x8F);
             printf("Level 3 intensity\n");
             break;
```

```
            case 2:
                IOWR(MY_PWM_BASE,1,0x3F);
                printf("Level 2 intensity\n");
                break;

            case 1:
                IOWR(MY_PWM_BASE,1,0x1F);
                printf("Level 1 intensity\n");
                break;

            case 0:
              return 0;
                break;

            default:
              printf("Please enter an integer value from 0 to 8\n");
                break;
        }
    }
  return 0;
}
```


Nios II - pwm1/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help
Nios II Debug
Project Explorer
altera.components
pwm1
pwm1_bsp [DE0_SOPC]
pwm2
pwm2_bsp [DE0_SOPC]
hello_world.c
case 2:
IOWR(MY_PWM_BASE,1,0x3F);
printf("Level 2 intensity\n");
break;
case 1:
IOWR(MY_PWM_BASE,1,0x1F);
printf("Level 1 intensity\n");
break;
case 0:
return 0;
break;
default:
printf("Please enter an integer value from 0 to 8\n");
break;
}
}
return 0;
}
Problems Tasks Console Properties Debug Nios II Console
pwm1 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0
Level 1 intensity
2
Level 2 intensity
3
Level 3 intensity
4
CH

定義客制化指令

Quartus 9.1 版

一、背景知識

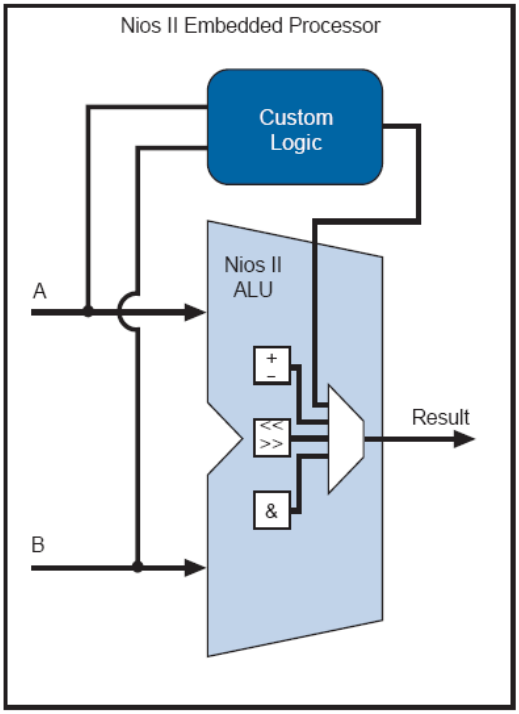
使用者自定義指令的概念 (資料取自和春技術學院線上教學課程)

使用使用者自定義指，使用者能夠向 NIOS II 處理器的算術邏輯單元(ALU)和指令集中增加使用者自定義功能，如下圖。完整的使用者自定義指令包括以下兩部分：

1. 使用者自定義邏輯：是完成使用者操作的硬體部份。這些邏輯作為 NIOS II 微處理器 ALU 的一部份。

2.軟體巨集：提供軟體介面使得使用者能夠存取使用者自定義邏輯。

當建立 NIOS II 嵌入式處理器時，使用者自定義邏輯將和 NIOS II 處理器的 ALU 結合在一起。NIOS II 配置嚮導還將建立相對應 C/C++和組譯的巨集程式碼，使得軟體可以存取這些使用者自行定義邏輯。如果使用者自定義指令用組合邏輯完成，則用來完成這個指令時脈週期的數目固定為 1，如果使用者自定義指令用時序邏輯完成，則必須指定時脈週期的數目。



