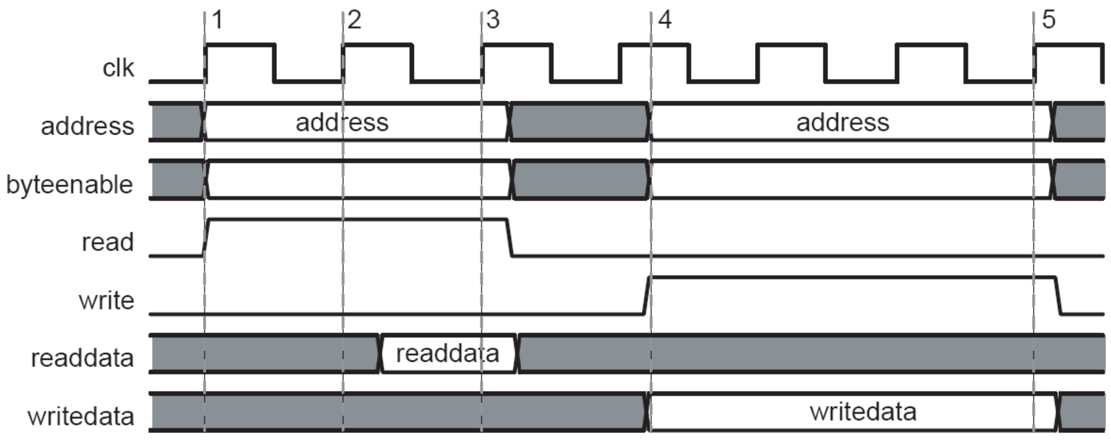
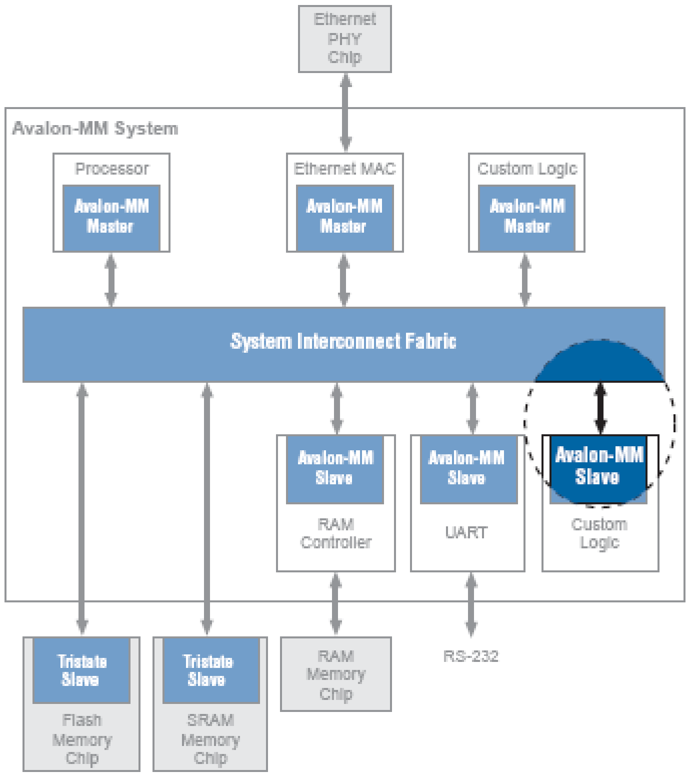
硬體樂透產生器

一、背景知識



基本從埠讀寫傳輸時序圖

請看上圖 clk 的狀態，匯流排在 clk 1 為 high 的時候輸出，在 clk 2 為 low 的時候結束，不插入等待週期。傳輸在一個週期內完成，目標週邊裝置必須能夠馬上且非同步向 Avalon 匯流排模組輸出相對應位址的資料。在 clk 第一個上升緣時，匯流排向 address 、byteenable_n 和 read_n 傳遞信號。匯流排對 address 進行解碼，便會驅動從埠的 chipselect 信號。如果信號正確，從埠在資料有效時會立刻驅動 readdata 輸出。匯流排會在下一個時脈上升緣 clk 3 時捕獲 readdata。



使用者邏輯與系統模組一起整合在 Avalon PLD 之中

Table 3–1. Avalon-MM Slave Port Signals (1) (Part 1 of 4)

Signal Type	Width	Dir	Req'd	Description
Fundamental Signals				
`read` `read_n`	1	In	No	Asserted to indicate a `read` transfer. If present, `readdata` is required.
`write` `write_n`	1	In	No	Asserted to indicate a `write` transfer. If present, `writedata` is required.
`address`	1-32	In	No	Specifies an offset into the slave address space. Each slave address value selects a word of slave data. For example, `address`= 0 selects the first <*slave data width*> bits of slave data; `address`=1 selects the second <*slave data width*> bits of slave data.
`readdata`	8,16,32, 64,128, 256,512, 1024	Out	No	The readdata provided by the slave in response to a `read` transfer.

Table 3–1. Avalon-MM Slave Port Signals (1) (Part 2 of 4)

Signal Type	Width	Dir	Req'd	Description
`writedata`	8,16,32,| 64,128,| 256,512, 1024	In	No	Data from the system interconnect fabric for write transfers. The width must be the same as the width of `readdata` if both are present.
`byteenable` `byteenable_n`	1,2,4,8, 16, 32, 64, 128	In	No	Enables specific byte lane(s) during transfers. Each bit in `byteenable` corresponds to a byte in `writedata` and `readdata`. During writes, byteenables specify which bytes are being written to; other bytes should be ignored by the slave. During reads, byteenables indicates which bytes the master is reading. Slaves that simply return `readdata` with no side effects are free to ignore byteenables during reads. When more than one bit is asserted, all asserted lanes are adjacent. The number of adjacent lines must be a power of two, and the specified bytes must be aligned on an address boundary for the size of the data. The following values are legal for a 32-bit slave: 1111 writes full 32 bits 0011 writes lower 2 bytes 1100 writes upper 2 bytes 0001 writes byte 0 only 0010 writes byte 1 only 0100 writes byte 2 only 1000 writes byte 3 only
`begintransfer`	1	In	No	Asserted by the system interconnect fabric for the first cycle of each transfer regardless of `waitrequest` and other signals.
Wait-State Signals				
`waitrequest` `waitrequest_n`	1	Out	No	Asserted by the slave when it is unable to respond to a `read` or `write` request. When asserted, the control signals to the slave, with the exception of `begintransfer` and `beginbursttransfer`, remain constant, as is illustrated by Figure 3–7 on page 3–13. An Avalon-MM slave may assert `waitrequest` during idle cycles. An Avalon-MM master may initiate a transaction when `waitrequest` is asserted. The design of Avalon-MM slaves must take these possibilities into account.

