Computer Architecture and System Laboratory

處理器設計與實作

實習講義

編撰者
成大電機所計算機架構與系統研究室CASLAB

國立成功大學電機系與電機研究所

Computer Architecture and System Laboratory

LAB 8: Simple CPU System Platform & Memory Allocation

實驗目的

1. AMBA2.0—AHB 的基本認識
2. 簡介EASY實驗平台環境
3. 了解Address Decoding (MMIO)
4. 練習程式讓CPU去存取Bus上的Slave的動作
5. 學習將平台透過Vivado合成，並在FPGA板驗證GPIO

Background

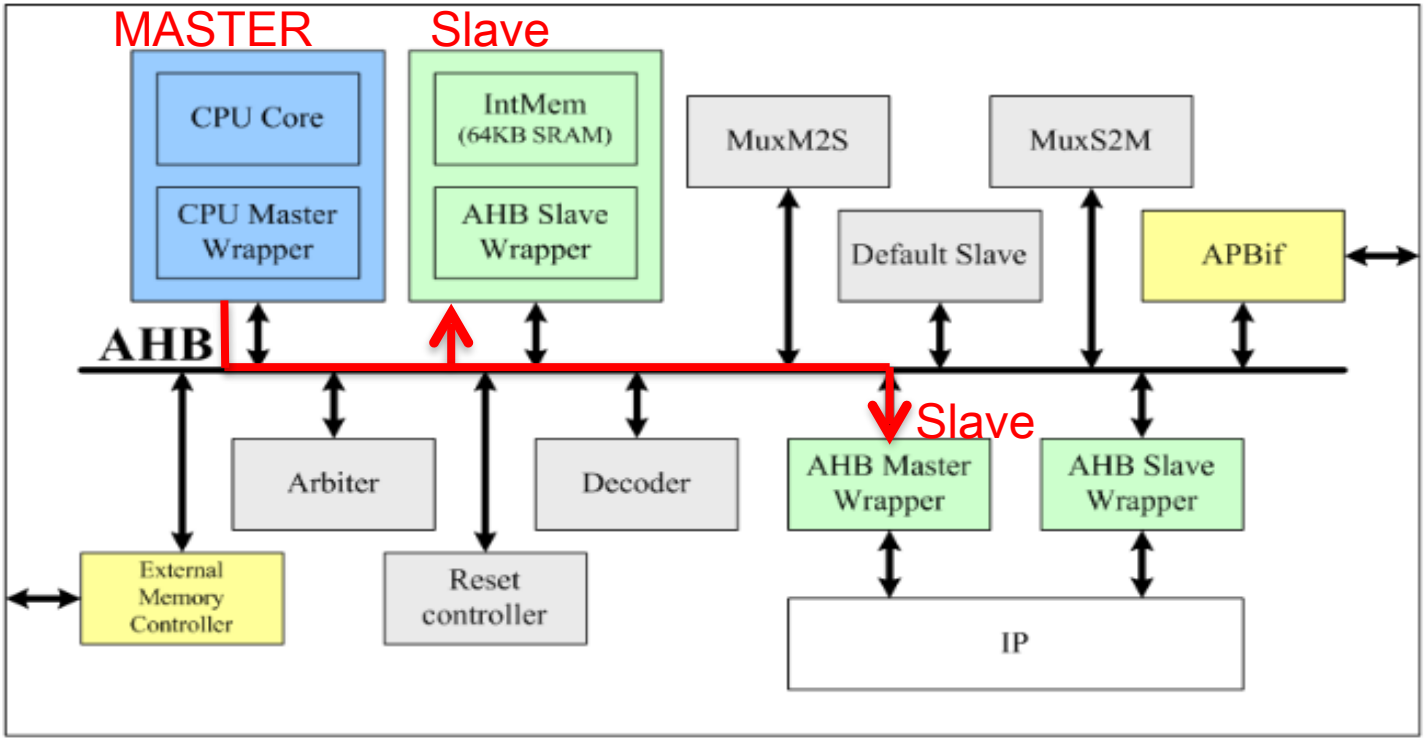
- System on Chip (SoC)
  - Processing Elements
    - ➢Microcontroller , Microprocessor , DSP ..
  - Memory Elements
    - ➢ROM , RAM , Flash…
  - I/O & Peripherals
    - ➢Timer , Watchdog , UART , GPIO…
  - Interconnection
    - ➢**AMBA** , CoreConnect , NoC …

AMBA2.0--AHB

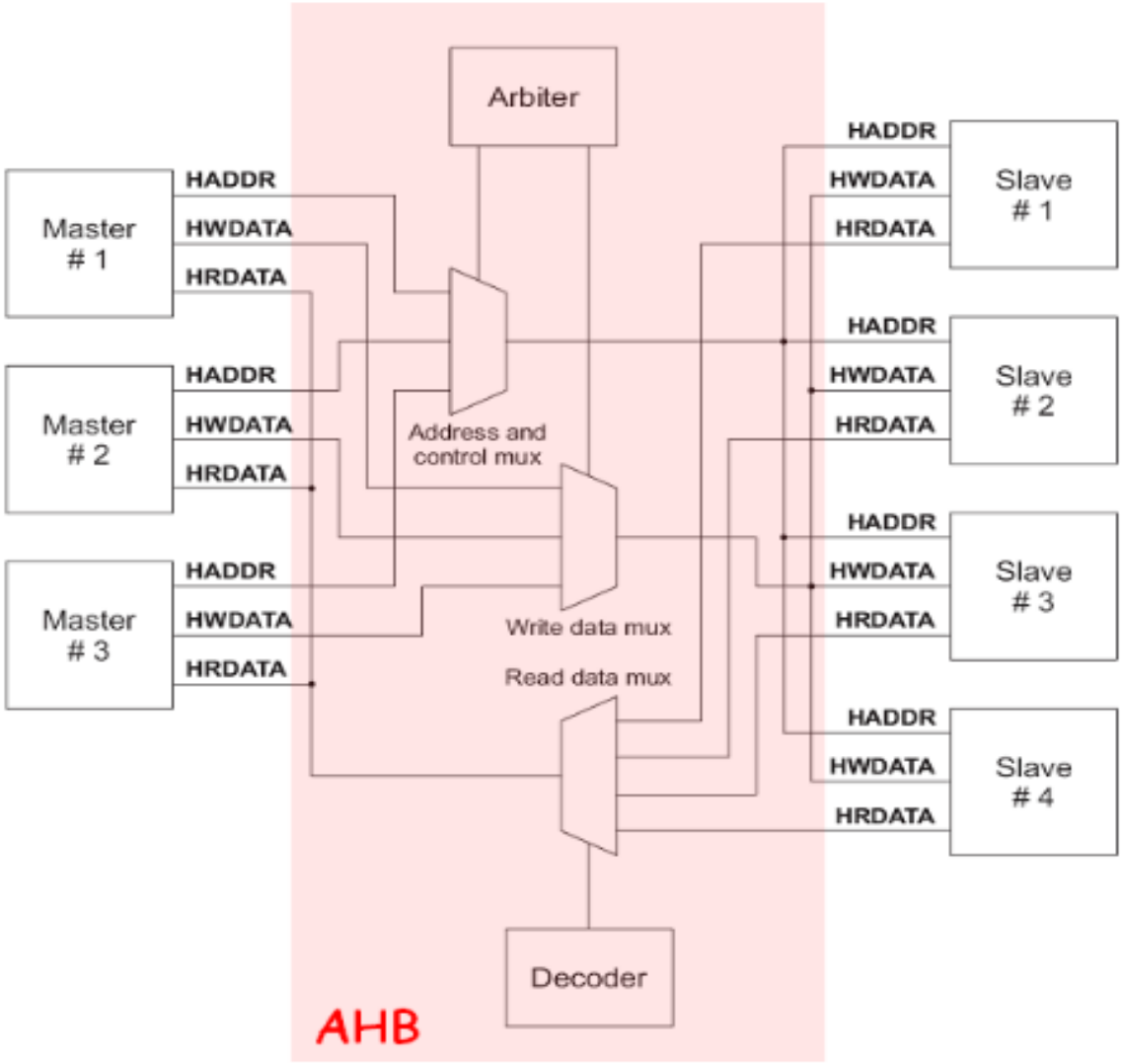
- Advanced Micro-controller Bus Architecture(AMBA)
  - ARM公司所推出的 on-chip bus 協定
- Advanced High-Performance Bus (AHB)
  - High performance
  - single-clock edge operation
  - Pipeline operation
  - multiple bus master

