

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工程研究所

LAB 9: Multi-Master Bus Platform & Generic I/Os

實驗目的

1. 簡介AHB Master Signals
2. 了解Multi-Master之Arbitration
3. 認識Generic I/Os : Timers 、 UART
4. 練習新增Master到平台上
5. 學習平台上Timer的使用
6. 將平台透過Vivado合成，並在FPGA板驗證UART

Background

AHB Master Signals

```

wire [31:0]  HADDRM_RISC32;
wire [1:0]   HTRANSM_RISC32;
wire        HWRITEM_RISC32;
wire [2:0]   HSIZEM_RISC32;
wire [2:0]   HBURSTM_RISC32;
wire [3:0]   HPROTM_RISC32;
wire [31:0]  HWDATAM_RISC32;
wire        HBUSREQM_RISC32;
wire        HLOCKM_RISC32;
wire        HGRANTM_RISC32;

```

圖2-1

Name	Source	Description
HADDR[31:0]	Master	32 bit system bus
HTRANS[1:0]	Master	transfer type
HWRITE	Master	H : write ° L : read °
HSIZE[2:0]	Master	size of the transfer
HBURST[2:0]	Master	Burst type
HPROT[3:0]	Master	Protection Control
HWDATA[31:0]	Master	write data bus
HBUSREQx	Master	bus request
HLOCKx	Master	Lock request
HGRANTx	Arbiter	Bus grant signal

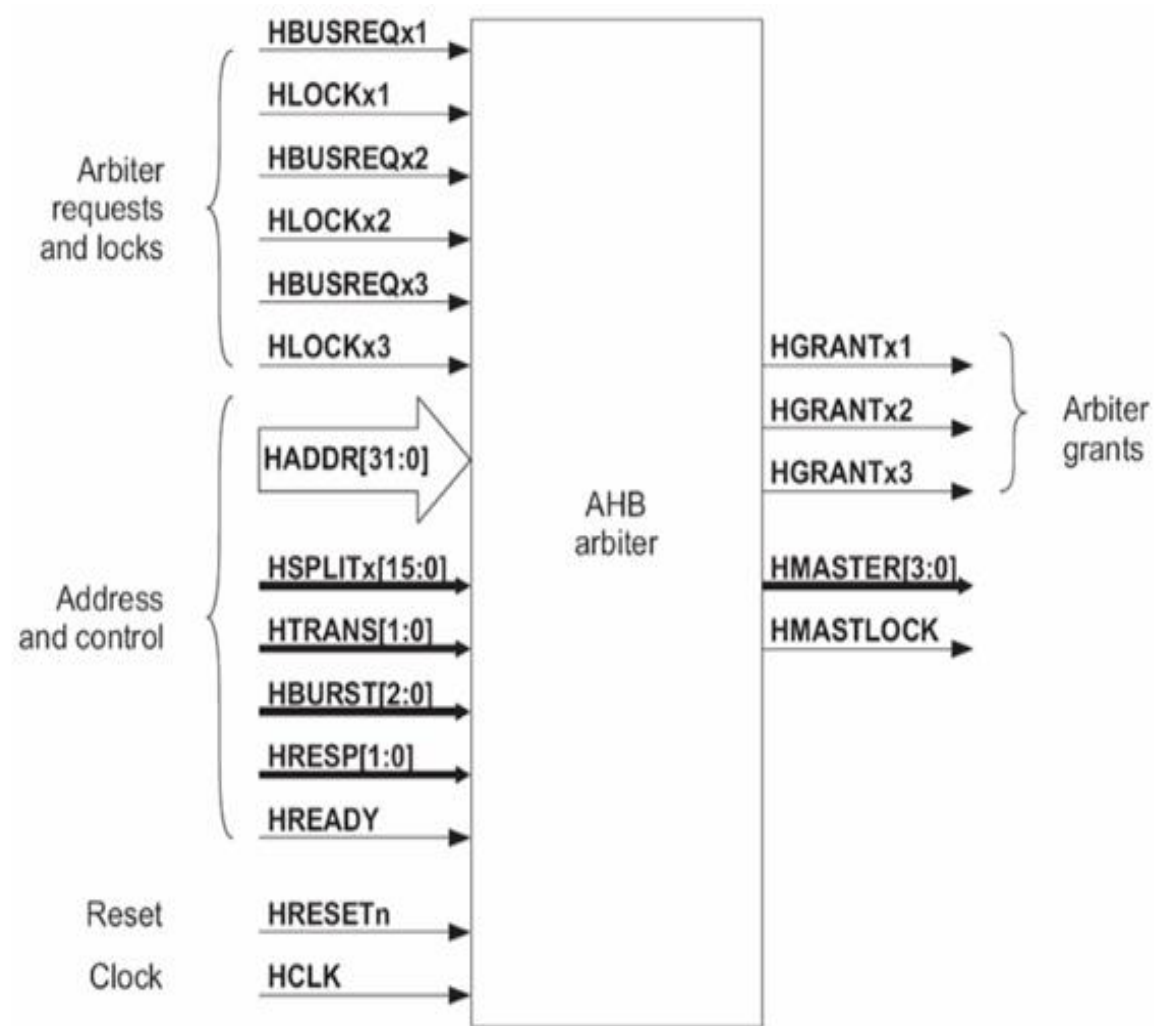
以上共10個Master Signal是每個Master都有的一組訊號：

1. 前7個訊號(**Output**) 會全部接到M2S的Multiplexer
2. HBUSREQx和HLOCKx (**Output**) 會接到Arbiter
3. Arbiter給Master HGRANTx訊號 (**Input**)