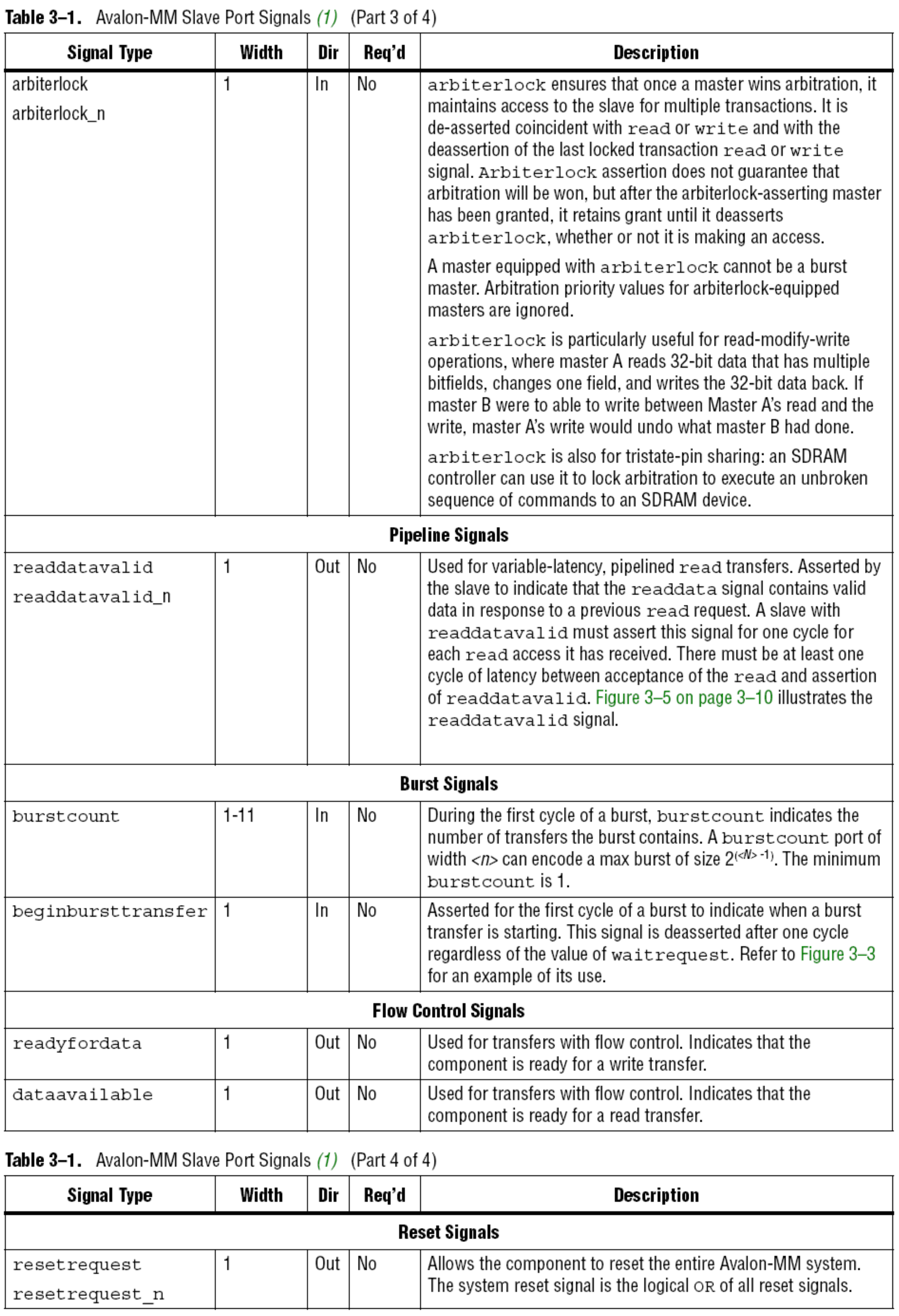


Table 3–1. Avalon-MM Slave Port Signals *(1)* (Part 3 of 4)

Signal Type	Width	Dir	Req'd	Description
arbiterlock arbiterlock_n	1	In	No	`arbiterlock` ensures that once a master wins arbitration, it maintains access to the slave for multiple transactions. It is de-asserted coincident with `read` or `write` and with the deassertion of the last locked transaction `read` or `write` signal. `Arbiterlock` assertion does not guarantee that arbitration will be won, but after the arbiterlock-asserting master has been granted, it retains grant until it deasserts `arbiterlock`, whether or not it is making an access. A master equipped with `arbiterlock` cannot be a burst master. Arbitration priority values for arbiterlock-equipped masters are ignored. `arbiterlock` is particularly useful for read-modify-write operations, where master A reads 32-bit data that has multiple bitfields, changes one field, and writes the 32-bit data back. If master B were to able to write between Master A's read and the write, master A's write would undo what master B had done. `arbiterlock` is also for tristate-pin sharing: an SDRAM controller can use it to lock arbitration to execute an unbroken sequence of commands to an SDRAM device.
Pipeline Signals				
`readdatavalid` `readdatavalid_n`	1	Out	No	Used for variable-latency, pipelined `read` transfers. Asserted by the slave to indicate that the `readdata` signal contains valid data in response to a previous `read` request. A slave with `readdatavalid` must assert this signal for one cycle for each `read` access it has received. There must be at least one cycle of latency between acceptance of the `read` and assertion of `readdatavalid`. Figure 3–5 on page 3–10 illustrates the `readdatavalid` signal.
Burst Signals				
`burstcount`	1-11	In	No	During the first cycle of a burst, `burstcount` indicates the number of transfers the burst contains. A `burstcount` port of width *<n>* can encode a max burst of size $2^{(<N> -1)}$. The minimum `burstcount` is 1.
`beginbursttransfer`	1	In	No	Asserted for the first cycle of a burst to indicate when a burst transfer is starting. This signal is deasserted after one cycle regardless of the value of `waitrequest`. Refer to Figure 3–3 for an example of its use.
Flow Control Signals				
`readyfordata`	1	Out	No	Used for transfers with flow control. Indicates that the component is ready for a write transfer.
`dataavailable`	1	Out	No	Used for transfers with flow control. Indicates that the component is ready for a read transfer.

Table 3–1. Avalon-MM Slave Port Signals *(1)* (Part 4 of 4)

Signal Type	Width	Dir	Req'd	Description
Reset Signals				
`resetrequest` `resetrequest_n`	1	Out	No	Allows the component to reset the entire Avalon-MM system. The system reset signal is the logical OR of all reset signals.