AHB的基本知識

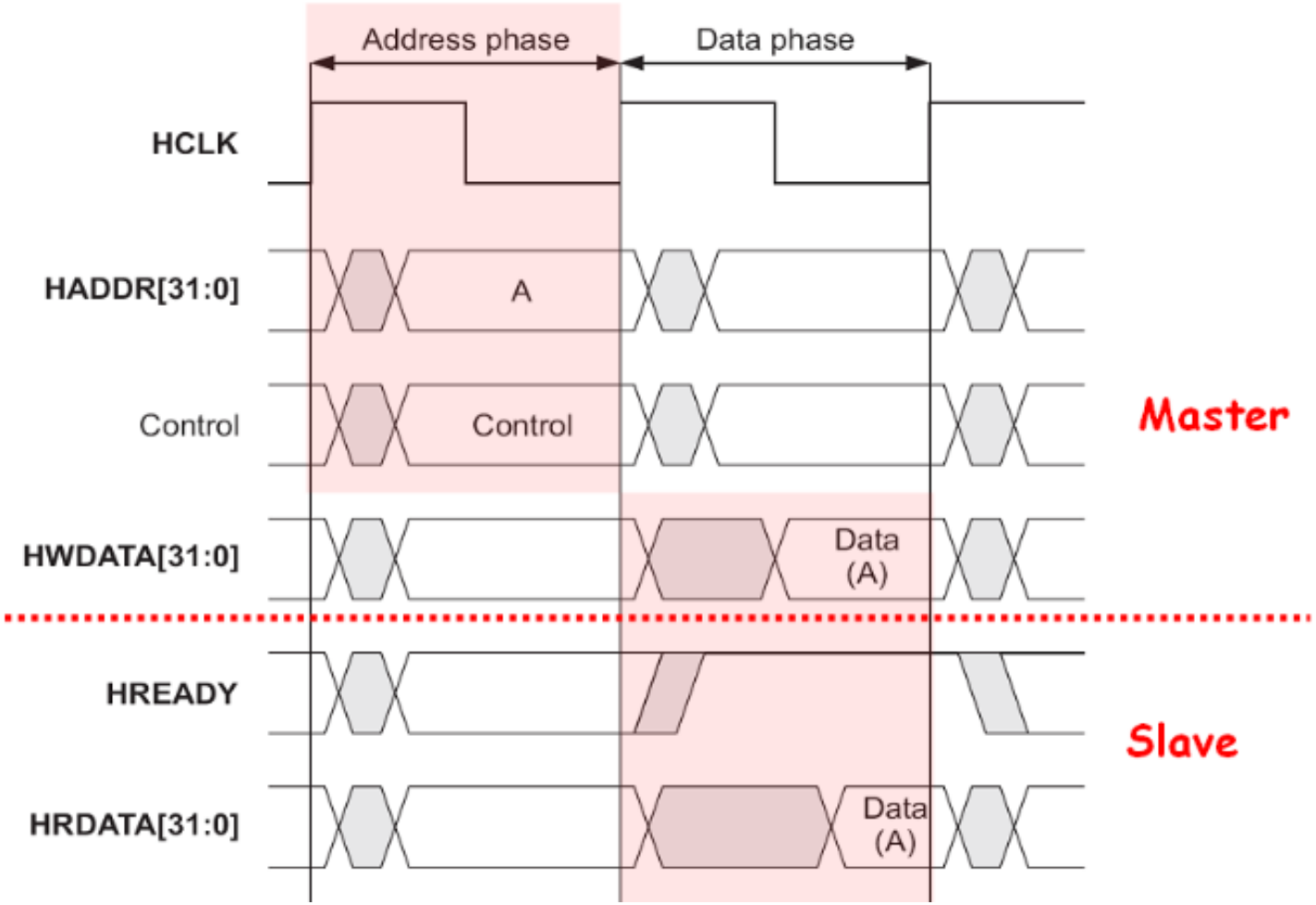
- AHB System 是由 Master、Slave、Infrastructure三部分所組成
  - AHB bus上的傳輸都是由 Master 所發出；由Slave負責回應
  - infrastructure 則由 Arbiter 、 Master to Slave multiplexor、 Slave to Master multiplexor、Decoder、Default slave、Reset Controller 所組



AHB Bus Controller



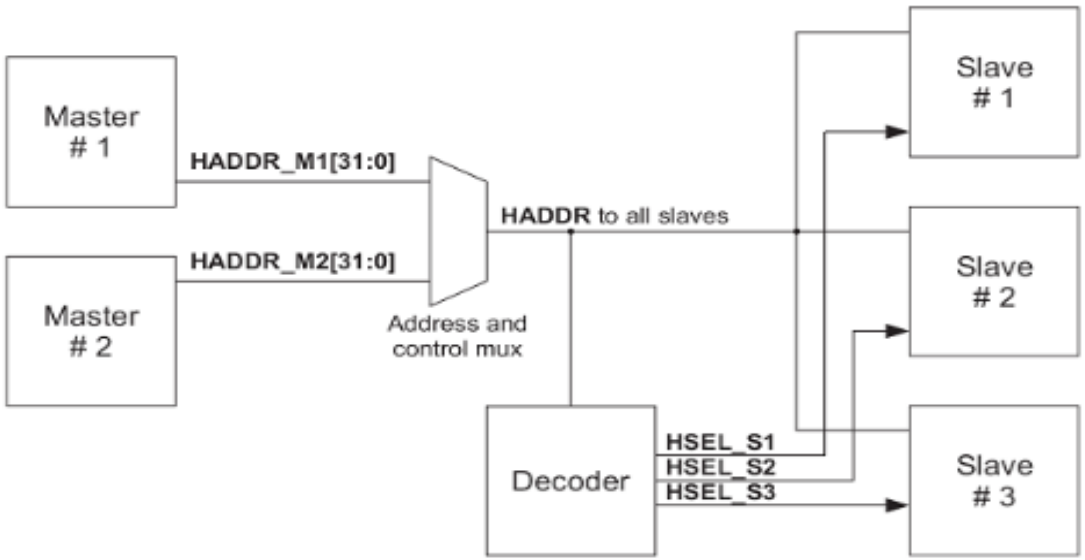
AHB Basic transfer

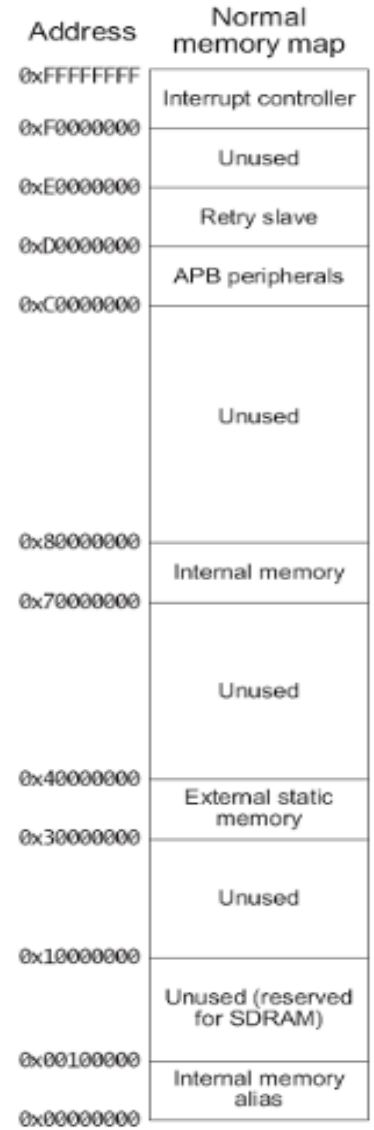


MMIO(Memory-mapped I/O)

- 系統中所有的實體記憶體或是I/O Device都會mapping到32 bits (4G)的位置空間中
- CPU的指令不只可以存取memory也可以存取Devices
- 系統上的Devices會監視bus address，若發現CPU存取的位置是該Device時會回應CPU的存取，並允許bus上的data存取該Device中的實體register

Address Decoder

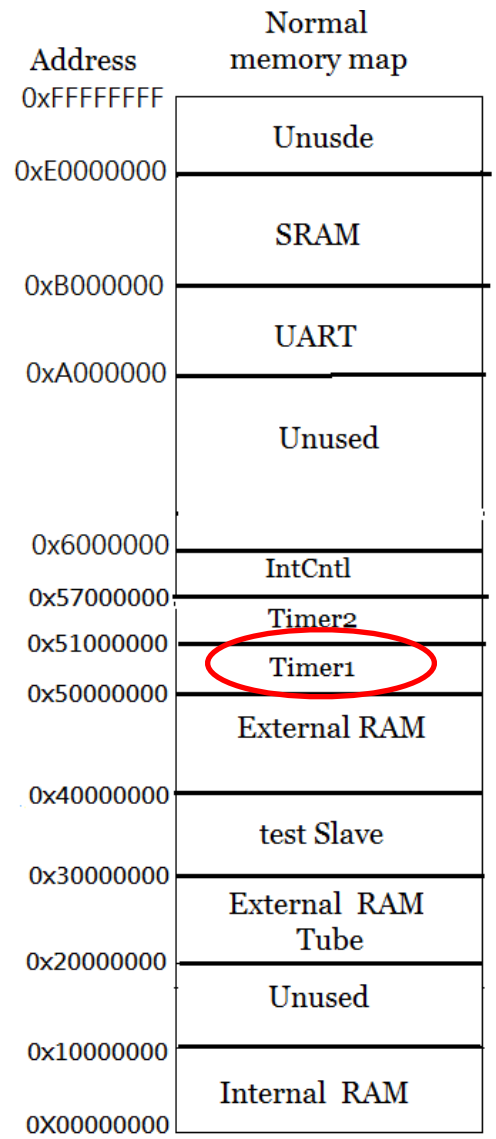




- Address Space
- HSELx
- HREADY
- 1KB boundary
- Default slave

Decoder 範例

```
case (HADDR[31:27])
  5'b00000 :
     HSELIntMem = 1'b1;

  5'b00100 ,
  5'b01000 :
          HSELExtMem = 1'b1;     //  External Memory + Tube
  5'b00110 :
      HSELAPBif = 1'b1;    //    test Slave
  5'b01010 :
    begin
      if(HADDR[26:24]==3'b000)
          HSELTimer1 = 1'b1;    //    Timer1

      else if(HADDR[26:24]==3'b001)
          HSELTimer2 = 1'b1;    //    Timer2

      else if(HADDR[26:24]==3'b111)
          HSELIntCntl = 1'b1;    //    IntCntl
    end
  default  :
    HSELDefault = 1'b1;    //  Undefined (Default Slave)

endcase
```