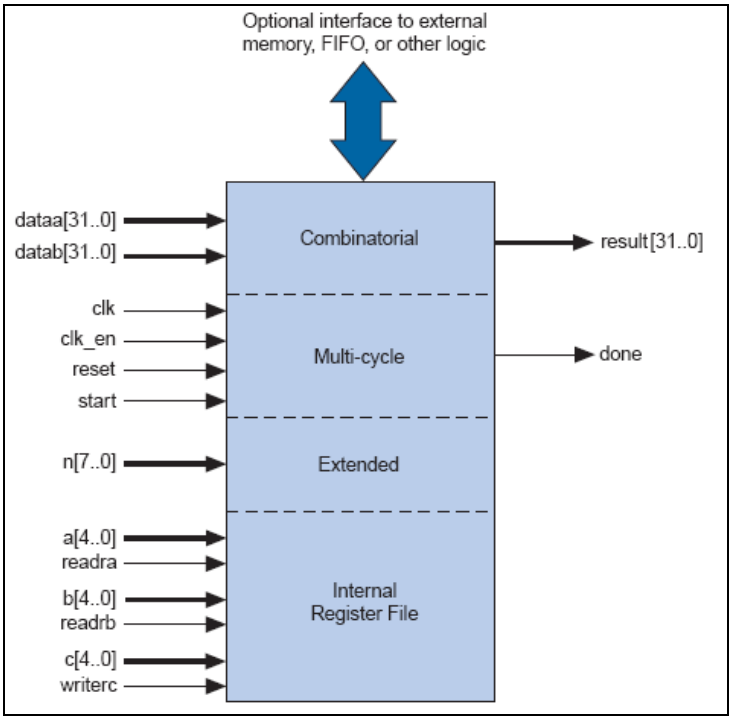
向 NIOS II ALU 中增加使用者自定義邏輯

硬體介面

使用者自定義指令支援多種設計檔，包括：
Verilog HDL
VHDL
EDIF netlist file
Quartus II Block Design File (.bdf)
Verilog Quartus Mapping File (.vqm)

使用者可以使用以下四個選項來實作使用者自定義邏輯：
1. 單週期邏輯選項
2. 多時脈邏輯選項
3. 參數化選項
4. 使用者自定義埠選項

由於使用者自定義邏輯需要直接連到 ALU 上，所以 NIOS II 提供一套預先定義好名稱和功能的介面如下圖。NIOS II 配置精靈會掃描使用者自定義邏輯，搜尋需要的埠，把這些埠連到 ALU 上。因此使用者自定義邏輯必須指定所有需要的埠類型(如單週期或多週期)。下表給出了每種操作類型所需的埠。必須使用預先定義的埠名稱，埠才能連到正確的介面上。



使用者自定義邏輯模組介面(32 位元 NIOS 處理器)

Table 1–1. Custom Instruction Architectural Types, Application & Hardware Interface

Architectural Type	Application	Hardware Interface
Combinatorial	Single clock cycle custom logic blocks	`dataa[31..0], datab[31..0], result[31..0]`
Multi-cycle	Multi clock cycle custom logic block of fixed or variable durations	`dataa[31..0], datab[31..0], result[31..0], clk, clk_en, start, reset, done`
Extended	Custom logic blocks that are capable of performing multiple operations	`dataa[31..0], datab[31..0], result[31..0], clk, clk_en, start, reset, done, n[7..0]`
Internal Register File	Custom logic blocks that access internal register file for input and/or output	`dataa[31..0], datab[31..0], result[31..0], clk, clk_en, start, reset, done, n[7..0], a[4..0], readra, b[4..0], readrb, c[4..0], writerc`
External Interface	Custom logic blocks that interface to logic outside of the NIOS II processor's data path	Standard custom instruction signals, plus user-defined interface to external logic.