Avalon 從埠信號

二、硬體設計

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

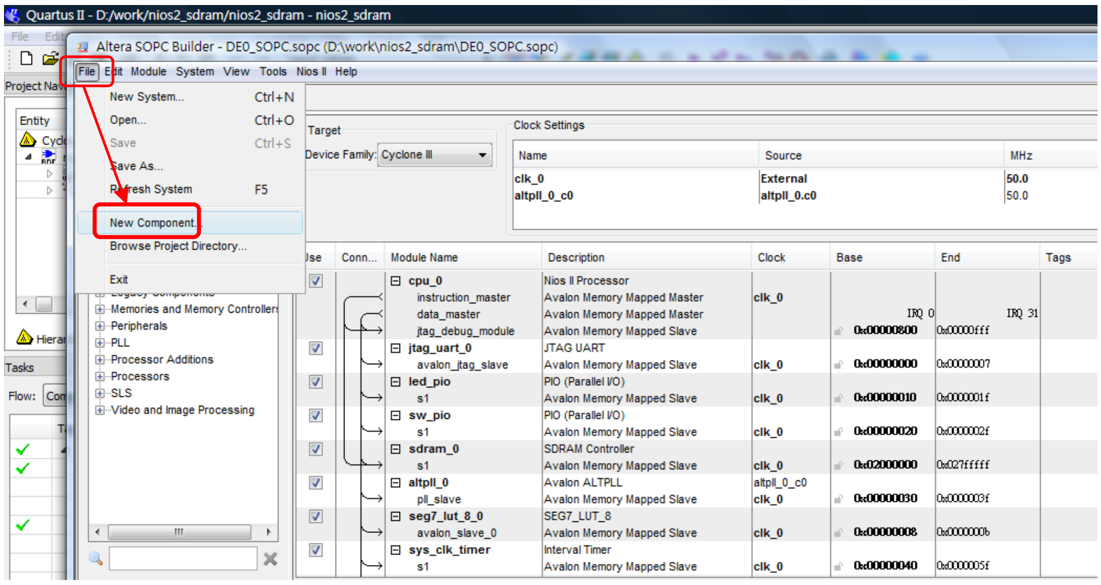
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
 use ieee.std_logic_unsigned.all;

 entity random_lotto is
    port(
            clk : in    std_logic;
            rst : in    std_logic;

            q_one: out std_logic_vector(3 downto 0);
            q_ten: out std_logic_vector(3 downto 0)
         );
 end random_lotto;

 architecture a of random_lotto is
    signal cnt_one,cnt_ten : std_logic_vector(3 downto 0);

 begin
    process (clk,rst)
    begin
      if rst='1' then
         cnt_one <= "0001";
         cnt_ten <= "0000";

      elsif clk'event and clk='1' then

         if cnt_ten = "0100" then        --40
            if cnt_one = "0010" then
               cnt_one <= "0001";            --1
               cnt_ten <= "0000";
            else
               cnt_one <= cnt_one + 1;
            end if;
         elsif cnt_one = "1001" then
```

```
                cnt_one <= "0000";
                cnt_ten <= cnt_ten + 1;
            else
                cnt_one <= cnt_one + 1;
            end if;
      end if;
    end process;

    process(clk,rst)
    begin
        if clk'event and clk='1' then
            if rst='1' then
                q_one <= "0000";
                q_ten <= "0000";

            else
                q_one <= cnt_one;
                q_ten <= cnt_ten;
  end if;
  end if;
  end process;
  end a;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned.all;

  entity wrapper_lotto is
      port(
          clk           : in    std_logic;
          reset         : in    std_logic;
          chipselect    : in    std_logic;
          read_n                : in    std_logic;

          readdata      : out   std_logic_vector(7 downto 0)
              );
```

```
end wrapper_lotto;

architecture a of wrapper_lotto is
component random_lotto
   port(
            clk : in    std_logic;
            rst : in    std_logic;
            q_one: out std_logic_vector(3 downto 0);
            q_ten: out std_logic_vector(3 downto 0)
         );
end component;
      signal temp_one,temp_ten:std_logic_vector(3 downto 0);
begin
      u0: random_lotto port map(clk,reset,temp_one,temp_ten);
      process(clk,read_n)
      begin
             if(clk'event and clk = '1' and read_n='0') then
                            readdata<=temp_ten&temp_one;
             end if;
      end process;