初始 Instruction Memory

- CPU(Master)會透過AHB讀取內部記憶體(IntMEM)中的程式執行

- IntMem.v

```
initial
begin
  $display("### Loading internal memory, Based Addr = 0x%x, Length = 0x%x ###",BaseAddr,MemSize*1024);
  for (i=0; i<=((MemSize * 256)-1); i = i+1)
    Mem[i] = 32'h0000_0000;
  if (FileName != "")
  begin
    $readmemh(FileName, Mem);
  end
  $display("### Load internal memory Complete ###");
end
```

- TBTic.v

```
//-----------------------------------------------------------------
// Re-define Parameter
//-----------------------------------------------------------------
// Memory size in Kbytes, set to 64 MB
defparam uEASY.uIntMem_1.MemSize  = 1024*64;

// Input filename
defparam uEASY.uIntMem_1.FileName = "./testcode/testcode.txt";
```

TBTic.v 為EASY平台的testbench ，其中定義Memory Size和initial 的machine code 路徑

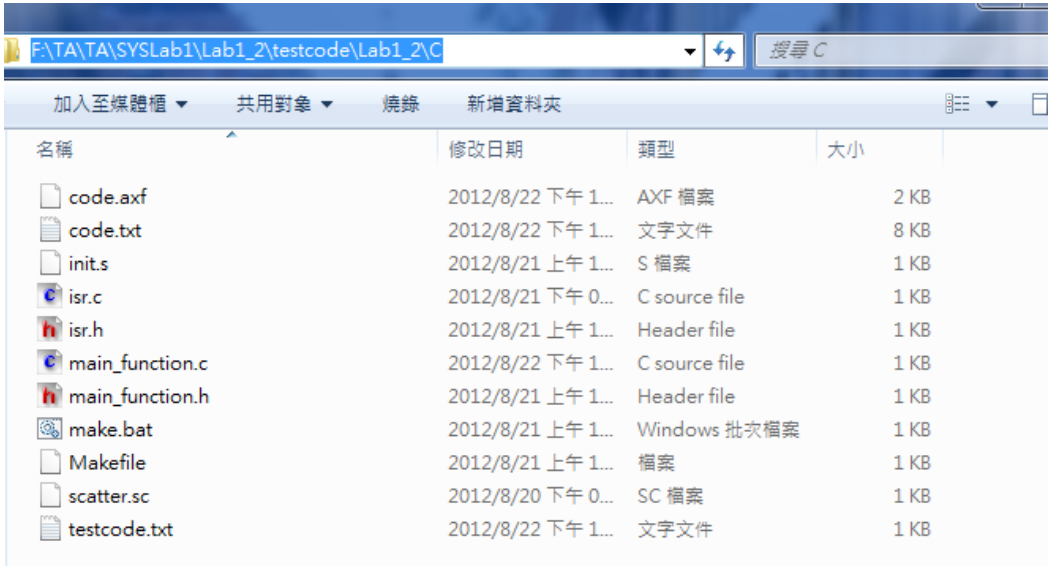
Volatile的用法

- 於Decoder的mapping可知，Timer1掛在系統位置0x50000000
- 在C code 中用 **volatile** 直接宣告變數Timer1等於在系統0x50000000的位置
  - Timer1[0] 等於 0x50000000的位置
  - Timer1[1] 等於 0x50000004的位置
  - 依此類推即可存取系統中其他的Slave, 如External Memory, GPIO…

```
int main ()
{

 volatile int *Timer1 =  (int*) 0x50000000;

  Timer1[0] = 1 ;
  Timer1[1] = 100 ;
 return 1 ;
}
```

ARM Cross Compiler

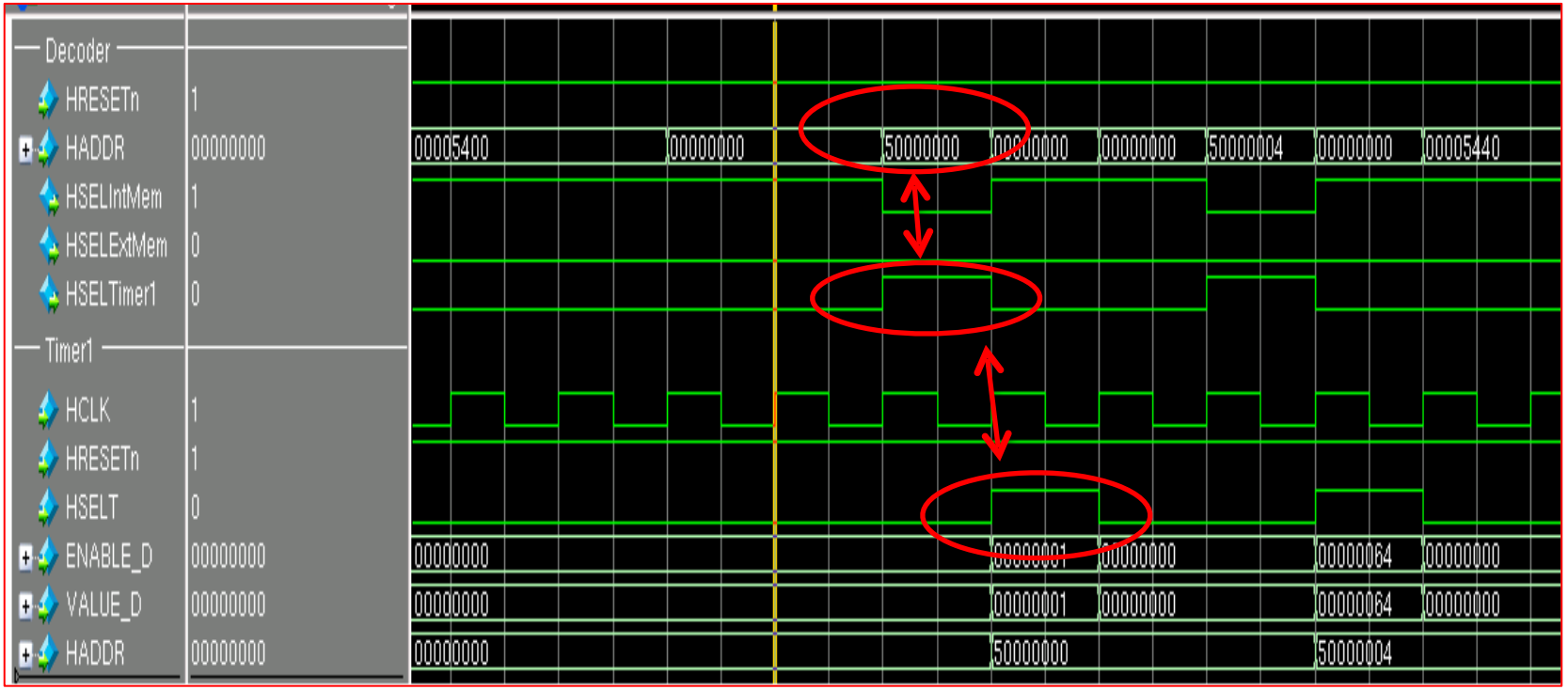
- 在每個子Lab中的testcode資料夾->子資料夾->C資料夾中，有個make.bat檔將init.s、isr.c、main_function.c三個檔案透過ARM Cross Compiler link再一起並編譯成ARM的machine code




```
armasm init.s --cpu=4 -o init.o --unsafe
armcc main_function.c -O2 --cpu=4 -c -o main_function.o
armcc isr.c -O2 --cpu=4 -c -o isr.o
armlink init.o main_function.o isr.o -o code.axf --scatter scatter.sc
fromelf -c -text -o code.txt code.axf
fromELF code.axf -o testcode.txt -vhx -32x1
del *.o
pause
```

ModelSim Simulation

- 用Modelsim模擬EASY運作情形
  - 當Master要存取Timer1的變數時只要讓HADDR = 0x50000000，Decoder就會把HSELTimer1拉起來，下個cycle Timer1的HSELT也被拉起