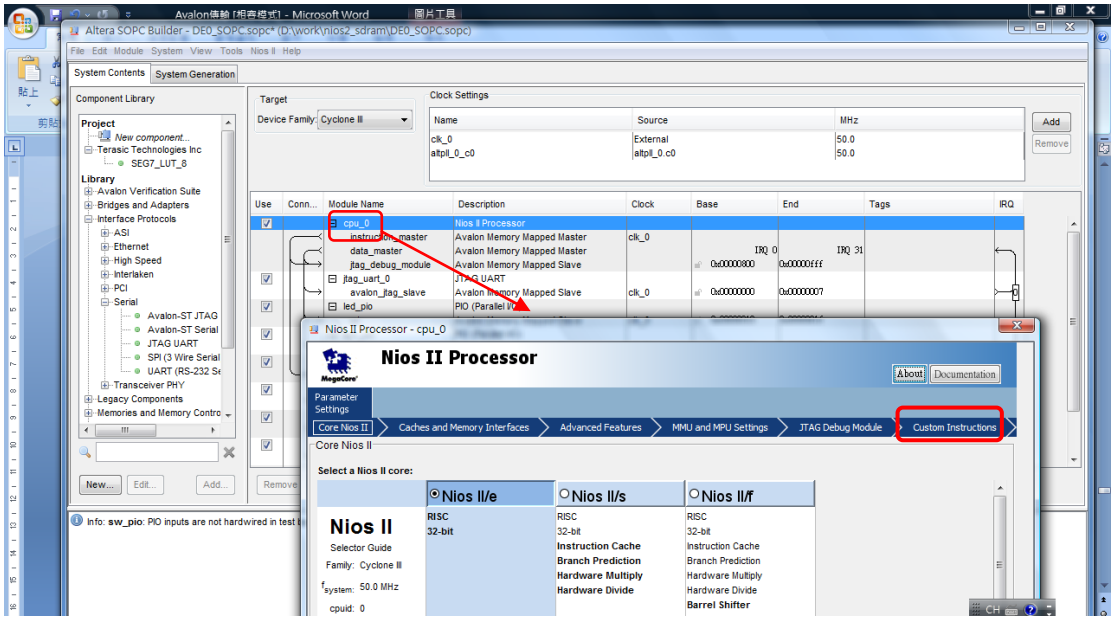
使用者自定義指令埠

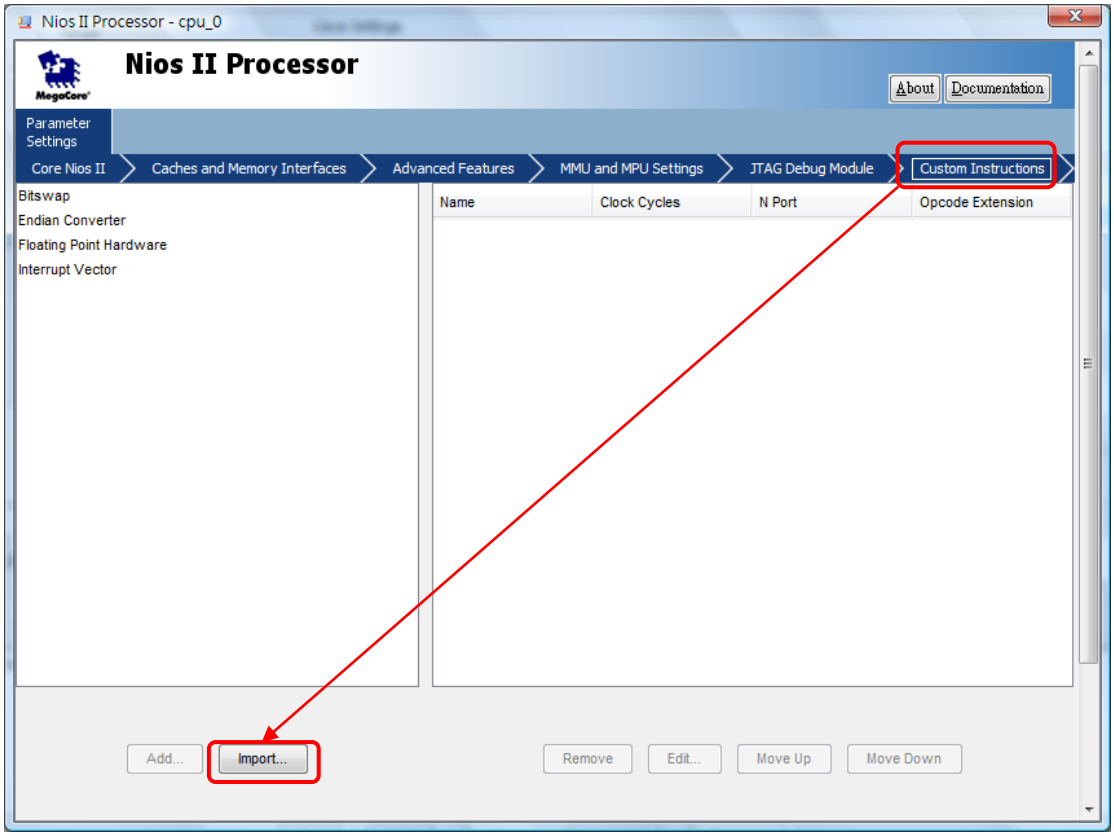
二、硬體設計

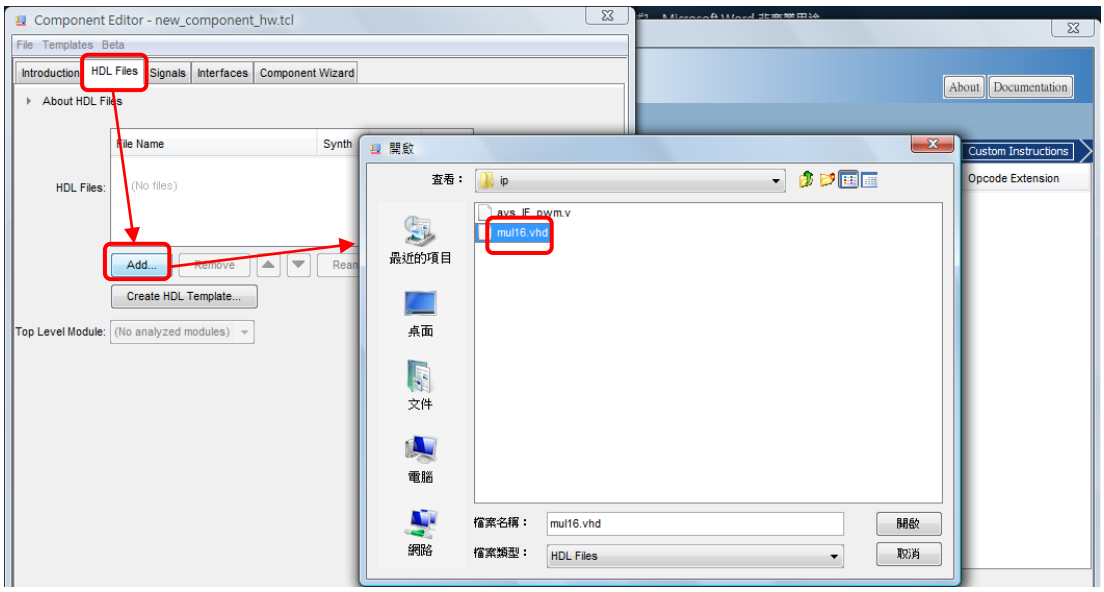
1. 在工作的 project 下建立一個資料夾 ip，並將 mul16.vhd 複製到 ip 資料夾內。

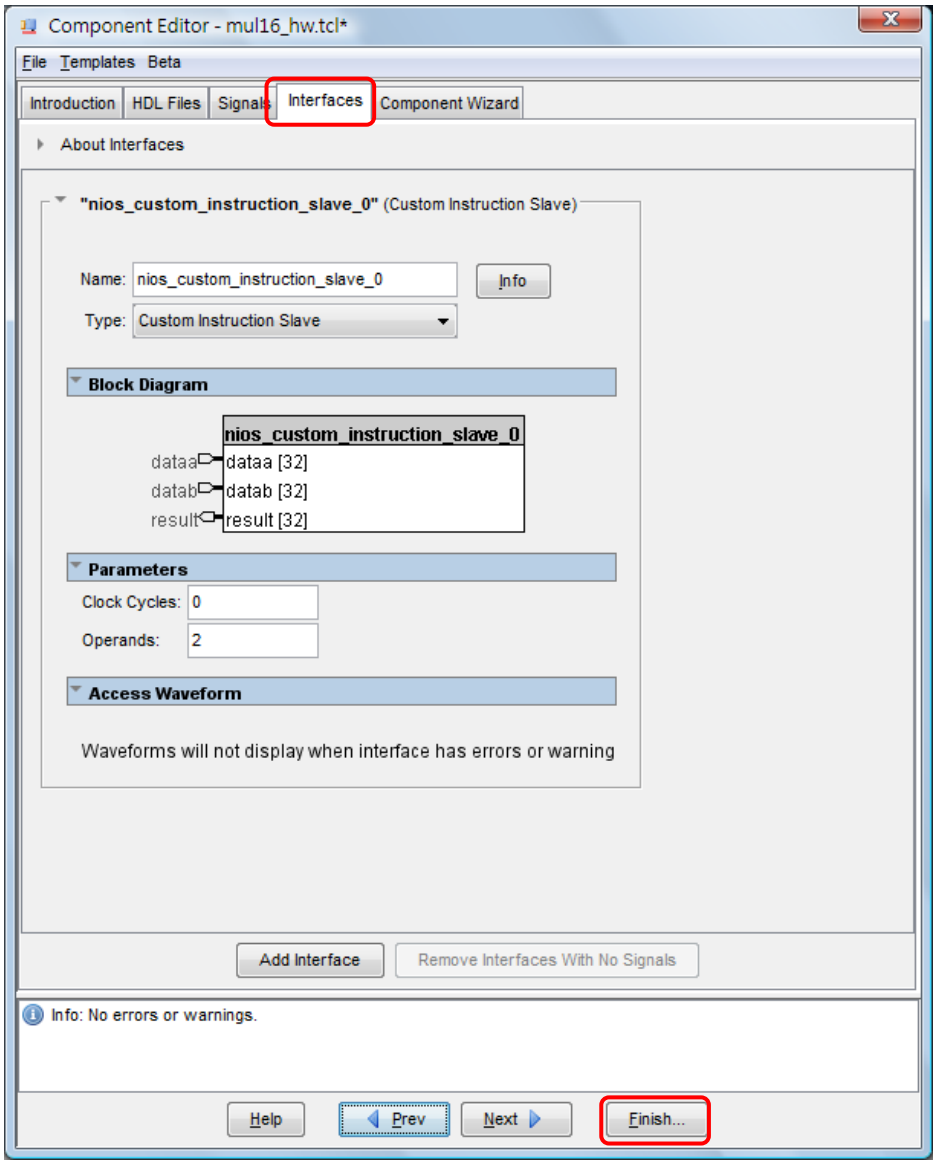
```
library IEEE;
use ieee.std_logic_1164.all;
USE ieee.std_logic_arith.ALL;
use ieee.std_logic_unsigned.all;

entity mul16 is
	port(
		dataa,datab: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
		result  : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
	);
END mul16;

ARCHITECTURE A OF mul16 is
BEGIN