end a;
```

2. File/ New Component。按住 Ctrl 鍵點選 random_lotto.vhd 及 wrapper_lotto.vhd 兩個檔案，按開啟鈕匯入這兩個硬體描述檔。

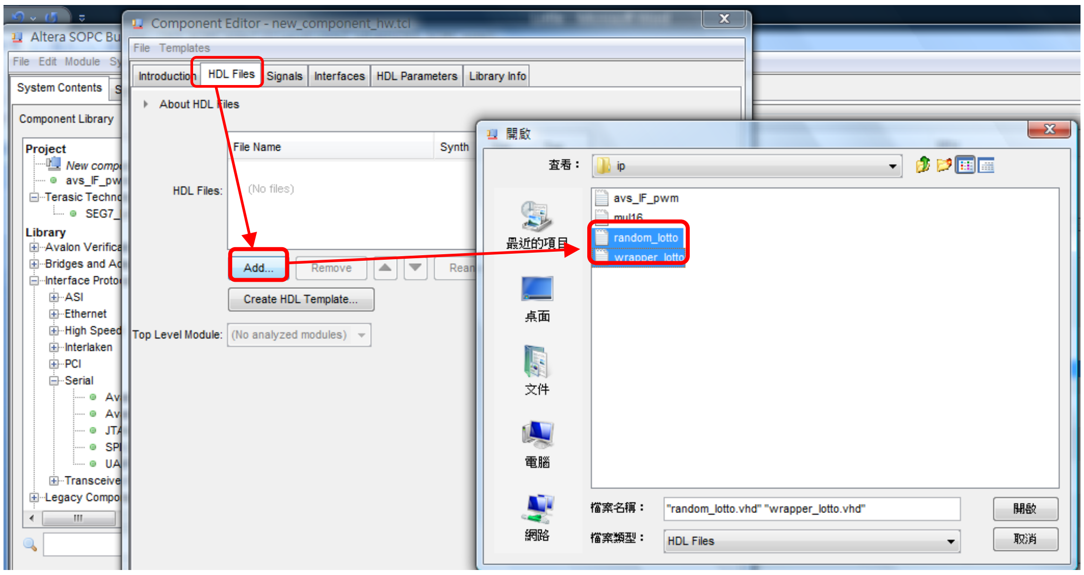

Quartus II - D:/work/nios2_sdram/nios2_sdram - nios2_sdram
Altera SOPC Builder - DE0_SOPC.sopc (D:\work\nios2_sdram\DE0_SOPC.sopc)
File Edit Module System View Tools Nios II Help
New System... Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As...
Refresh System F5
New Component...
Browse Project Directory...
Exit
Memories and Memory Controllers
Peripherals
PLL
Processor Additions
Processors
SLS
Video and Image Processing

Target
Device Family: Cyclone III

Clock Settings

Name	Source	MHz
clk_0	External	50.0
altpll_0_c0	altpll_0.c0	50.0

Use	Conn...	Module Name	Description	Clock	Base	End	Tags
✓		cpu_0	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	clk_0			
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave		0x00000800	0x00000fff	
✓		jtag_uart_0	JTAG UART				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00000000	0x00000007	
✓		led_pio	PIO (Parallel I/O)				
		s1	Avalon Memory Mapped Slave	clk_0	0x00000010	0x0000001f	
✓		sw_pio	PIO (Parallel I/O)				
		s1	Avalon Memory Mapped Slave	clk_0	0x00000020	0x0000002f	
✓		sdram_0	SDRAM Controller				
		s1	Avalon Memory Mapped Slave	clk_0	0x02000000	0x027fffff	
✓		altpll_0	Avalon ALTPLL	altpll_0_c0			
		pll_slave	Avalon Memory Mapped Slave	clk_0	0x00000030	0x0000003f	