Vivado 合成電路

lab8 - [D:/gitdev/colab/test/lab8/lab8_takeaway0/lab8.xpr] - Vivado 2014.4

File Edit Flow Tools Window Layout View Help

Synthesis and Implementation Out-of-date more info

Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
  - Package IP
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Open Elaborated Design
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Design
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Design
- Program and Debug
  - Bitstream Settings
  - Generate Bitstream
  - Open Hardware Manager

Project Manager - lab8

Sources

- Design Sources (97)
  - Non-module Files (92)
  - EASY (EASY.v) (8)
  - blk_mem_gen_1 (blk_mem_gen_1.xci)
    - risc32 (risc32.edif)
  - IP-XACT (1)
  - Coefficient Files (1)
- Constraints (1)
- Simulation Sources (95)

Hierarchy | IP Sources | Libraries | Compile Order

Sources | Templates

Properties

Project Summary

Active run: synth_1
Part: xc7a100tcsg324-1
Strategy: Vivado Synthesis Defaults

Active run: impl_1
Part: xc7a100tcsg324-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Summary | Route Status

DRC Violations

Summary:
- 0 errors
- 0 critical warnings
- 11 warnings
- 0 advisories

Timing - Post-Implementation

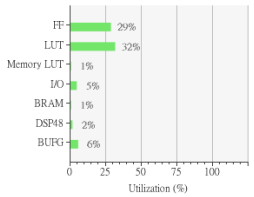
Worst Negative Slack (WNS): 1.595 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 112299

Implemented Timing Report

Setup | Hold | Pulse Width

Post-Synthesis | Post-Implementation

Utilization - Post-Implementation



Graph | Table

Post-Synthesis | Post-Implementation

Power

Total On-Chip Power: 0.117 W
Junction Temperature: 25.5 ℃
Thermal Margin: 59.5 ℃ (12.9 W)
Effective ΘJA: 4.6 ℃/W
Power supplied to off-chip devices: 0 W
Confidence level: Medium

Summary | On-Chip

Design Runs

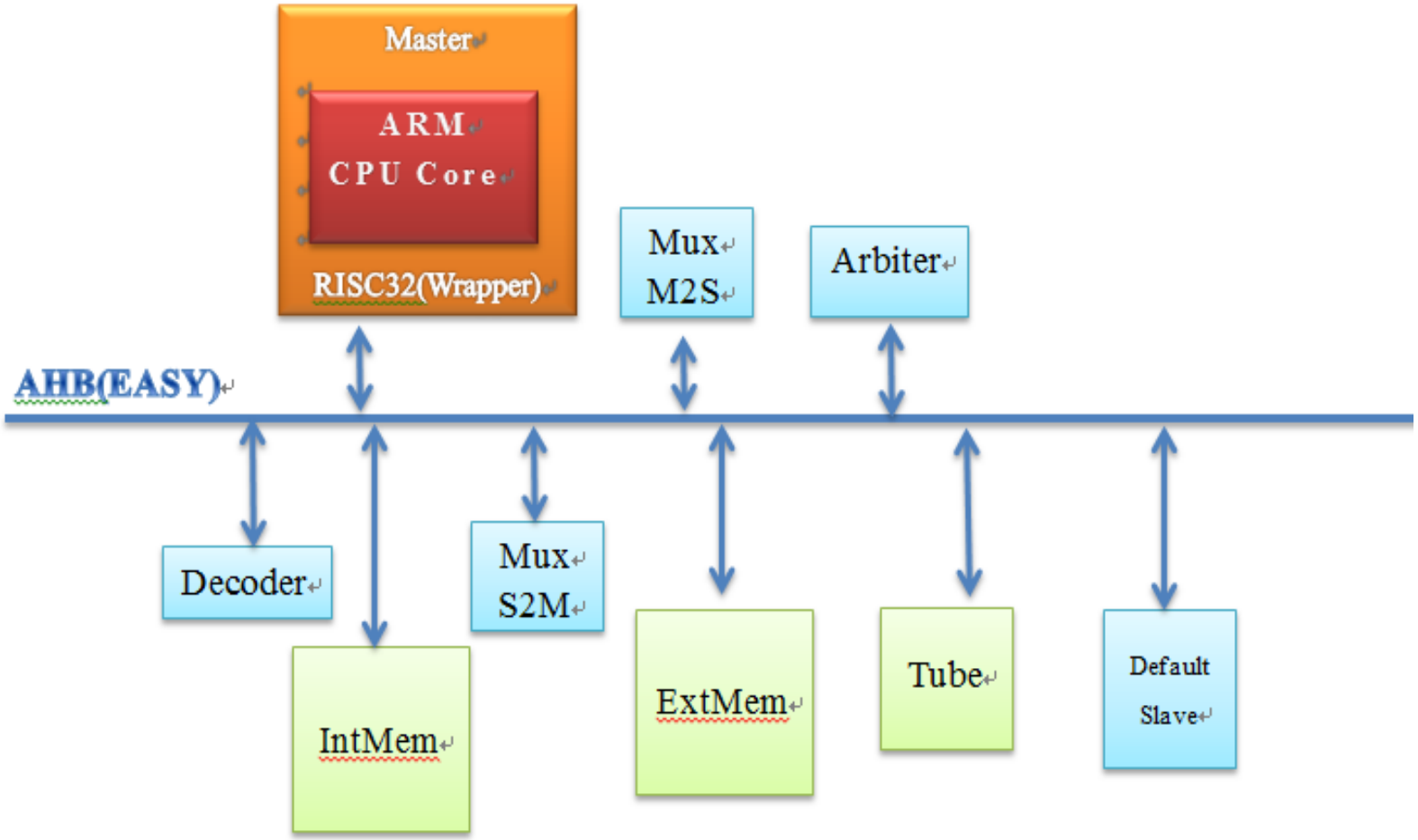
Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP	Start	Elapsed	Status	Progress	Strategy
synth_1 (active)	constrs_1							32.84	29....	0.00	1.67	3/29/16 12:23 PM	00:00:34	Synthesis Out-of-date	100%	Vivado Synthesis Defaults (Vivado Synthesis 2014)
impl_1 (active)	constrs_1	1.60	0.00	0.04	0.00	0.00	0	32.18	29....	0.37	1.67	3/29/16 12:24 PM	00:05:12	Implementation Out-of-date	100%	Vivado Implementation Defaults (Vivado Implementi
Out-of-Context Module Runs																
blk_mem_gen_1_synth_1	blk_mem_gen_1							0.11	0.01	23.70	0.00	3/11/16 2:06 PM	00:02:12	Synthesis Out-of-date	100%	Vivado Synthesis Defaults (Vivado Synthesis 2014)
blk_mem_gen_0_synth_1	blk_mem_gen_0							0.00	0.00	0.37	0.00	3/11/16 2:06 PM	00:01:37	synth_design Complete!	100%	Vivado Synthesis Defaults (Vivado Synthesis 2014)

Tcl Console | Messages | Log | Reports | Design Runs

實驗系統架構

Example AMBA SYstem (EASY)

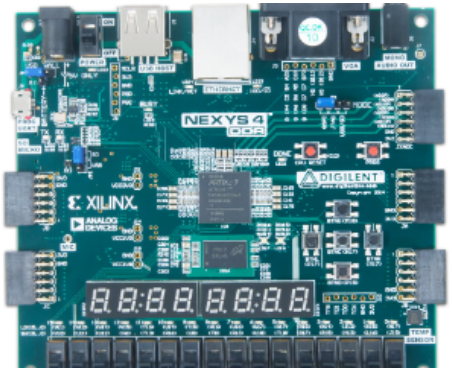
EASY平台 (ARM processor)



實驗環境

Tool used：

1. ARM GNU Toolchain
   - 利用GNU的ARM Cross Compiler將C轉成arm assembly和machine code
2. Mentor Modelsim
   - 看平台上的wave來驗證CPU和AHB的行為
3. Vivado 2016.4
   - 將驗證過後的平台透過Vivado合成成bit檔
4. Nexys 4 DDR board
   - 利用此驗證版將平台燒錄至FPGA中並使用板子上的I/O驗證行為



實作(一)

EASY Simulation

實作(一)

- 請同學練習利用ModelSim 模擬EASY 運作情形
  1. 打開Decoder.v可以看到如右圖之程式
  2. 可以看到Tube掛在0x20000000的位置
  3. Tube是一個可以在模擬系統中印出字體的Slave

```
  always @(HRESETn or HADDR )
begin

  HSELDefault = 1'b0 ;
  HSEL_Slave1 = 1'b0 ;
  HSEL_Slave2 = 1'b0 ;
  HSEL_Slave3 = 1'b0 ;


  if (!HRESETn)
    HSELDefault = 1'b1;      //  Reset (Default Slave)
  else
  begin
       case (HADDR[31:27])
      5'b00000 :
          HSEL_Slave1 = 1'b1;

      5'b00100 :
          HSEL_Slave2 = 1'b1;     //  Tube
      5'b00110 :
          HSEL_Slave3 = 1'b1;     //    testSlave


      default  :
        HSELDefault = 1'b1;   //  Undefined (Default Slave)


    endcase
  end
end
```