		result<=  (dataa(15 downto 0))* (datab(15 downto 0));
END A;
```

2. 雙擊點選 cpu_0 元件，選擇 Custom Instruction 標籤。選擇 Import/Add。

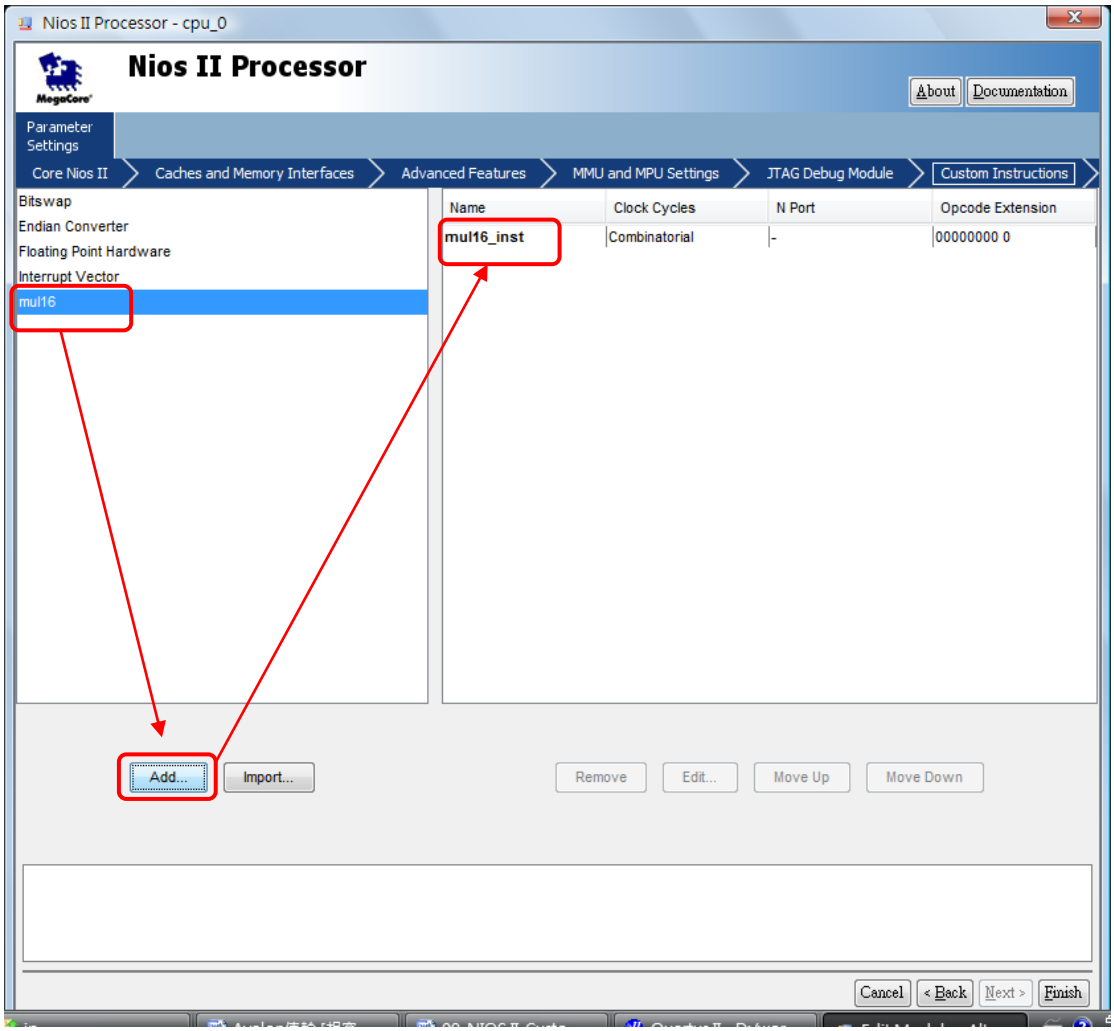

Altera SOPC Builder - DE0_SOPC.sopc* (D:\work\nios2_sdram\DE0_SOPC.sopc)
System Contents
System Generation
Component Library
Target
Device Family: Cyclone II
Clock Settings
Name
Source
MHz
clk_0
External
50.0
altpll_0_c0
altpll_0.c0
50.0
Add
Remove
Use
Conn...
Module Name
Description
Clock
Base
End
Tags
IRQ
cpu_0
Nios II Processor
instruction_master
Avalon Memory Mapped Master
clk_0
data_master
Avalon Memory Mapped Master