✓		seg7_lut_8_0	SEG7_LUT_8				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00000008	0x0000000b	
✓		sys_clk_timer	Interval Timer				
		s1	Avalon Memory Mapped Slave	clk_0	0x00000040	0x0000005f	



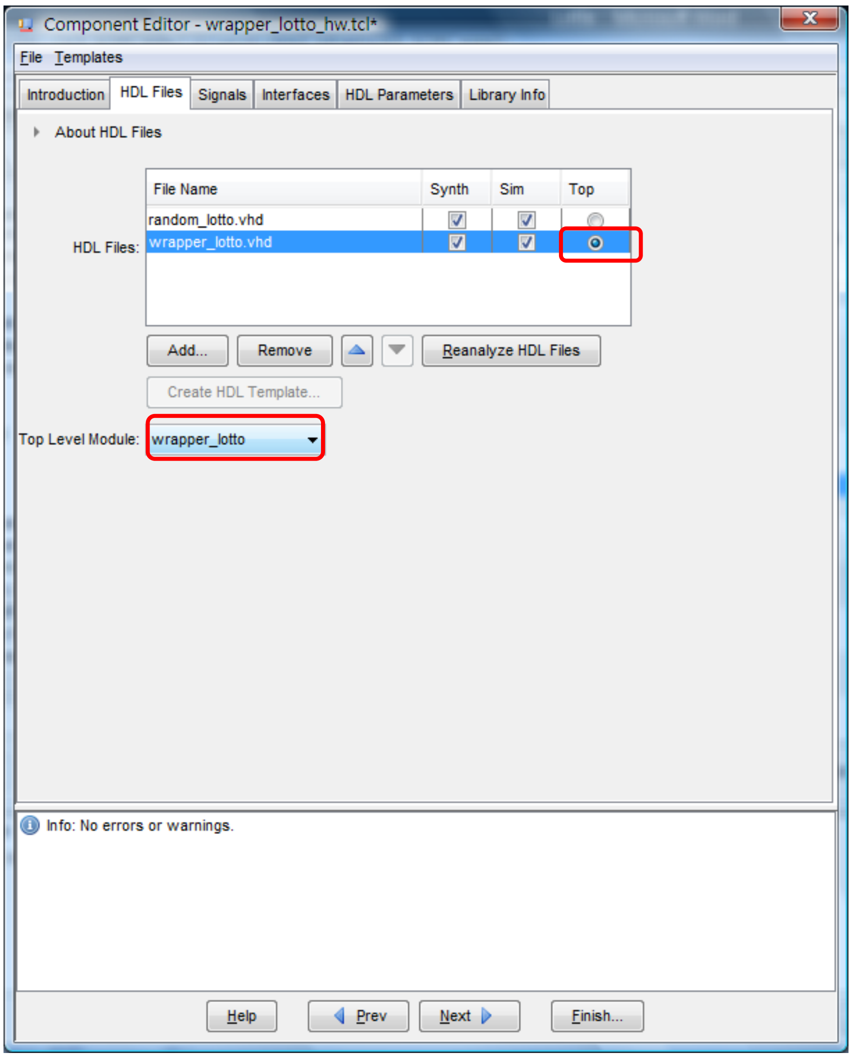
HDL Files: (No files)

Top Level Module: (No analyzed modules)

檔案名稱： "random_lotto.vhd" "wrapper_lotto.vhd"

檔案類型： HDL Files

3. 將 Top Level Module 設為 random_lotto，指定最上層的模組，此時 Sopc Builder 會檢查輸出入埠是否符合 Avalon Bus 的規範。


Component Editor - wrapper_lotto_hw.tcl*
File Templates
Introduction HDL Files Signals Interfaces HDL Parameters Library Info
About HDL Files

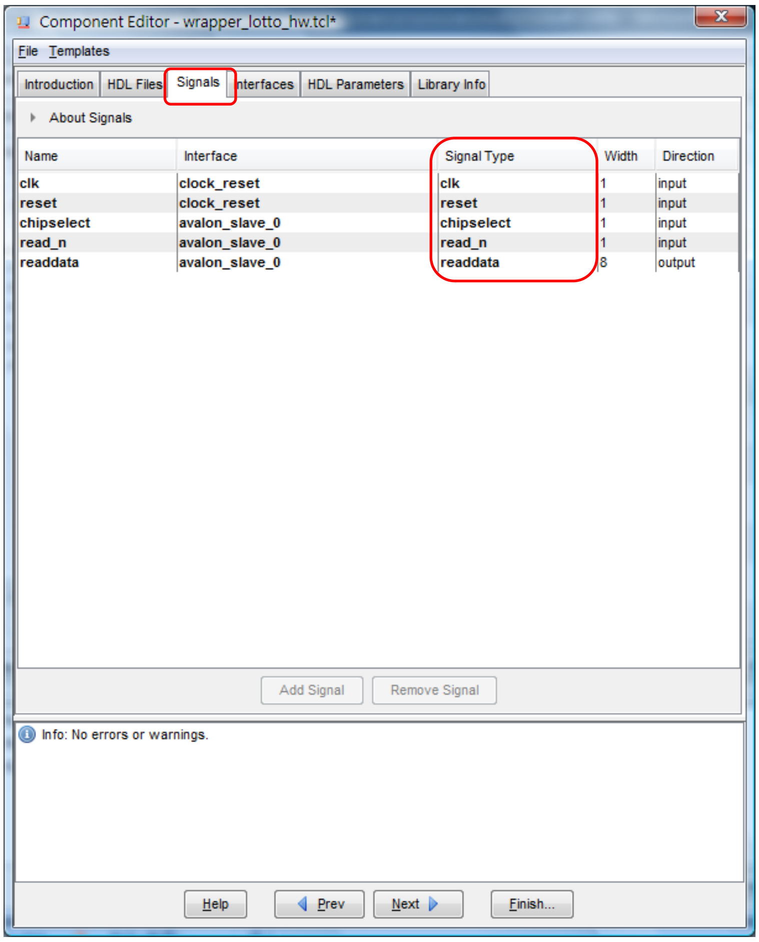
HDL Files:

File Name	Synth	Sim	Top
random_lotto.vhd	✓	✓	○
wrapper_lotto.vhd	✓	✓	●

Add... Remove Reanalyze HDL Files
Create HDL Template...

Top Level Module: wrapper_lotto

Info: No errors or warnings.

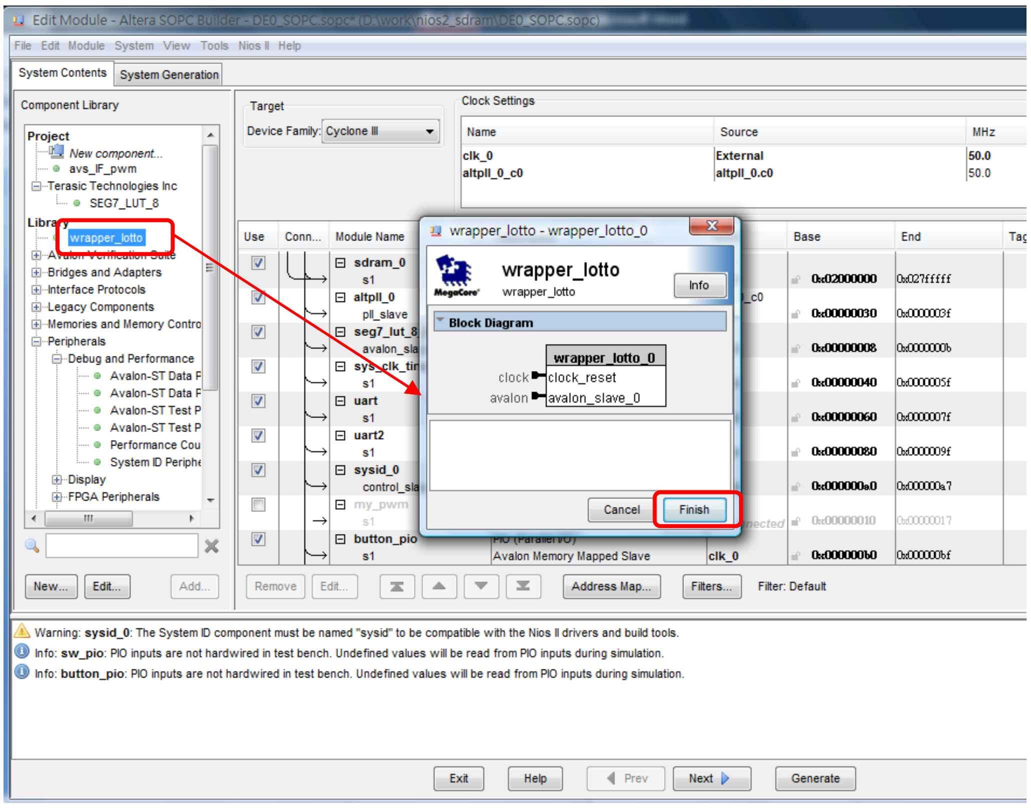


Name	Interface	Signal Type	Width	Direction
clk	clock_reset	clk	1	input
reset	clock_reset	reset	1	input
chipselect	avalon_slave_0	chipselect	1	input
read_n	avalon_slave_0	read_n	1	input
readdata	avalon_slave_0	readdata	8	output

Info: No errors or warnings.

4. 按下 Finish 完成設定。

5. Add “wrapper_lotto” component，Rename the peripheral to lotto. Click “Generate”.



Target

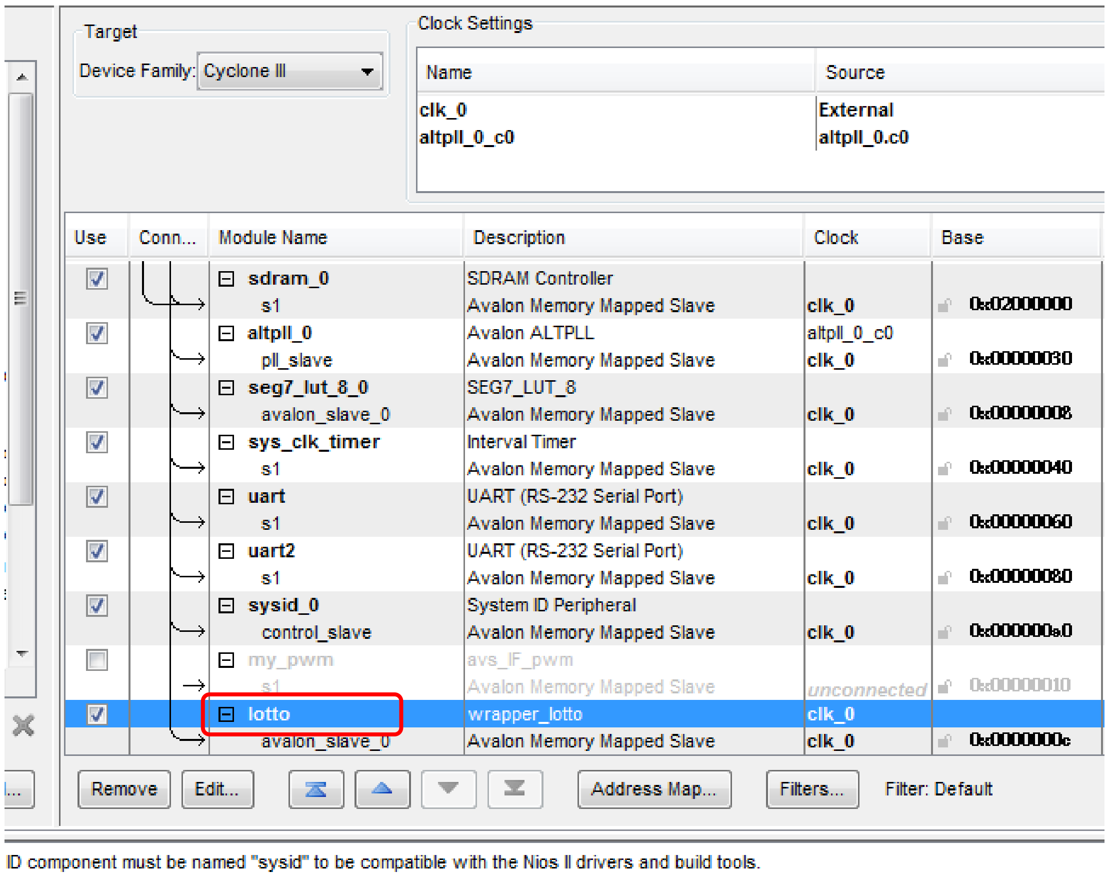
Device Family: Cyclone III

Clock Settings

Name	Source	MHz
clk_0	External	50.0
altpll_0_c0	altpll_0.c0	50.0

Use	Conn...	Module Name	Base	End
✓		sdram_0		