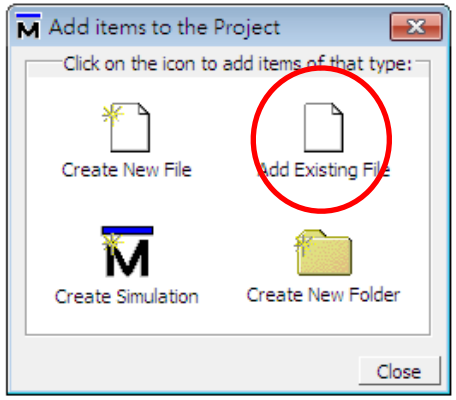
實作(一)

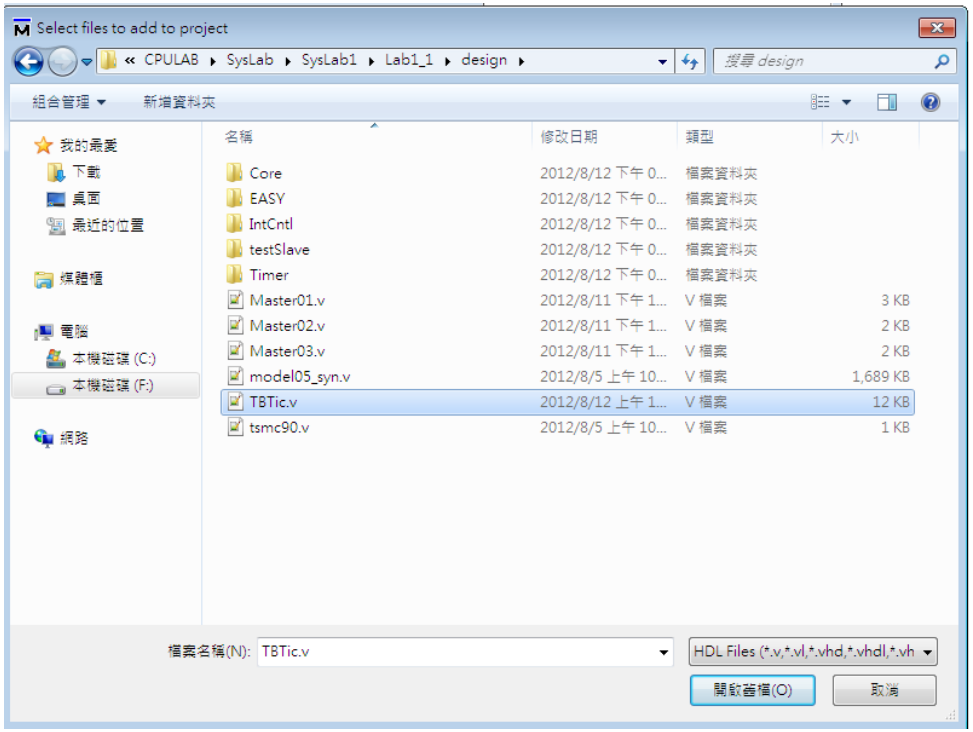
4. 寫C code (testcode\C\main_function.c)
   - 用Volatile 宣告變數*Tube 等於位置0x20000000的地址所放的值
   - 並用右圖寫法依序對Tube[0]寫入所要的字元
   - 儲存main_function.c檔並透過Compiler轉成testcode.txt放到testcode資料夾下

```
#include "main_function.h"

void main_function ()
{
 volatile int *tube =  (int*) 0x20000000;

  tube[0] = '\n' ;
  tube[0] = 'H' ;
  tube[0] = 'e' ;
  tube[0] = 'l' ;
  tube[0] = 'l' ;
  tube[0] = 'o' ;
  tube[0] = '\t' ;
  tube[0] = 'E' ;
  tube[0] = 'A' ;
  tube[0] = 'S' ;
  tube[0] = 'Y' ;
  tube[0] = '!' ;
  tube[0] = '\n' ;

}
```

實作(一)

5. 在Lab8-1資料夾下開啟ModelSim專案(test.mpf)後，匯入design資料夾下的TBTic.v(開啟test.mpf後會出現一個TBTic.v，要先將它移除然後右鍵add重新加進來)



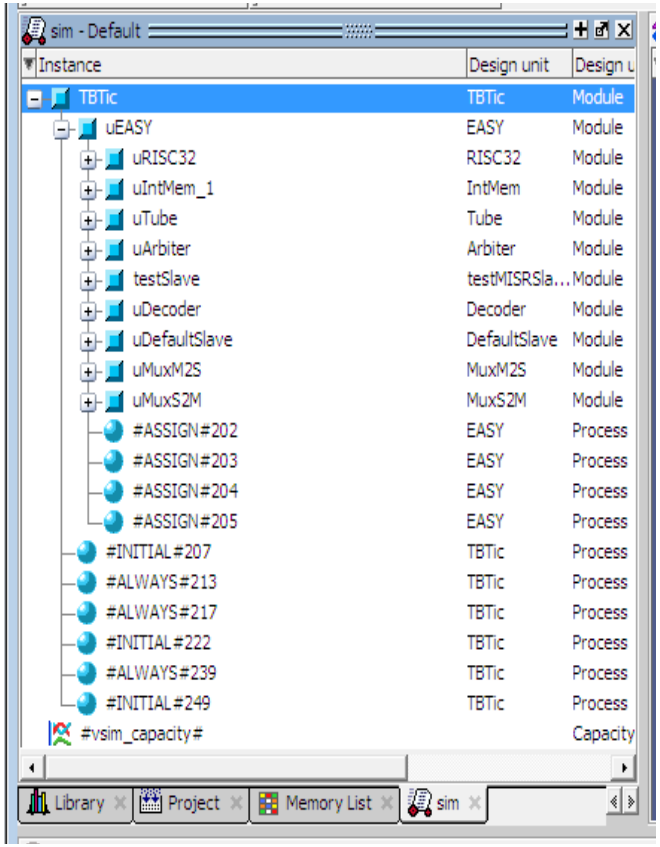


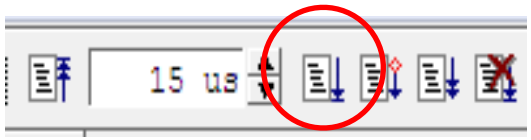
實作(一)

6. Compile完後按Simulate(輸入TBTic)，可以在sim視窗看到EASY平台的Hierarchy

7. Veiw->Wave 跳出視窗後File -> Load 選lab8_1.do(跳ERROR不影響)

8. 設定模擬時間為15 us後按run






```
VSIM 6> run
### Loading internal memory, Based Addr = 0x00000000, Length = 0x04000000 ###
### Load internal memory Complete ###
VSIM 7> run
# TUBE: HELLO   EASY!!
# TUBE: Program exit
# ** Note: $finish    : ./design/EASY/Tube.v(171)
#    Time: 14230 ns  Iteration: 2  Instance: /TBTic/uEASY/uTube
# 1
# Break in Module Tube at ./design/EASY/Tube.v line 171
```

在transcript的視窗就會看到Tube印出 HELLO EASY!

實作(二)

Address Decoding

實作(二)

- 請同學練習在EASY平台上將外部記憶體掛上Bus
  - ➢ 在Decoder中加入External MEM到位置0x40000000，並在0x40000040位置寫入自己的學號後8碼

步驟:

1. 在design\EASY**Decoder.v**中新增HSEL_Slave4，使HADDR= 0x40000000時選到它

```
  case (HADDR[31:27])
5'b00000 :
    HSEL_Slave1 = 1'b1;      //Internal RAM

5'b00100 :
    HSEL_Slave2 = 1'b1;      //  Tube
5'b00110 :
    HSEL_Slave3 = 1'b1;      //    testSlave
5'b01000 :
    HSEL_Slave4 = 1'b1;      //External RAM


default  :
  HSELDefault = 1'b1;   //  Undefined (Default Slave)
```

實作(二)

2. 在 **EASY.v** 中增加以下幾條線 ：

```
wire          HSEL_ExtMem;
wire [31:0]   HRDATA_ExtMem;
wire          HREADY_ExtMem;
wire  [1:0]   HRESP_ExtMem;
```

3. 再把uExtMem的HSEL_ExtMem接到 uDecoder 的HSEL_Slave4

```
ExtMem uExtMem (
   .HCLK        (HCLK),
   .HRESETn     (HRESETn),
   .HADDR       (HADDR),
   .HTRANS      (HTRANS),
   .HWRITE      (HWRITE),
   .HSIZE       (HSIZE),
   .HWDATA      (HWDATA),
   .HSEL        ( ),
   .HREADYin    (HREADY),

   .HRDATA      ( ),
   .HREADYout   ( ),
   .HRESP       ( )
);
```

```
// The system address Decoder
 Decoder uDecoder (
    .HRESETn      (HRESETn),
    .HADDR        (HADDR),

    .HSELDefault (HSELDefault),  // Default Slave
    .HSEL_Slave1 (HSEL_mem),        // Interna Memory
    .HSEL_Slave2 (HSEL_tube),       // Tube
    .HSEL_Slave3 (HSEL_testSlave),  //testSlave
    .HSEL_Slave4 ( )                //External Memory
 );
```

