Lab 6　DMA 實習

銘傳大學電腦與通訊工程學系

陳慶逸

一、背景知識

在實際應用時，若需要在兩個記憶體之間或者週邊設備與記憶體之間頻繁的進行資料存取操作，這些操作如果透過 CPU 來進行，勢必會耗費大量的 CPU 時間。透過分析這些操作會發現，整個過程並不需要任何的算術邏輯運算，完全可以排除 CPU 的干預。這在種情況下，則可以使用 DMA (Direct Memory Access)來完成，從而提高系統的工作效率。

Nios II DMA 模組用於實現兩個記憶體之間，或者記憶體和週邊裝置之間，或是兩個週邊裝置之間的直接資料傳輸。其週邊設備模組如下圖:

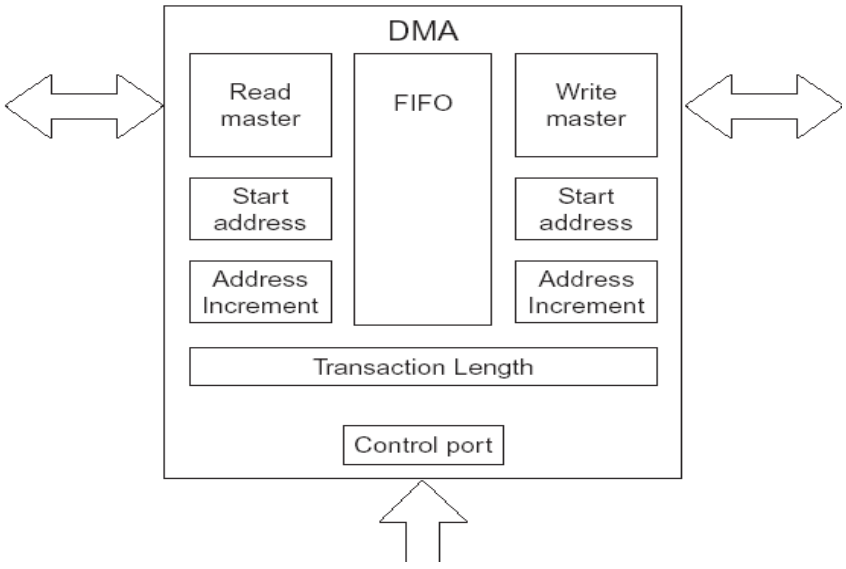
一個 DMA 控制器具有 3 個 Avalon 匯流排介面，分別為:

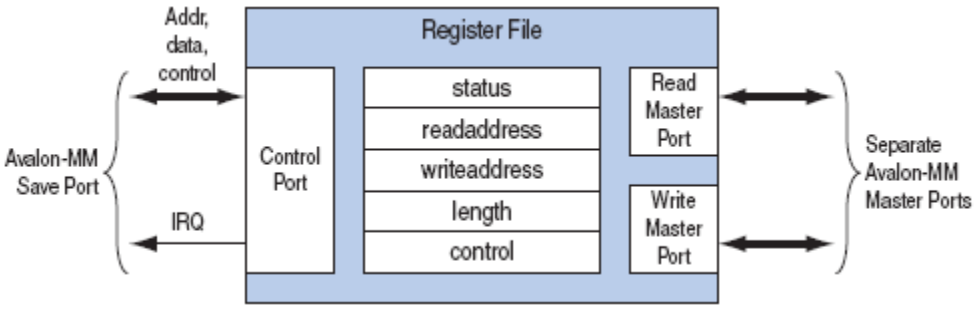
(1). 一個 Avalon 讀主埠 (Avalon Read Master Port)

(2). 一個 Avalon 寫主埠 (Avalon Write Master Port)

(3). 一個 Avalon 從埠(Avalon Slave Port)，讀寫均可。

兩個主埠都是用於 DMA 傳輸通道的，而從埠用於 DMA 控制相關暫存器的讀寫。





一個典型的 DMA 傳輸過程如下：

(1). CPU 透過寫控制埠(Control port)配置 DMA 控制器，準備啟動 DMA 傳輸。

(2). CPU 啟動 DMA 週邊設備進行 DMA 傳輸，DMA 週邊設備開始在沒有 CPU 干預的情況下傳送資料。

(3). DMA 讀埠從讀位址讀取資料，然後向 FIFO 依序寫入資料，DMA 寫埠從 FIFO 讀出資料，並向目標位址寫入資料。在傳輸過程中可以不需要 CPU 進行干預，但 CPU 可以終止當前的 DMA 傳輸過程。

(4). 當傳送資料達到指定長度或者遇到封包結束符號(EOP)，則 DMA 傳輸結束，DMA 週邊設備將產生一個 DMA 傳輸結束中斷。

DMA 暫存器定義:

Table 5. DMA Register Map

A2..A0	Register Name	R/W	Description/Register Bits											
			31	. . .	9	8	7	6	5	4	3	2	1	0
0	status[1]	RW								len	weop	reop	busy	done
1	readaddress	RW	Read master start address											
2	writeaddress	RW	Write master start address											
3	length	RW	Length in bytes											
4	reserved1	–	Reserved											
5	reserved2	–	Reserved											
6	control	RW			wcon	rcon	leen	ween	reen	i_en	go	word	hw	byte
7	reserved3	–	Reserved											

Note
(1) A write operation to the `status` register clears the `len`, `weop`, `reop`, and `done` bits.

(1) Readaddress: 源地址

(2) Writeaddress: 目的地址

(3) Length

(4) control

1-1 Status 暫存器

(1) done: 傳輸完成 flag

(2) busy: DMA 傳輸進行中，busy=1，否則為 0。

(3) reop: 如果 DMA 控制器接收到讀取資料端發出的封包結束事件，DMA 傳輸結束；reop=1。

(4) weop: 如果 DMA 控制器接收到寫資料端發出的封包結束事件，DMA 傳輸結束；weop=1。

(5) len: 傳送完指定長度的位元組後，DMA 傳輸結束，len=1。

1-2 readaddress & writeaddress

(1) readaddress: 讀主控端開始位址。

(2) writeaddress: 寫主控端開始位址。

1-3 Length

Length 的值表示從讀埠到寫埠所要傳輸的位元組數，在系統產生時確定。每當 DMA 的寫埠完成一次寫傳輸，length 暫存器的值便會減小，直到到達 0。

1-4 Control 暫存器

位元序	位元名稱	描述	
0	byte	8bit	
1	hw	16bit	
2	word	32bit	
3	go	DMA 致能	1 有效

4	i_en	中斷致能	1 有效
5	reen	讀端結束致能	
6	ween	寫端結束致能	
7	leen	指定長度結束致能	
8	rcon	從一個固定位址讀取資料	
9	wcon	向一個固定位址寫入資料	

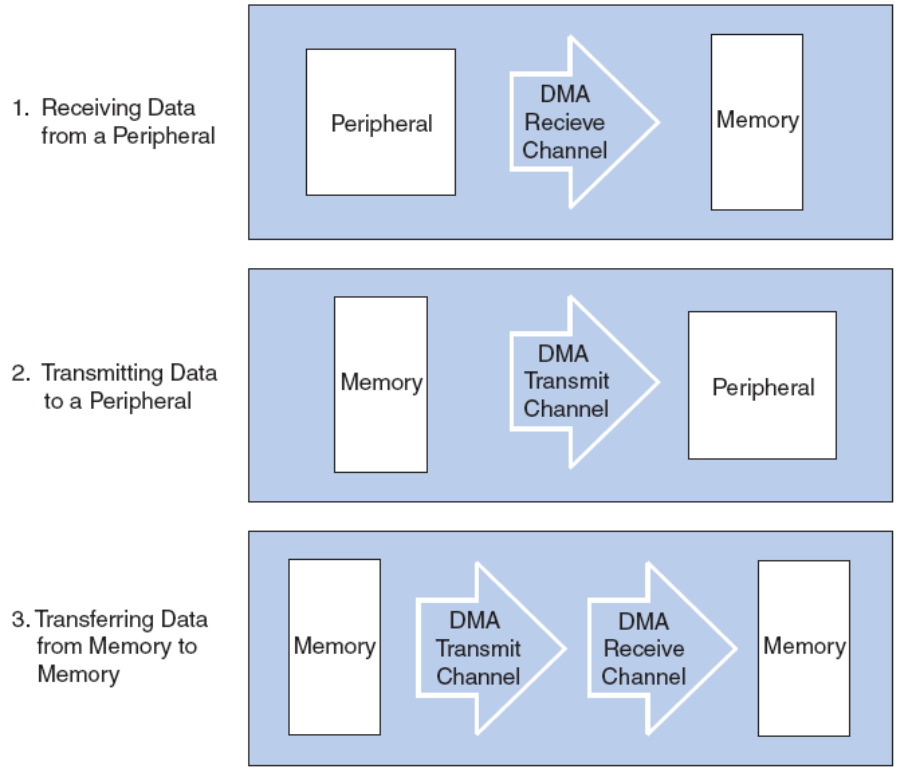
NIOS 提供 DMA 機制用來處理資料的一種協定。NIOS DMA 可以處理周邊設備的資料傳輸，它提供有：