		s1	0x02000000	0x027fffff
✓		altpll_0		
		pll_slave	0x00000030	0x0000003f
✓		seg7_lut_8		
		avalon_sla	0x00000008	0x0000000b
✓		sys_clk_tir		
		s1	0x00000040	0x0000005f
✓		uart		
		s1	0x00000060	0x0000007f
✓		uart2		
		s1	0x00000080	0x0000009f
✓		sysid_0		
		control_sla	0x000000a0	0x000000a7
		my_pwm		
		s1	0x00000010	0x00000017
✓		button_pio		
		s1	0x000000b0	0x000000bf

Warning: **sysid_0**: The System ID component must be named "sysid" to be compatible with the Nios II drivers and build tools.
Info: **sw_pio**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info: **button_pio**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.



Target

Device Family: Cyclone III

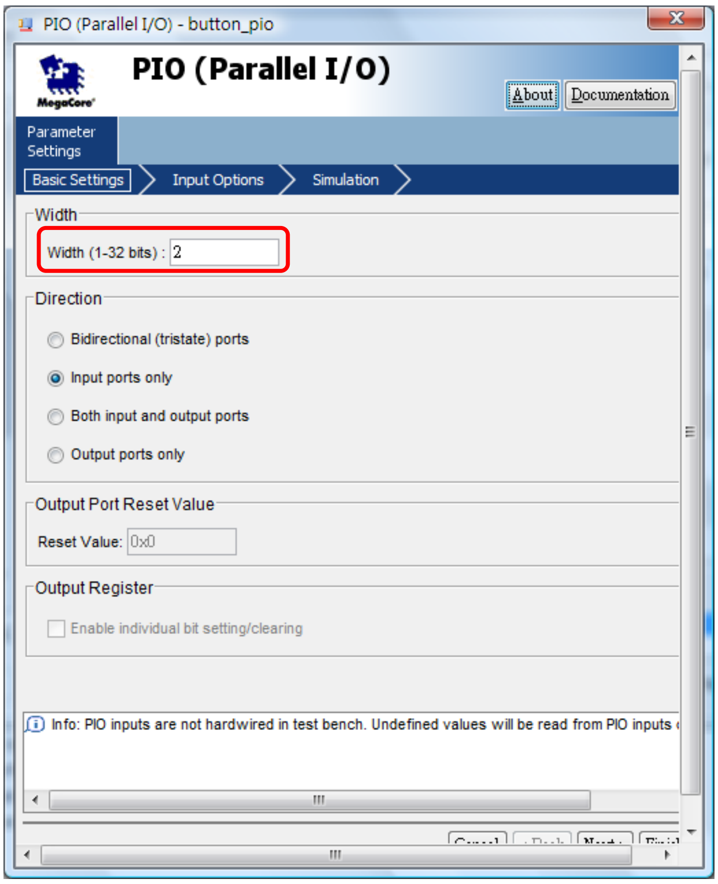
Clock Settings

Name	Source
clk_0	External
altpll_0_c0	altpll_0.c0

Use	Conn...	Module Name	Description	Clock	Base
✓		sdram_0	SDRAM Controller		
		s1	Avalon Memory Mapped Slave	clk_0	0x02000000
✓		altpll_0	Avalon ALTPLL	altpll_0_c0	
		pll_slave	Avalon Memory Mapped Slave	clk_0	0x00000030
✓		seg7_lut_8_0	SEG7_LUT_8		
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00000008
✓		sys_clk_timer	Interval Timer		
		s1	Avalon Memory Mapped Slave	clk_0	0x00000040
✓		uart	UART (RS-232 Serial Port)		
		s1	Avalon Memory Mapped Slave	clk_0	0x00000060
✓		uart2	UART (RS-232 Serial Port)		
		s1	Avalon Memory Mapped Slave	clk_0	0x00000080
✓		sysid_0	System ID Peripheral		
		control_slave	Avalon Memory Mapped Slave	clk_0	0x000000a0