實作(二)

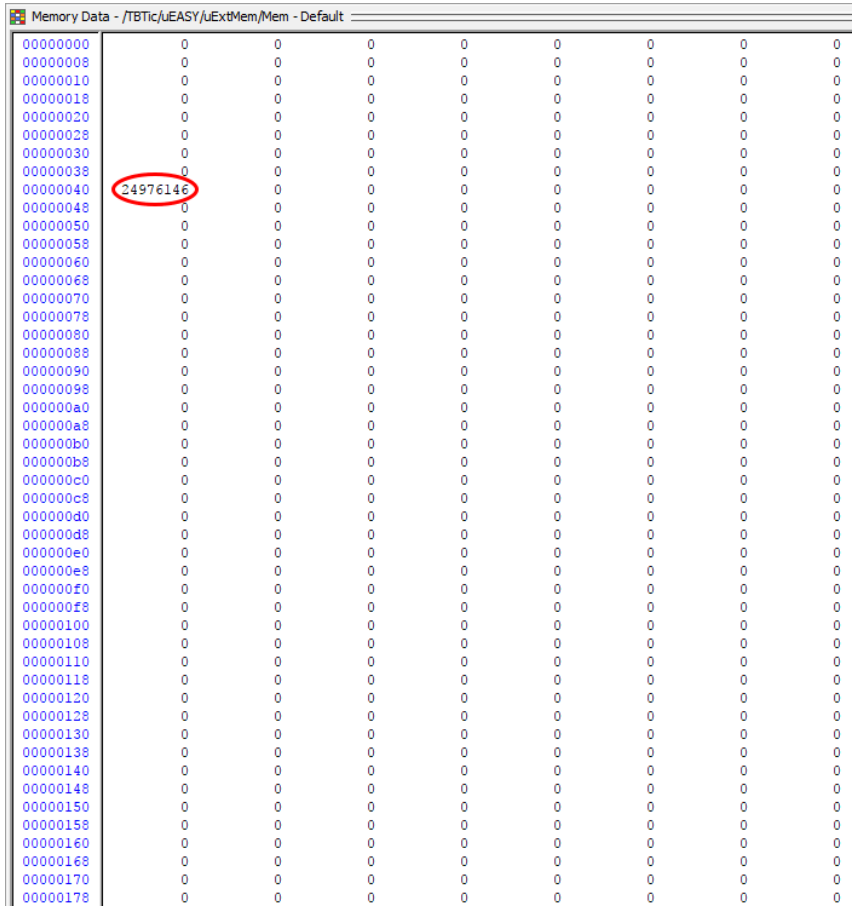
4. 把uExtMem的HSEL_ExtMem, HRDATA_ExtMem, HREADY_ExtMem, HRESP_ExtMem 接到uMuxS2M

```
ExtMem uExtMem (
   .HCLK        (HCLK),
   .HRESETn     (HRESETn),
   .HADDR       (HADDR),
   .HTRANS      (HTRANS),
   .HWRITE      (HWRITE),
   .HSIZE       (HSIZE),
   .HWDATA      (HWDATA),
   .HSEL        ( ),
   .HREADYin    (HREADY),

   .HRDATA      ( ),
   .HREADYout   ( ),
   .HRESP       ( )
);
```

```
// Central multiplexer - slaves to masters
 MuxS2M uMuxS2M (
   .HCLK           (HCLK),
   .HRESETn        (HRESETn),

   .HSELDefault    (HSELDefault),  // Default Slave
   .HSEL_Slave1    (HSEL_mem),         // Interna Memory
   .HSEL_Slave2    (HSEL_tube),        // Tube
   .HSEL_Slave3    (HSEL_testSlave),   // testSlave
   .HSEL_Slave4    (   ),              // External Memory
   .HSEL_Slave5    (Logic0),
   .HSEL_Slave6    (Logic0),

   .HRDATA_S1      (HRDATA_mem),
   .HREADY_S1      (HREADY_mem),
   .HRESP_S1       (HRESP_mem),

   .HRDATA_S2      (HRDATA_tube),
   .HREADY_S2      (HREADY_tube),
   .HRESP_S2       (HRESP_tube),

   .HRDATA_S3      (HRDATA_testSlave),
   .HREADY_S3      (HREADY_testSlave),
   .HRESP_S3       (HRESP_testSlave),

   .HRDATA_S4      (   ),              // External Memory
   .HREADY_S4      (   ),
   .HRESP_S4       (   ),

   .HRDATA_S5      (),
   .HREADY_S5      (),
   .HRESP_S5       (),
```

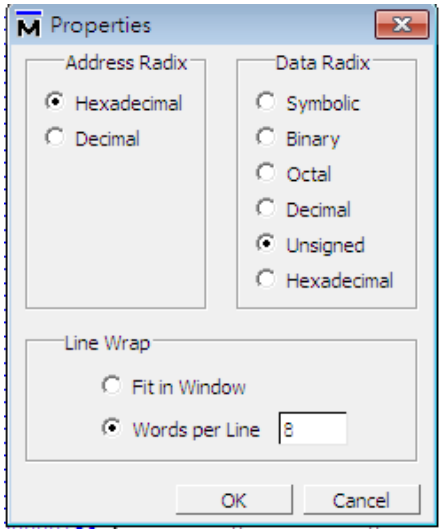
實作(二)

5. 最後再依照實作(一)的方法，寫C code讓CPU在0x40000040的位置寫入自己的學號後8碼，用Modelsim 跑模擬可看到ExtMem被寫入



View -> Memory List
選uExtMem

右鍵->Properties..設定如下



實作(三)

FPGA Verification

實作(三)--挑戰題

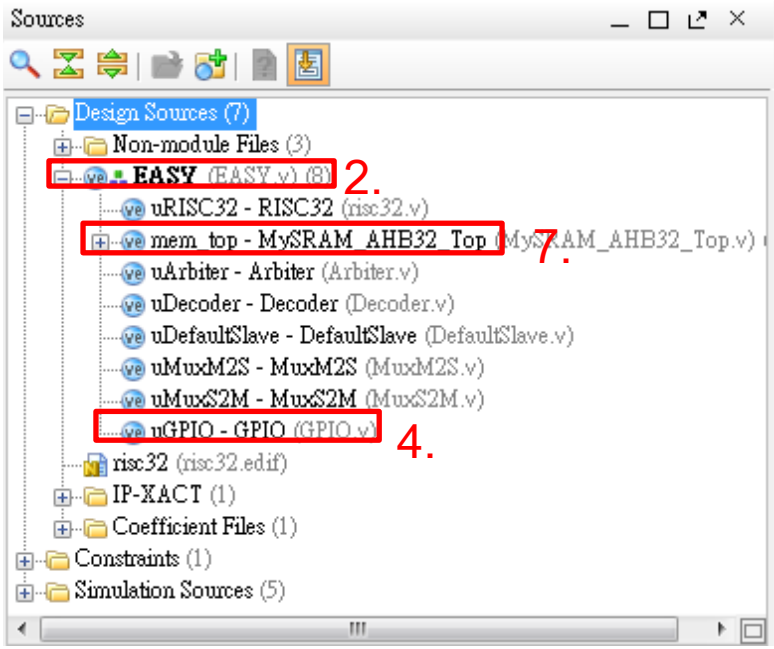
- 請同學在Vivado版本的EASY平台中，掛上Slave(GPIO)在位置0x20000000，用程式讓CPU對GPIO寫入自己的組別號碼，並讓Slave(GPIO)被寫入的值可以透過Vivado合成，燒錄到FPGA板後，在FPGA板子上的LED顯示出組別號碼的ASCII code。

1. 開啟桌面的Vivado 2016.4

   File -> Open Project

   開啟Lab8 -> vivado -> lab8.xpr

   (如有跳出warning，選upgate)



實作(三)

2. 點開EASY.v檔可得知此平台有兩個input(HCLK, HRESTn), 8個 output(LED0~LED7)。

3. 與實作二相同的做法將GPIO接上AHB訊號線

```
wire        HSEL_mem;
wire [31:0] HRDATA_mem;
wire        HREADY_mem;
wire  [1:0] HRESP_mem;

wire        HSEL_LED;
wire [31:0] HRDATA_LED;
wire        HREADY_LED;
wire  [1:0] HRESP_LED;
```

```
// The system address Decoder
  Decoder uDecoder (
    .HRESETn      (HRESETn),
    .HADDR        (HADDR),

   .HSELDefault (HSELDefault),   // Default Slave
    .HSEL_Slave1 (HSEL_mem),      // Internal Memory
    .HSEL_Slave2 (HSEL_LED)       // LED
 );

  GPIO  uGPIO(
     .HRESETn(HRESETn),
    .HSEL_LED(HSEL_LED),
     .clk(clk55MHz),
     .HREADY(HREADY),

     .LED(LED),
    .HREADY_LED(HREADY_LED),
     .HRESP_LED(HRESP_LED),
    .HWDATA(HWDATA)
     );
```

```
module EASY (
   HCLK,
   HRESETn,

   LED0,
   LED1,
   LED2,
   LED3,
   LED4,
   LED5,
   LED6,
   LED7
   );

input      HCLK;
input      HRESETn;

output LED0;
output LED1;
output LED2;
output LED3;
output LED4;
output LED5;
output LED6;
 output LED7;
```