jtag_debug_module
Avalon Memory Mapped Slave
jtag_uart_0
JTAG UART
avalon_jtag_slave
Avalon Memory Mapped Slave
clk_0
led_pio
PIO (Parallel I/O)
Nios II Processor - cpu_0
Nios II Processor
About
Documentation
Parameter Settings
Core Nios II
Caches and Memory Interfaces
Advanced Features
MMU and MPU Settings
JTAG Debug Module
Custom Instructions
Core Nios II
Select a Nios II core:
Nios II/e
Nios II/s
Nios II/f
Nios II
Selector Guide
Family: Cyclone II
RISC
32-bit
Instruction Cache
Branch Prediction
Hardware Multiply
Hardware Divide
Barrel Shifter


Nios II Processor - cpu_0
Nios II Processor
About
Documentation
Parameter Settings
Core Nios II
Caches and Memory Interfaces
Advanced Features
MMU and MPU Settings
JTAG Debug Module
Custom Instructions
Bitswap
Endian Converter
Floating Point Hardware
Interrupt Vector
Name
Clock Cycles
N Port
Opcode Extension
Add...
Import...
Remove
Edit...
Move Up
Move Down


Component Editor - new_component_hw.tcl
File Templates Beta
Introduction
HDL Files
Signals
Interfaces
Component Wizard
About HDL Files
File Name
Synth
HDL Files:
(No files)
Add...
Remove
Create HDL Template...