		my_pwm	avs_IF_pwm		
		s1	Avalon Memory Mapped Slave	unconnected	0x00000010
✓		lotto	wrapper_lotto	clk_0	
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x0000000c

ID component must be named "sysid" to be compatible with the Nios II drivers and build tools.

6. 新增 2 bits 的 pio，並設定為 rising edge 觸發。



Width

Width (1-32 bits) : 2

Direction

Bidirectional (tristate) ports [✓ Input ports only] Both input and output ports Output ports only

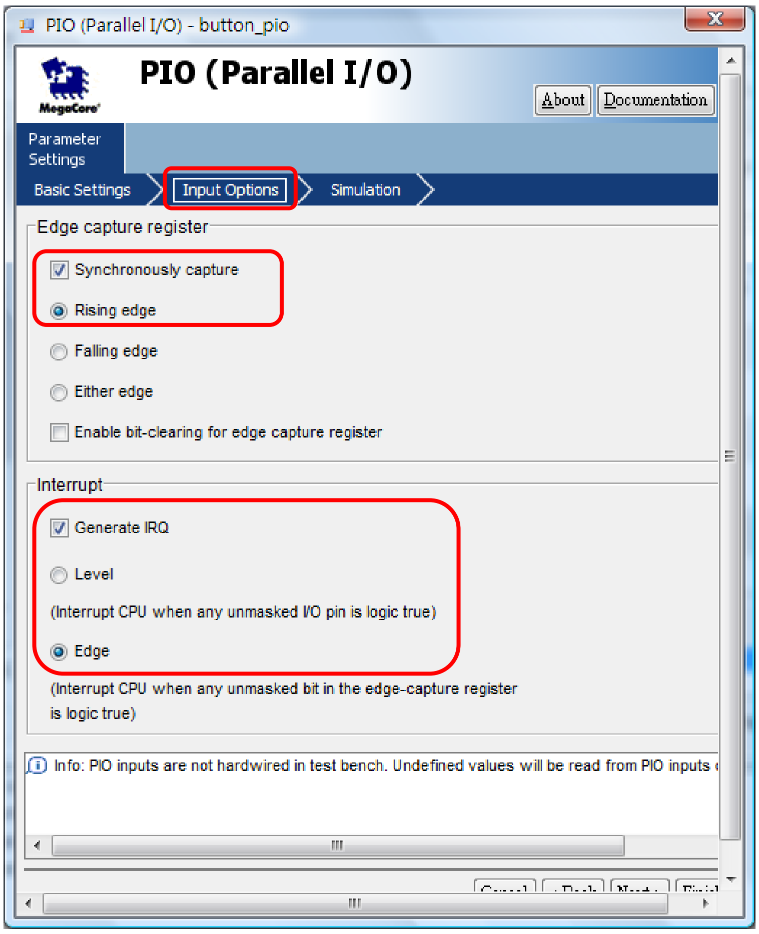
Output Port Reset Value

Reset Value: 0x0

Output Register

Enable individual bit setting/clearing

Info: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs


PIO (Parallel I/O) - button_pio

Edge capture register

[✓] Synchronously capture

(●) Rising edge

( ) Falling edge

( ) Either edge

[ ] Enable bit-clearing for edge capture register

Interrupt

[✓] Generate IRQ

( ) Level

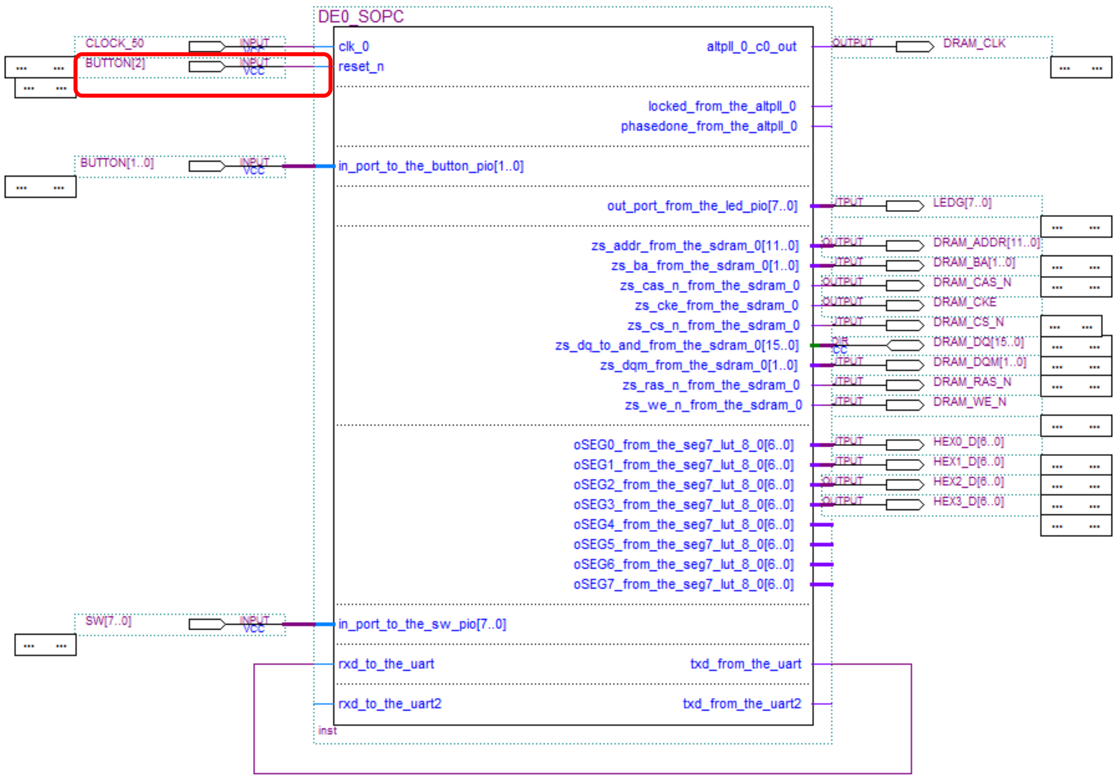
(Interrupt CPU when any unmasked I/O pin is logic true)

(●) Edge

(Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

Info: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs

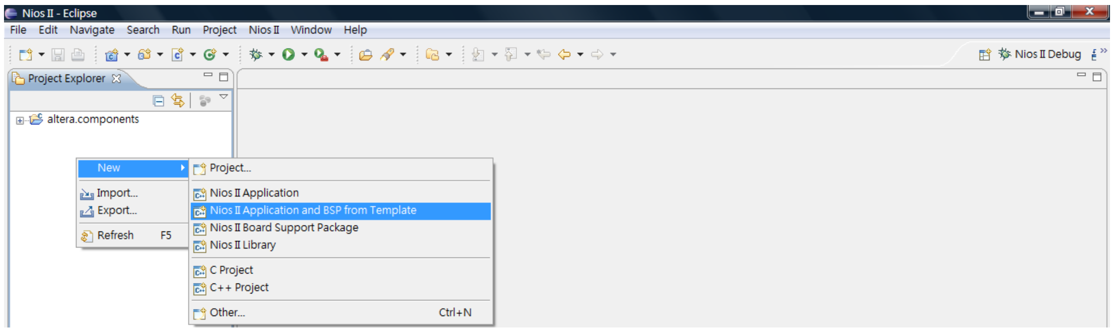
7. 按滑鼠右鍵選擇 Update Symbol or Block 更新 Nios II Symbol。重新整理連線後點選 Start Compilation 編譯電路。


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

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

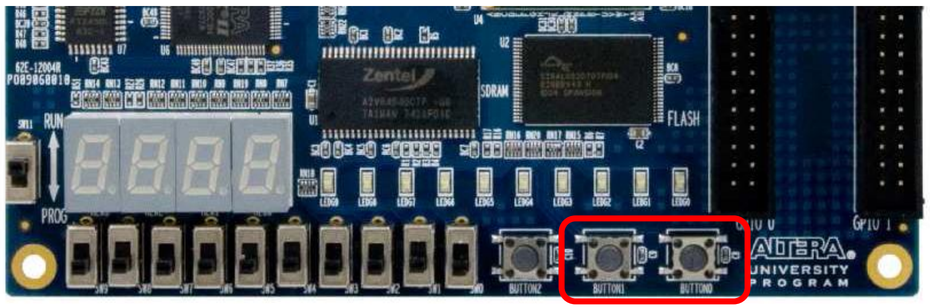


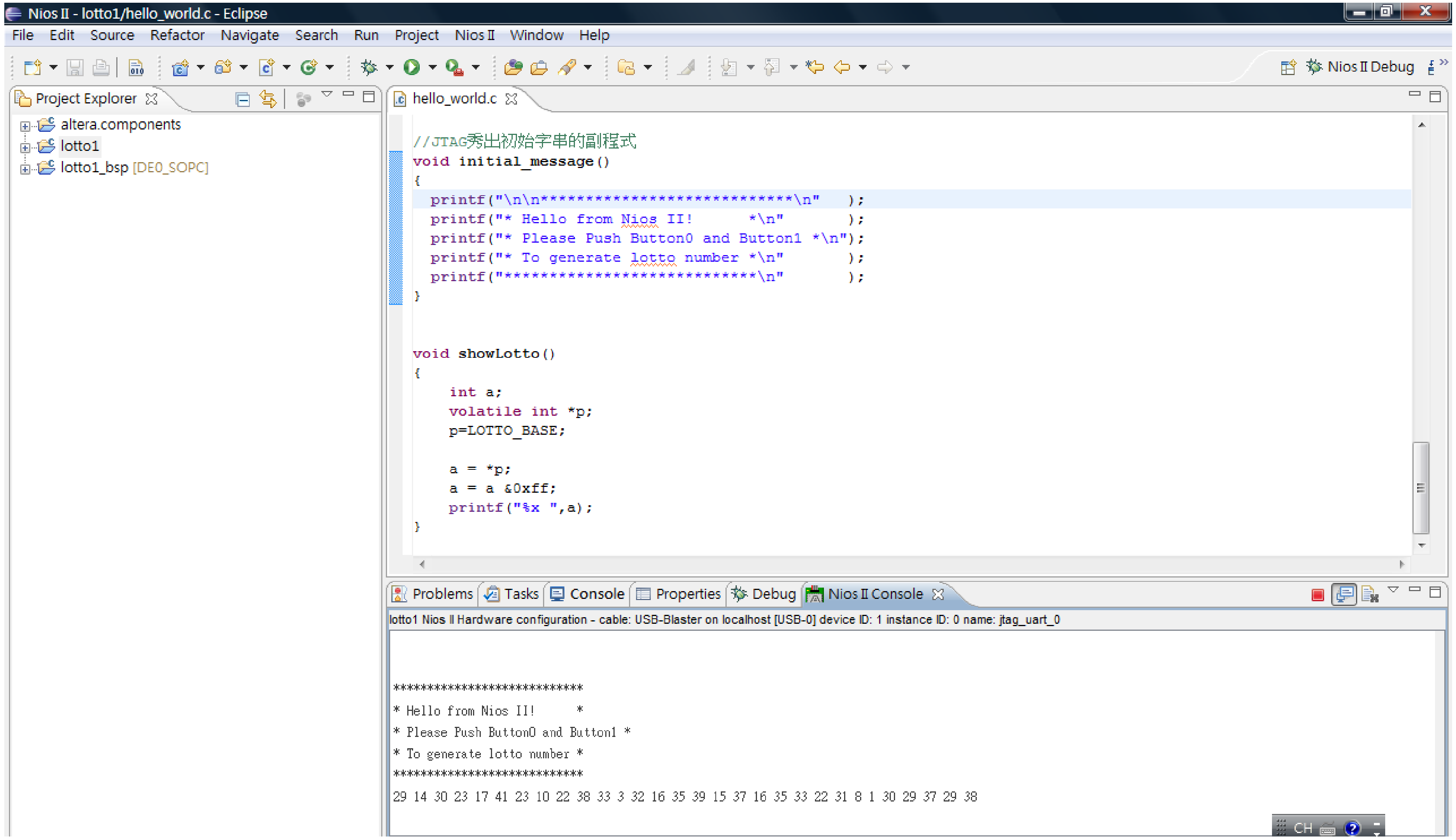
1-2 程式。