(EASY.v)

實作(三)

4.點開GPIO.v檔，可得知此GPIO(general purpose I/O)將會讀取記憶體的資料並且用該資料來控制LED燈的開關。

如下所示，GPIO在write_ready==1時，將會把HWDATA [7:0]的資料寫入GPIO中的LED_reg暫存器， 而LED_reg則被 GPIO輸出給EASY.v使用。

```
always@(write_ready or HRESETn)
 begin
    if(!HRESETn)
        LED_reg = 8'b0000_0000;
    else
    begin
        if(write_ready)
        LED_reg = HWDATA[7:0] ;
      else
        LED_reg =  LED_reg;
    end
 end
```

(GPIO.v)

```
output [7:0] LED;
output HREADY_LED;
output [1:0]HRESP_LED;

reg  [7:0] LED_reg ;
reg  write_ready;

assign HREADY_LED = 1'b1;
assign HRESP_LED  = 2'b00;

assign LED = LED_reg ;
```

(EASY.v)

實作(三)

將從GPIO接收到的output LED 連接到EASY平台(EASY.v)的output LED0~LED7，便可控制LED燈的開關。

```
assign        LED0 = ;
assign        LED1 = ;
assign        LED2 = ;
assign        LED3 = ;
assign        LED4 = ;
assign        LED5 = ;
assign        LED6 = ;
assign        LED7 = ;
```

填空

(EASY.v)

5. 燒錄時，必須有一個xdc腳位檔，告訴FPGA 此專案的每個input/output腳位對應這個板子的那些腳位，而LED0~LED7便定義在該檔案中。

Real name of LED in NEXYS

Outputs of EASV.v

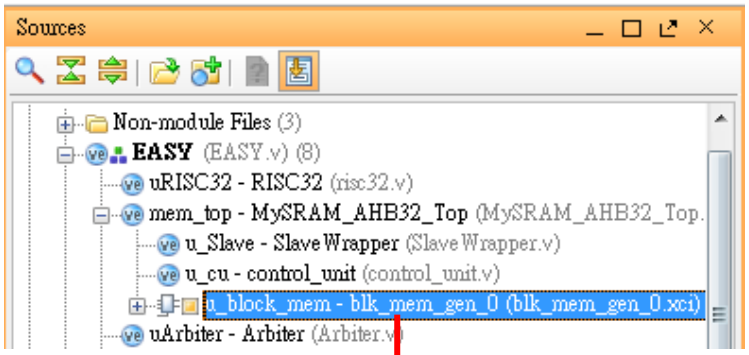
```
31 ## LEDs
32
33 set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
34 set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
35 set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED2 }]; #IO_L17N_T2_A25_15 Sch=led[2
36 set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
37 set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED4 }]; #IO_L7P_T1_D09_14 Sch=led[4]
38 set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED5 }]; #IO_L18N_T2_A11_D27_14 Sch=
39 set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED6 }]; #IO_L17P_T2_A14_D30_14 Sch=
40 set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED7 }]; #IO_L18P_T2_A12_D28_14 Sch=
41 #set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
```

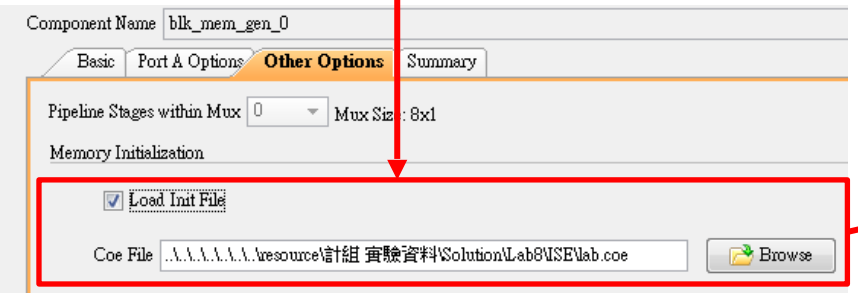
=>此部分助教已經幫同學完成在EASY.xdc

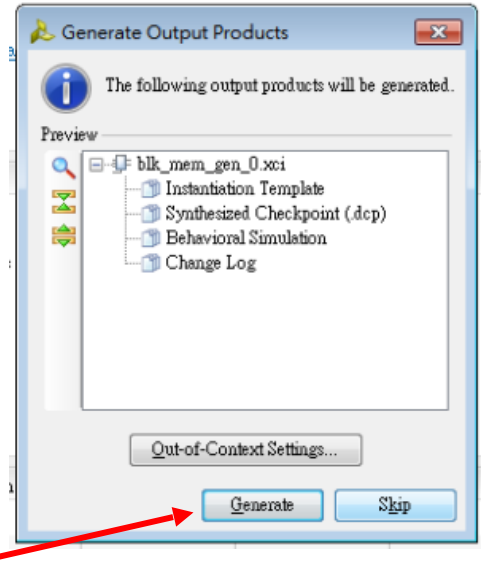
實作(三)

6. 完成上述動作後已完成硬體的部分，請依照實作(一)的方式，對GPIO寫入你們組別號碼的字元，使用vivado/C_Vivado內的編譯環境，將程式轉為lab.coe 。

7. 將新產生的lab.coe檔載入記憶體中，blk_mem_gen_0並在Other Options分頁下選擇剛剛產生的lab.coe，設定完成後再按下Generate產生新的記憶體。

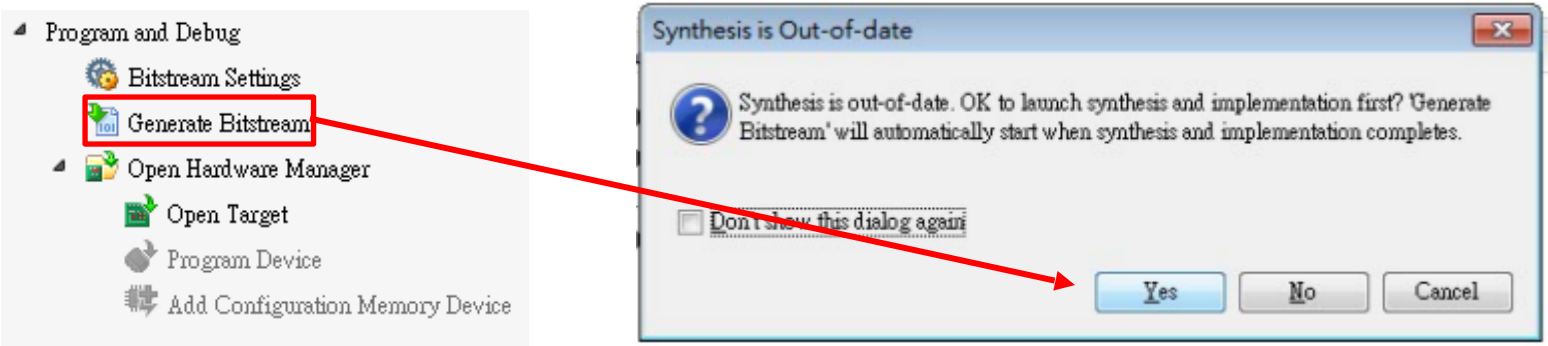




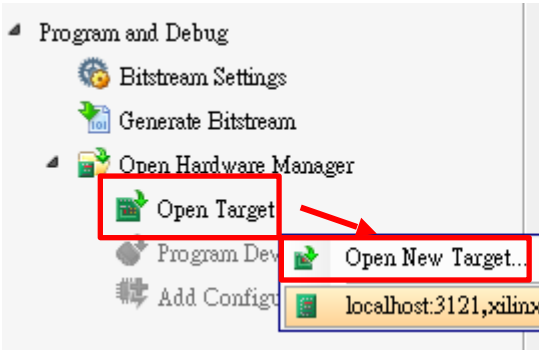


實作(三)