Top Level Module:
(No analyzed modules)
About
Documentation
Custom Instructions
Opcode Extension
開啟
查看：
ip
avs_IF_pwm.v
mul16.vhd
最近的項目
桌面
文件
電腦
網路
檔案名稱：
mul16.vhd
開啟
檔案類型：
HDL Files
取消

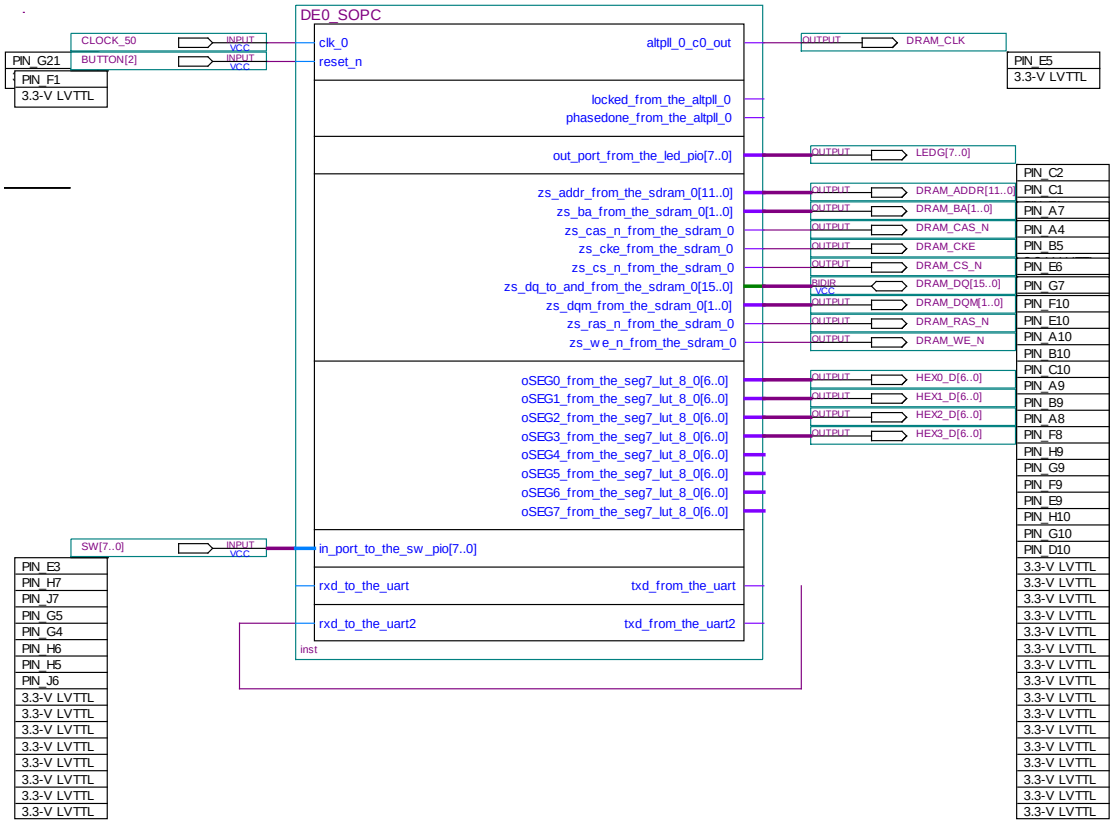

Component Editor - mul16_hw.tcl*
File Templates Beta
Introduction
HDL Files
Signals
Interfaces
Component Wizard
About Interfaces
"nios_custom_instruction_slave_0" (Custom Instruction Slave)
Name: nios_custom_instruction_slave_0
Info
Type: Custom Instruction Slave
Block Diagram
nios_custom_instruction_slave_0
dataa
datab
result
dataa [32]
datab [32]
result [32]
Parameters
Clock Cycles: 0
Operands: 2
Access Waveform
Waveforms will not display when interface has errors or warning
Add Interface
Remove Interfaces With No Signals
Info: No errors or warnings.
Help
Prev
Next
Finish...