```
#include <stdio.h>
#include "unistd.h"
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"

/* 一個volatile型態的全域變數記錄著按鈕邊緣捕捉暫存器的值 */
volatile int edge_capture;

//按鈕相關的副程式
void handle_button_interrupts(void* context, alt_u32 id);
void init_button_pio();
void handle_button_press();
void initial_message();  //初始化的副程式

void showLotto();   //元件輸出的副程式


//主程式
int main(void)
{
  init_button_pio();
  initial_message();

  while( 1 )
  {
    usleep(100000);    //等待0.1秒

    if (edge_capture != 0)   //如果edge_capture不等於0
    {
      handle_button_press();  //執行按鈕按下的副程式
    }
   }
  return(0);
}
//按鈕中斷副程式，若按鈕被按下，則edge_capture值被改變
void handle_button_interrupts(void* context, alt_u32 id)
{
  volatile int* edge_capture_ptr = (volatile int*) context;
  *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0);
}

//註冊按鈕中斷副程式
```

```
void init_button_pio()
{
  void* edge_capture_ptr = (void*) &edge_capture;
  IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE, 0xf);  //將四個按鈕的中斷打開
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0x0);  //將記錄按鈕按下的暫存器歸0
  alt_irq_register( BUTTON_PIO_IRQ, edge_capture_ptr, handle_button_interrupts );  //註冊按鈕中斷副程式
}

//根據edge_capture的變數值來決定要執行的副程式
void handle_button_press()
{
    switch (edge_capture)  //針對edge_capture變數的內容，來決定要執行的副程式
    {
    case 0x1:    //按鈕1若按下
      showLotto();
      edge_capture=0;
    break;

    case 0x2:    //按鈕2若按下
      showLotto();
    break;

    case 0x4:    //按鈕3若按下
    break;

    case 0x8:   //按鈕4若按下
    break;
    }
}



//JTAG秀出初始字串的副程式
void initial_message()
{
  printf("\n\n****************************\n"   );
  printf("* Hello from Nios II!     *\n"      );
  printf("* Please Push Button0 and Button1 *\n");
  printf("* To generate lotto number *\n"      );
  printf("****************************\n"      );
}



void showLotto()
{
    int a;
    volatile int *p;
    p=LOTTO_BASE;

    a = *p;
    a = a &0xff;
    printf("%x ",a);
}
```

1-3 Run As/ Nios II Hareware




Nios II - lotto1/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help
Nios II Debug
Project Explorer
altera.components
lotto1
lotto1_bsp [DE0_SOPC]
hello_world.c
Problems Tasks Console Properties Debug Nios II Console

```c
//JTAG秀出初始字串的副程式
void initial_message()
{
  printf("\n\n****************************\n"    );
  printf("* Hello from Nios II!       *\n"       );
  printf("* Please Push Button0 and Button1 *\n");
  printf("* To generate lotto number *\n"        );
  printf("****************************\n"       );
}


void showLotto()
{
    int a;
    volatile int *p;
    p=LOTTO_BASE;

    a = *p;
    a = a &0xff;
    printf("%x ",a);
}
```

lotto1 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0

```
****************************
* Hello from Nios II!      *
* Please Push Button0 and Button1 *
* To generate lotto number *
****************************
29 14 30 23 17 41 23 10 22 38 33 3 32 16 35 39 15 37 16 35 33 22 31 8 1 30 29 37 29 38
```