8. 按下Program and Dubug下的Generate Bitstream。



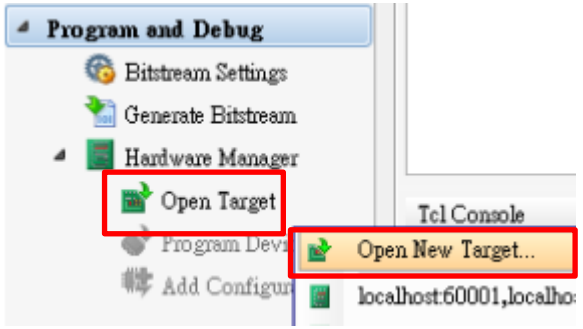
9. 完成之後(需要一點時間)，點開Open Target並選取板子。

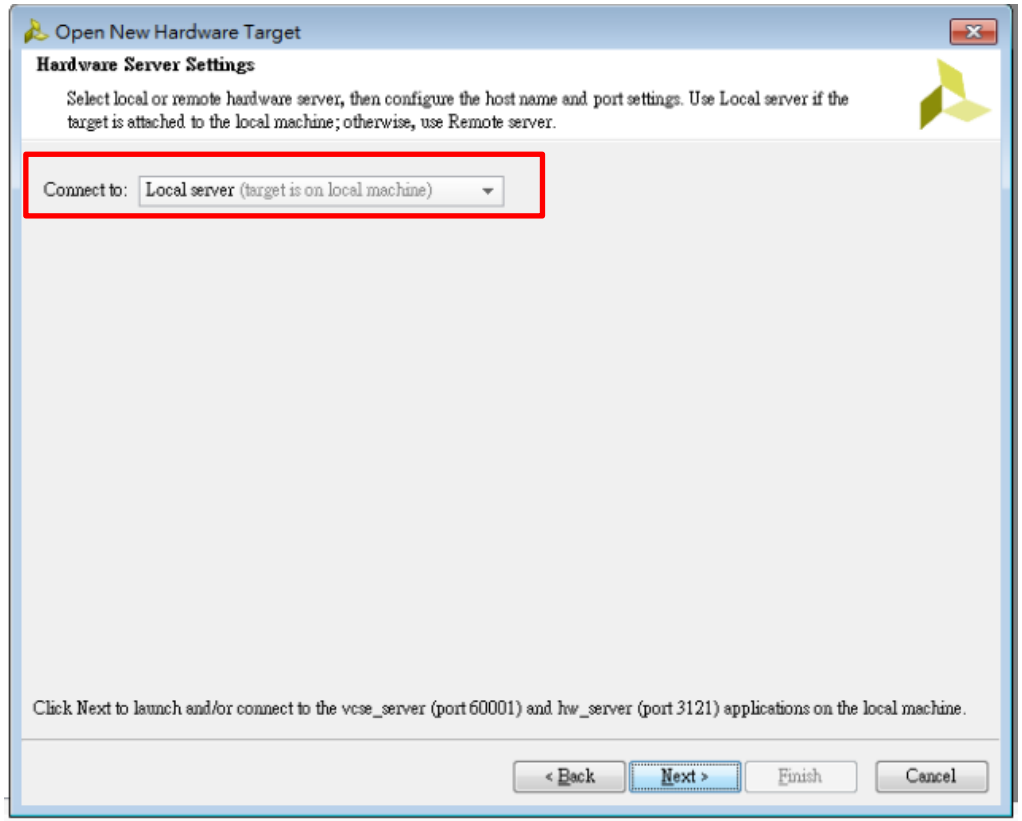


實作(三)

9. 完成之後(需要一點時間)，先跟開發版建立連線。

選取Open Target -> Open New Target, 並且以Local server的方式連線

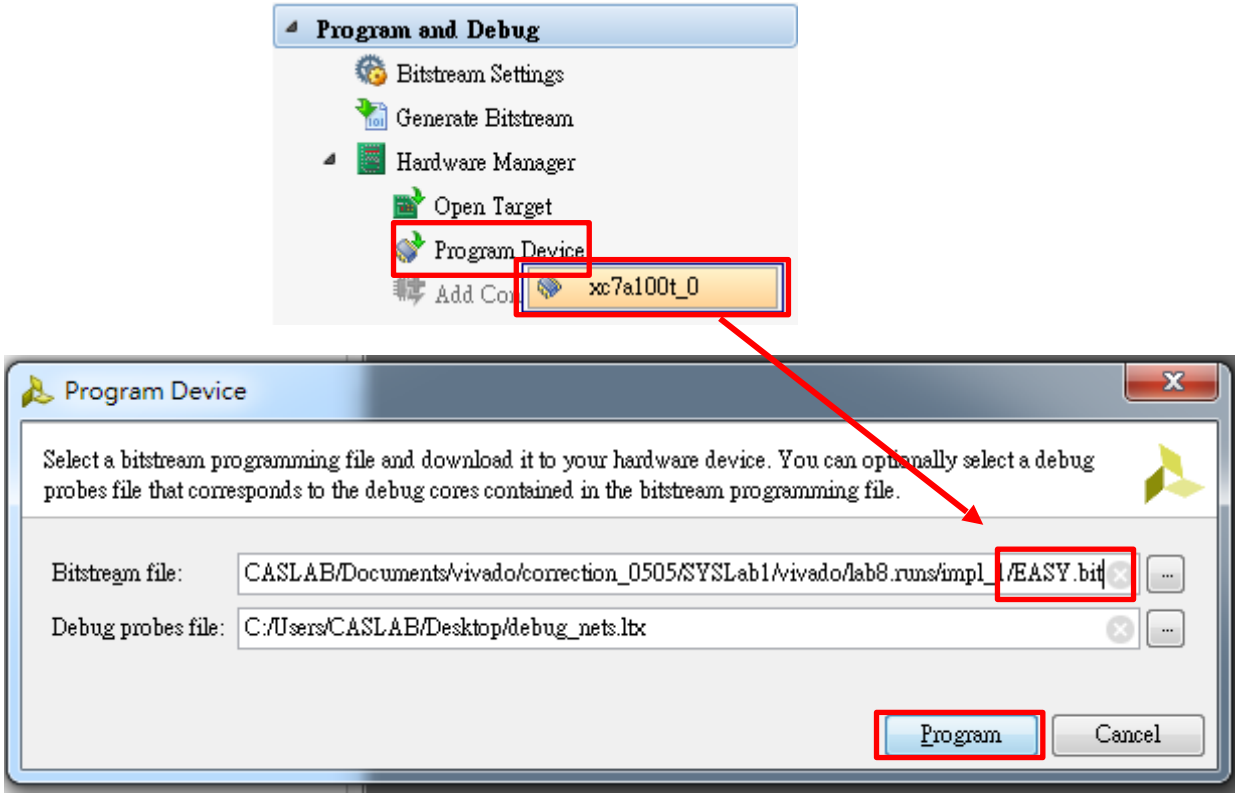




實作(三)

10. 最後再透過Program Device將產生的Bitstream file檔燒入至FPGA，

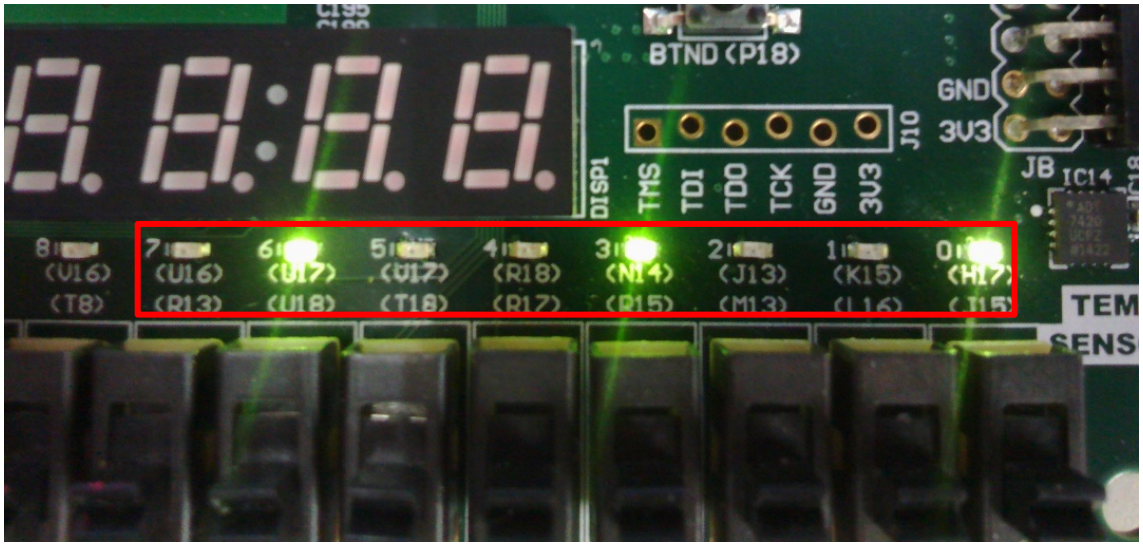
產生的bit檔預設位置在lab8.runs/impl_1資料夾下



實作(三)

11. 剛剛寫入的值將以二進位的方式顯示在開法板的LED燈上。

結果如下圖所示，亮燈的LED為(U17, N14, H17)，即為
(LED6, LED3, LED0) = (0100 1001) = 73



實驗結報

- 結報格式(每組一份)
  - 封面 (第幾組+組員)
  - 實驗內容(程式碼註解、結果截圖)
  - 實驗心得
- 繳交位置
  - ftp://140.116.164.225/ port: 21
  - 帳號/密碼：ca_lab/Carch2020
  - Deadline: 12/07 18:00pm
- TA Contact Information:
  - 助教信箱：anita19961013@gmail.com
  - Lab：92617
  - Office hour：(Tuesday)8:00pm~10:00pm