(1) 介於兩記憶體之間的傳輸，

(2) 介於記憶體和周邊設備的傳輸，

(3) 介於兩周邊設備的傳輸。

且 NIOS DMA 常被用來連接具有 stream 能力的周邊設備，也就是說它可使資料傳遞是可變動或者是固定長度的資料流量。

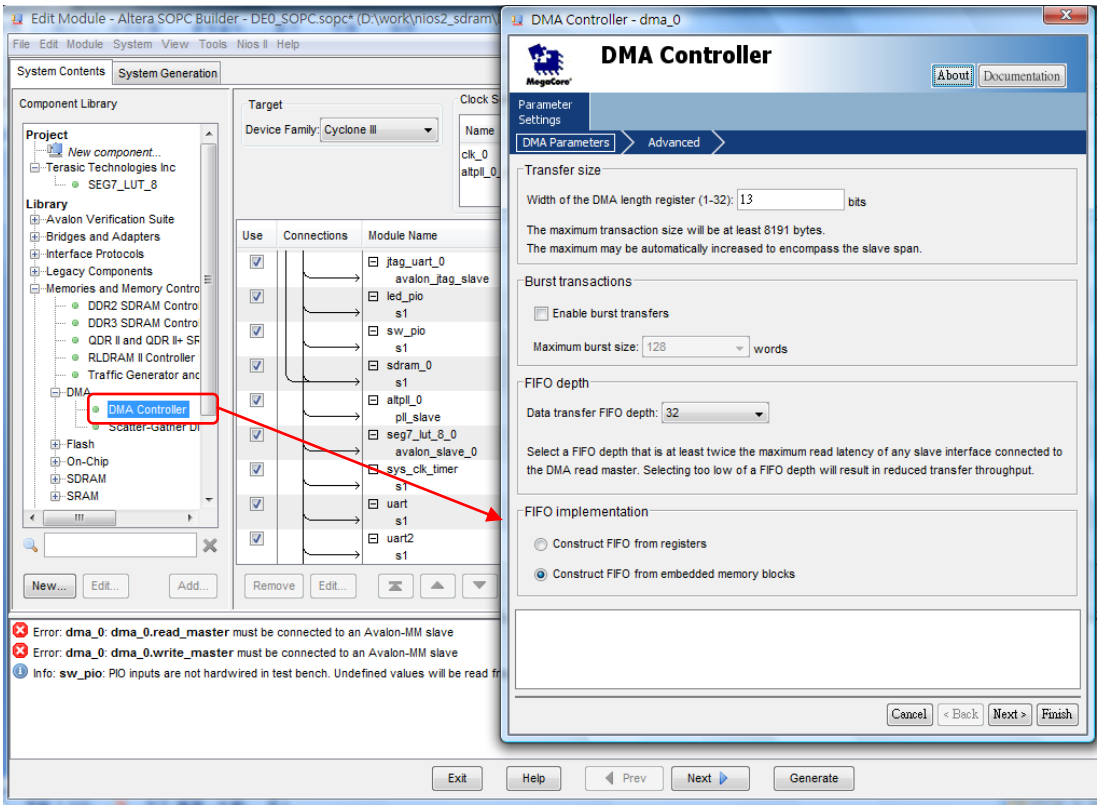


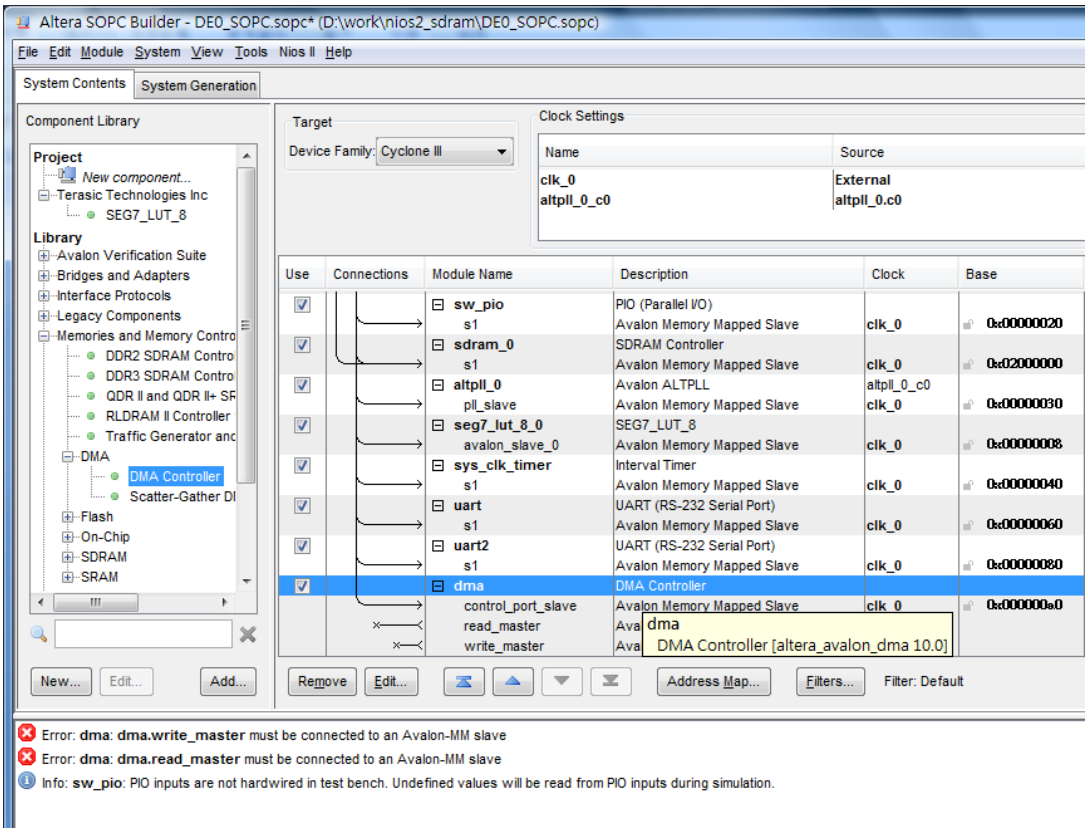
	rcon	wcon
Memory to Memory	0	0

Memory to Device	0	1
Device to Memory	1	0

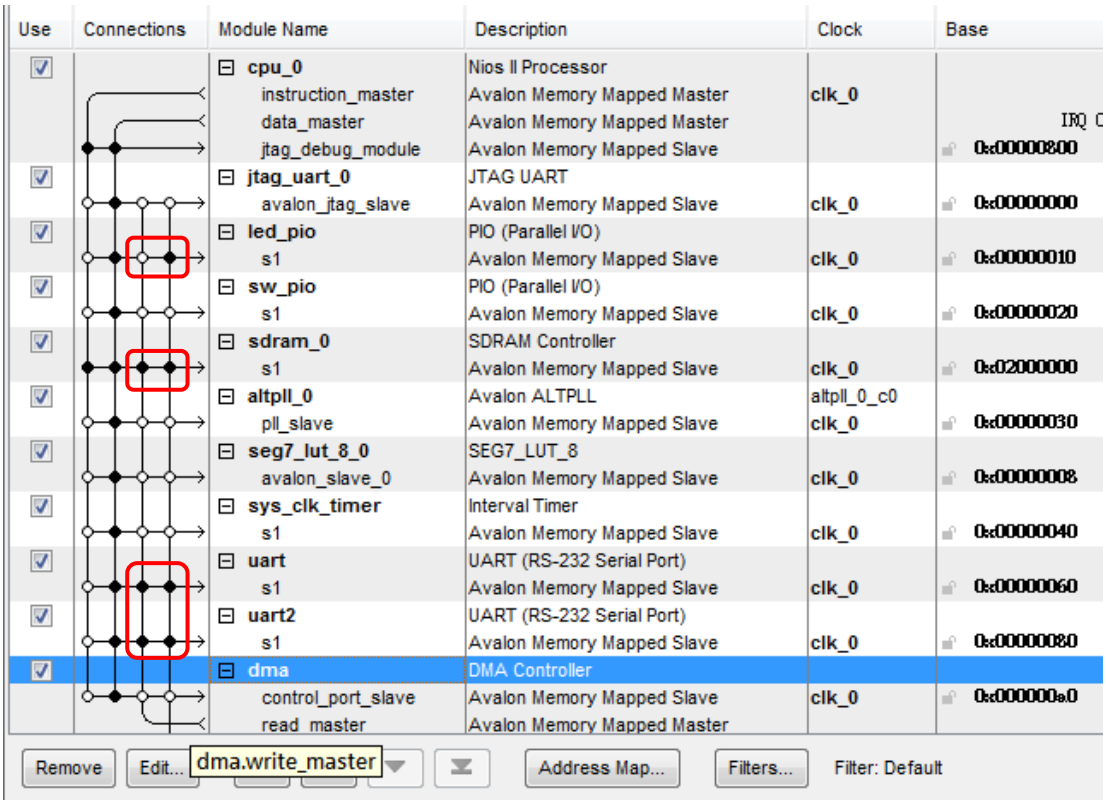
二、硬體設計