3. 再次雙擊點選 cpu_0 元件，選擇 Custom Instruction 標籤。增加 mul16 元件。



4. 按 Generate 重新產生系統。

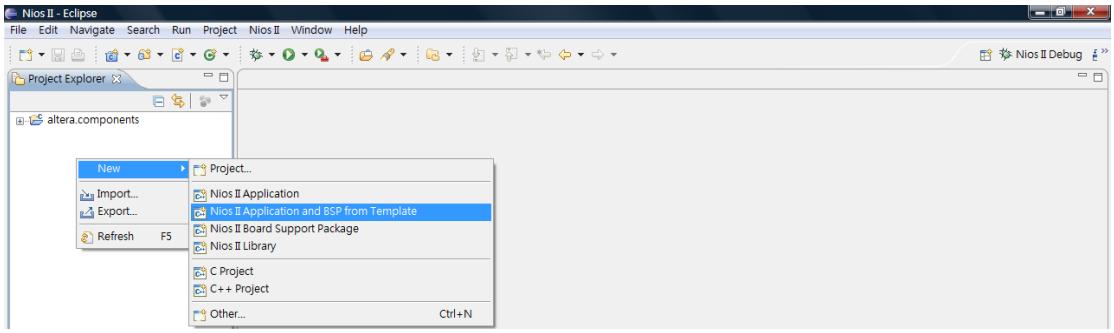
5. 在 Quartus II 中更新 Symbol，並點選 Start Compilation 重新編譯電路。



三、Nios II EDS 軟體設計

1. 在 Nios II DES 中建立新專案。

1-1 File/ New/ Nios II Application and BSP from Template



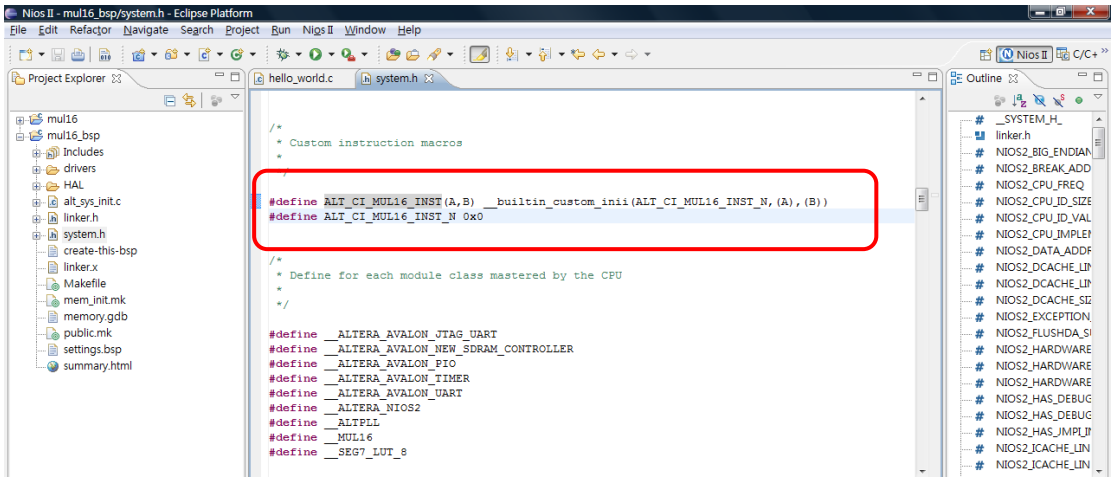
1-2 程式：

```c
#include <stdio.h>
#include <system.h>

int main()
{
  int x=101;
  int y=25;
  int z;

  z=ALT_CI_MUL16_INST(x,y);
  printf("x=%d, y=%d, z=x*y=%d\n",x,y,z);
  return 0;
}
```

在 system.h 中，有關於客制化指令的定義:

```c
/*
 * Custom instruction macros
 *
 */

#define ALT_CI_MUL16_INST(A,B) __builtin_custom_inii(ALT_CI_MUL16_INST_N,(A),(B))
#define ALT_CI_MUL16_INST_N 0x0
```

這一行指令是將__builtin_custom_inii(0,A,B)客制化指令定義為 ALT_CI_MUL16_INST(A,B)，可增加程式的可讀性。



```
z = ALT_CI_MUL16_INST(x,y);
```

這一行指令是呼叫客制化指令 ALT_CI_MUL16_INST，並將 x 及 y 當個參數傳入 ALU 中的乘法器。

1-3 Run As/ Nios II Hareware