1. 在 SOPC Builder 中增加 DMA IP Core，重新命名為 dma。

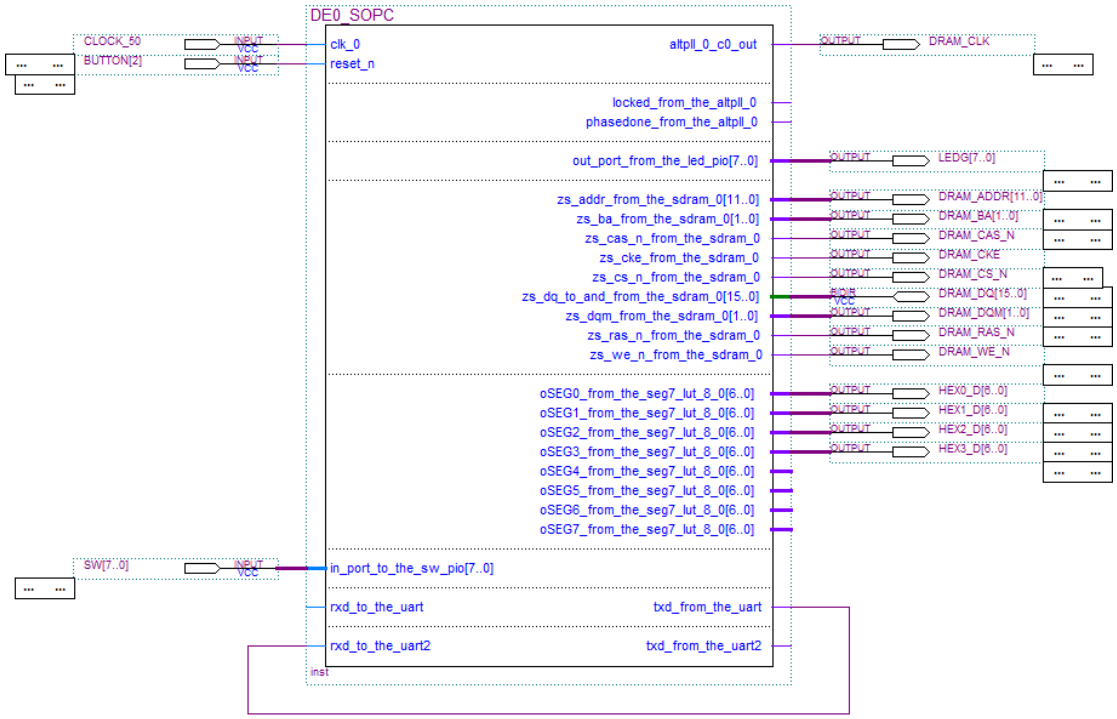




3. 連接 DMA 相關元件(led_pio, sdram_0, uart, uart2)。



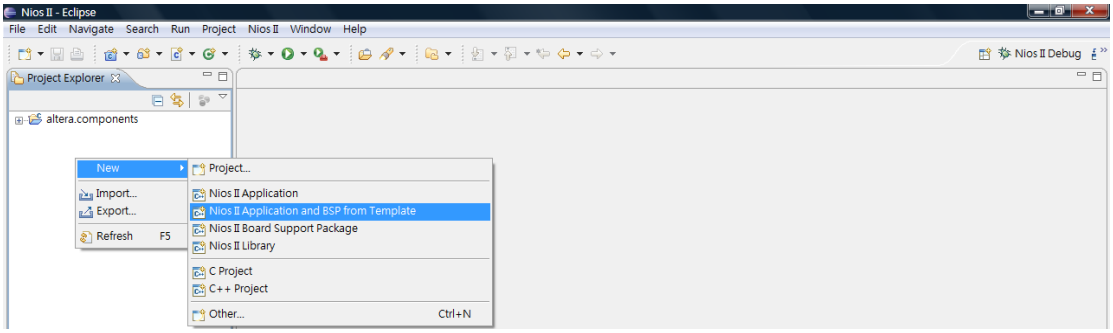
4.點選 Start Compilation 編譯電路。



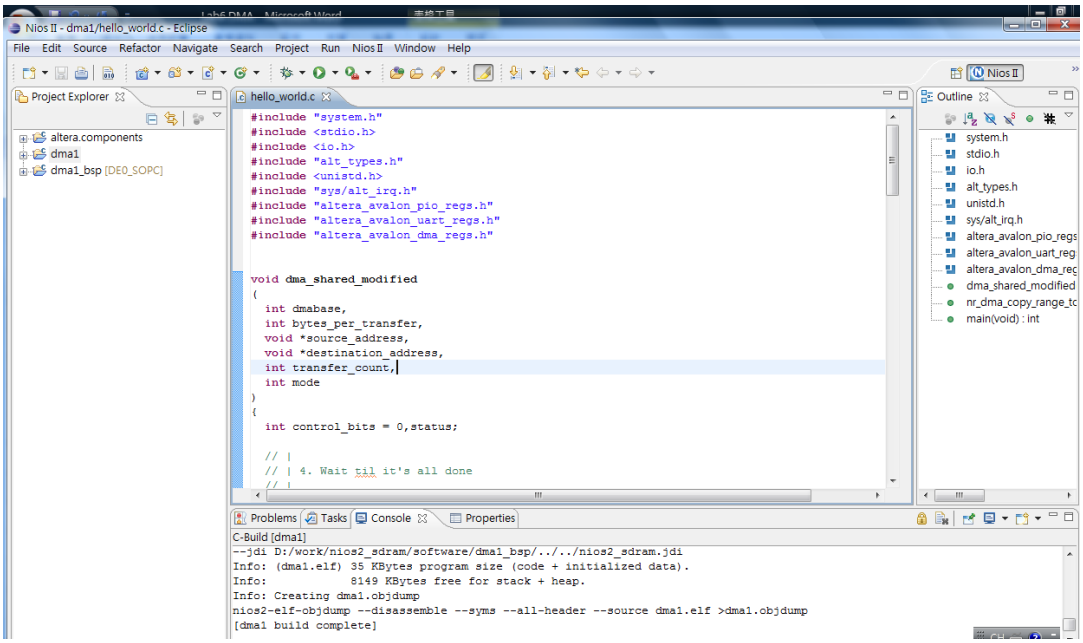
三、Nios II EDS 軟體設計

1. 在 Nios II DES 中建立新專案。

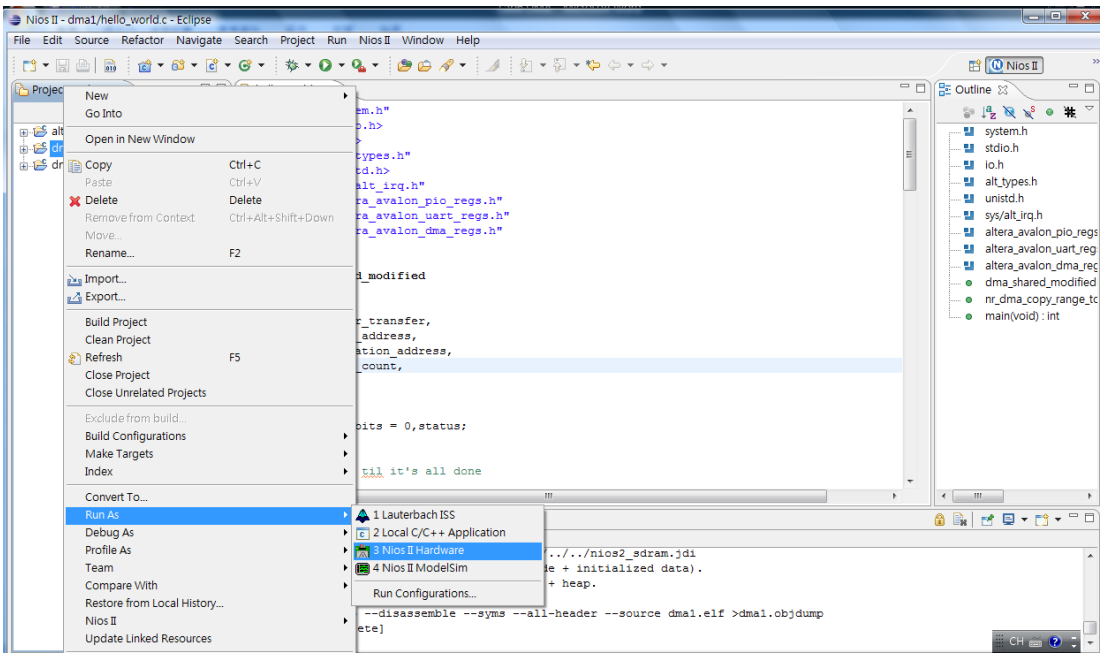
1-1 File/ New/ Nios II Application and BSP from Template



1-2 程式。



1-3 Run As/ Nios II Hareware


Nios II - dma1/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Project Run Nios II Window Help
Nios II
Outline
system.h
stdio.h
io.h
alt_types.h
unistd.h
sys/alt_irq.h
main(void) : int
New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Exclude from build...
Build Configurations
Make Targets
Index
Convert To...
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Nios II
Update Linked Resources
1 Lauterbach ISS
2 Local C/C++ Application
3 Nios II Hardware
4 Nios II ModelSim
Run Configurations...
--disassemble --syms --all-header --source dma1.elf >dma1.objdump

程式內容

```
#include "system.h"
#include <stdio.h>
#include <io.h>
#include "alt_types.h"
#include <unistd.h>
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_uart_regs.h"
#include "altera_avalon_dma_regs.h"



void dma_shared_modified
(
  int dmabase,
  int bytes_per_transfer,
  void *source_address,
  void *destination_address,
  int transfer_count,
  int mode
)
{
```

```
int control_bits = 0,status;

// |
// | 4. Wait til it's all done
// |
status=IORD_ALTERA_AVALON_DMA_STATUS(dmabase);
while((status & ALTERA_AVALON_DMA_STATUS_BUSY_MSK) != 0)
{
  status=IORD_ALTERA_AVALON_DMA_STATUS(dmabase);
  printf("status=%x\n",status);

}

// |
// | 1. Halt anything that's going on
// |

IOWR_ALTERA_AVALON_DMA_CONTROL(dmabase,0);

// |
// | 2. Set up everything but the go-bar
// |
```

```
  IOWR_ALTERA_AVALON_DMA_STATUS(dmabase,0);
  IOWR_ALTERA_AVALON_DMA_RADDRESS(dmabase, (int)source_address);
  IOWR_ALTERA_AVALON_DMA_WADDRESS(dmabase, (int)destination_address);
  IOWR_ALTERA_AVALON_DMA_LENGTH(dmabase, transfer_count * bytes_per_transfer);

  // |
  // | 3. construct the control word...
  // |

  control_bits =
      mode       // ocon and rcon bits
      | ALTERA_AVALON_DMA_CONTROL_BYTE_MSK    // low three bits of control reg
 //     | ALTERA_AVALON_DMA_CONTROL_I_EN_MSK // enable length (else runs forever)
      | ALTERA_AVALON_DMA_CONTROL_GO_MSK  // and... go!
      | ALTERA_AVALON_DMA_CONTROL_LEEN_MSK //傳輸完畢就停止
      | ALTERA_AVALON_DMA_CONTROL_RCON_MSK
      | ALTERA_AVALON_DMA_CONTROL_WCON_MSK;

  IOWR_ALTERA_AVALON_DMA_CONTROL(dmabase, control_bits);
  return;
  }

// +-----------------------------
```

```
void nr_dma_copy_range_to_range_modified
    (
    int dma,
    int bytes_per_transfer,
    void *first_source_address,
    void *destination_address,
    int transfer_count
    )
  {
  dma_shared_modified(dma,bytes_per_transfer,first_source_address,destination_address,transfer_count,
      0);
  }



int main(void)
{
    int i = 0;
    unsigned char status = 0;
    unsigned char txdata = 0;

    printf("start!!!  \n");
```

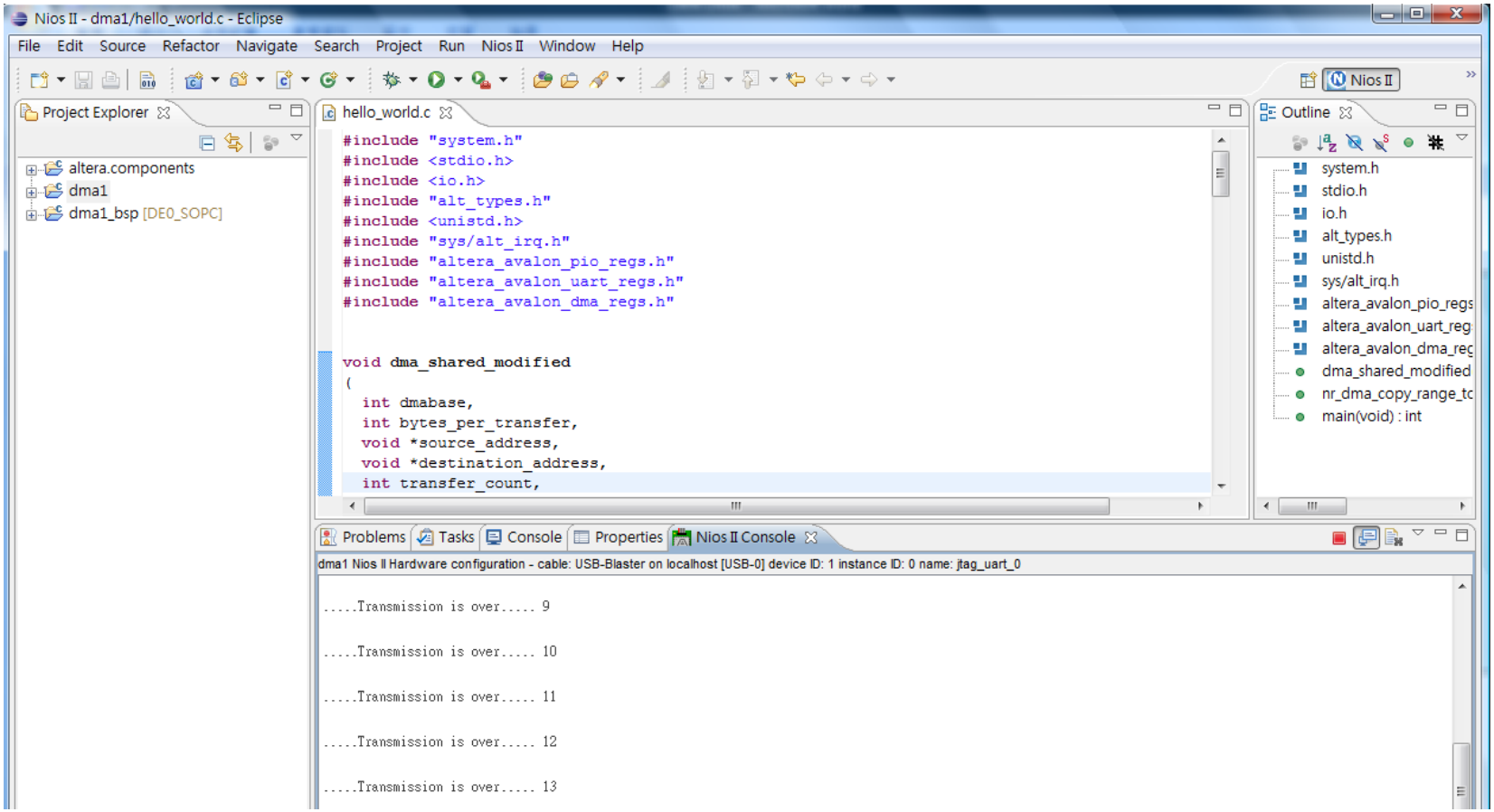
```
IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, txdata++);
usleep(200000);
status = IORD_ALTERA_AVALON_UART_STATUS(UART2_BASE);

while (1)
{
    if (status & 0x080)
    {
        nr_dma_copy_range_to_range_modified
        (
            DMA_BASE,
            1,
            (int *)UART2_BASE,
            (int *)LED_PIO_BASE,
            1
        );
        IOWR_ALTERA_AVALON_UART_STATUS(UART2_BASE,0);
        printf(".....Transmission is over..... %d\n\n", i++);

        IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, txdata++);
        usleep(200000);
        status = IORD_ALTERA_AVALON_UART_STATUS(UART2_BASE);
```

```
        } else {
            IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, txdata);
            usleep(200000);
            status = IORD_ALTERA_AVALON_UART_STATUS(UART2_BASE);
        }
    }

    return(0);
}
```


Nios II - dma1/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Project Run Nios II Window Help
Project Explorer
altera.components
dma1
dma1_bsp [DE0_SOPC]
hello_world.c
#include "system.h"
#include <stdio.h>
#include <io.h>
#include "alt_types.h"
#include <unistd.h>
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_uart_regs.h"
#include "altera_avalon_dma_regs.h"
void dma_shared_modified
(
int dmabase,
int bytes_per_transfer,
void *source_address,
void *destination_address,
int transfer_count,
Outline
system.h
stdio.h
io.h
alt_types.h
unistd.h
sys/alt_irq.h
altera_avalon_pio_regs
altera_avalon_uart_reg
altera_avalon_dma_reg
dma_shared_modified
nr_dma_copy_range_to
main(void) : int
Problems Tasks Console Properties Nios II Console
dma1 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0
.....Transmission is over..... 9
.....Transmission is over..... 10
.....Transmission is over..... 11
.....Transmission is over..... 12
.....Transmission is over..... 13

四、作業

作業一

題目： DMA 控制 LED

簡介：

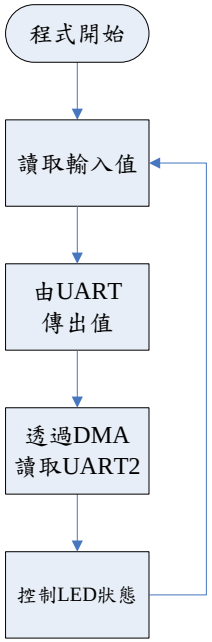
由鍵盤輸入 8 位元的數值，用 UART 送出數值，由 DMA 來收 UART 的接收暫存器，並傳送到 LED 燈顯示。

使用元件

1、 DMA => 搬移數值資料

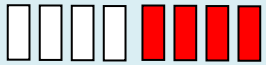
2、 UART => 接收以及傳送數值

3、 LED PIO => 輸出數值代表的亮滅信號



執行結果:

```
Please input your number : ( 0 -> 255 )
00001111
```



```
Please input your number : ( 0 -> 255 )
11111111
```

```
Please input your number : ( 0 -> 255 )
```