About HBUSREQ & HGRANT

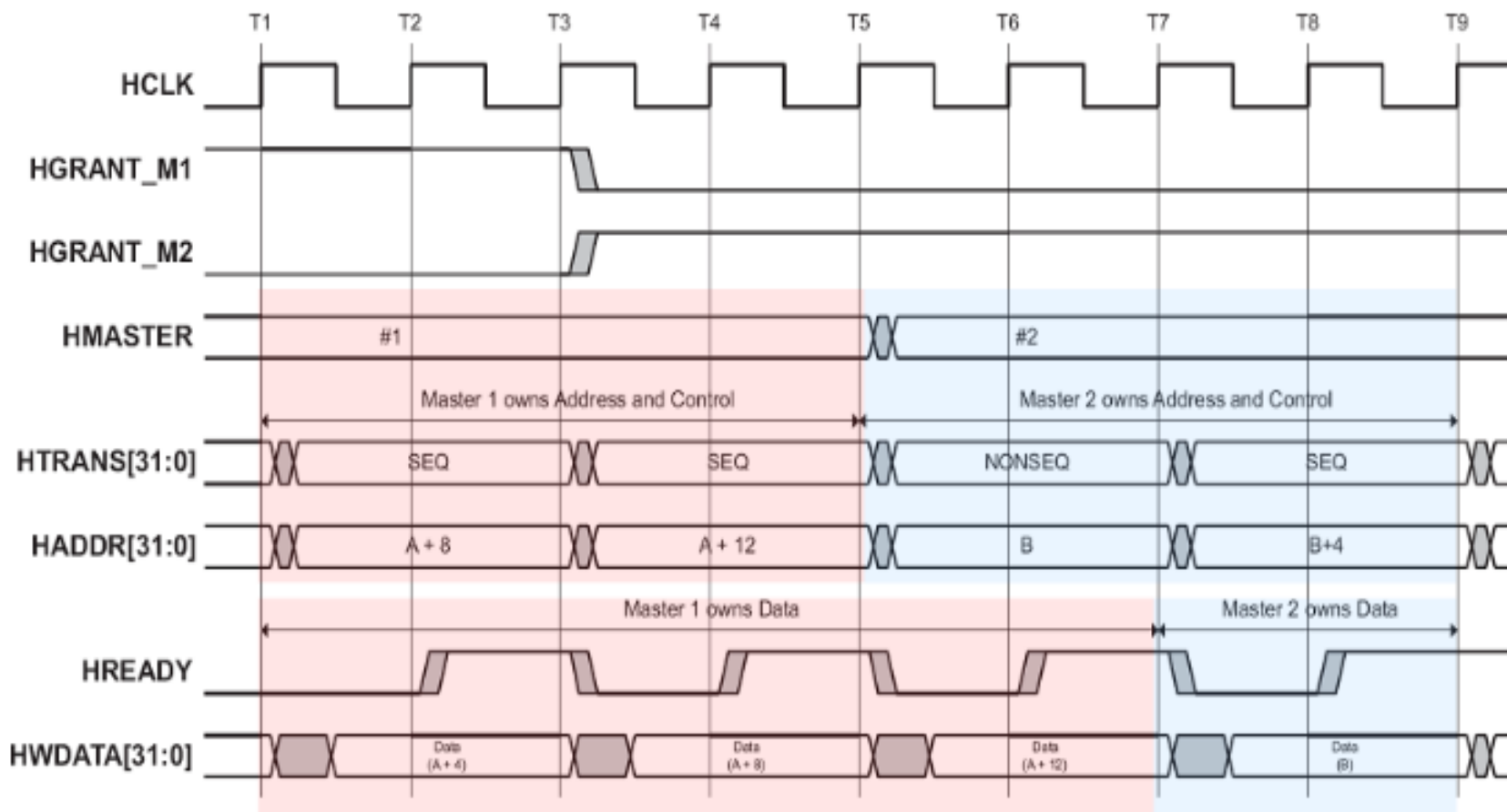
- ⊕ AHB支援Multiple Master，但同一時間只允許一個Master去access bus，因此需要Arbiter去做Arbitration
- ⊕ 當Master要access Bus時，Master將HBUSREQ signal給drive high，Arbiter去sample各個Master的HBUSREQ訊號後再決定哪個Master有最高的priority，將此Master的HGRANT訊號drive high，表示現在是該Master得到Bus的使用權。
- ⊕ 同時，Arbiter會用HMASTER在AHB上廣播現在Grant給那個Master，M2S_Multiplexer就會把那個Master的那7個訊號送給到AHB上給所有Slave看。

Bus Arbiter Interface



Data Bus Ownership

有時就算GRANT給M2時，M1的傳輸還沒完成(HREADY=0)時就須要等一個cycle



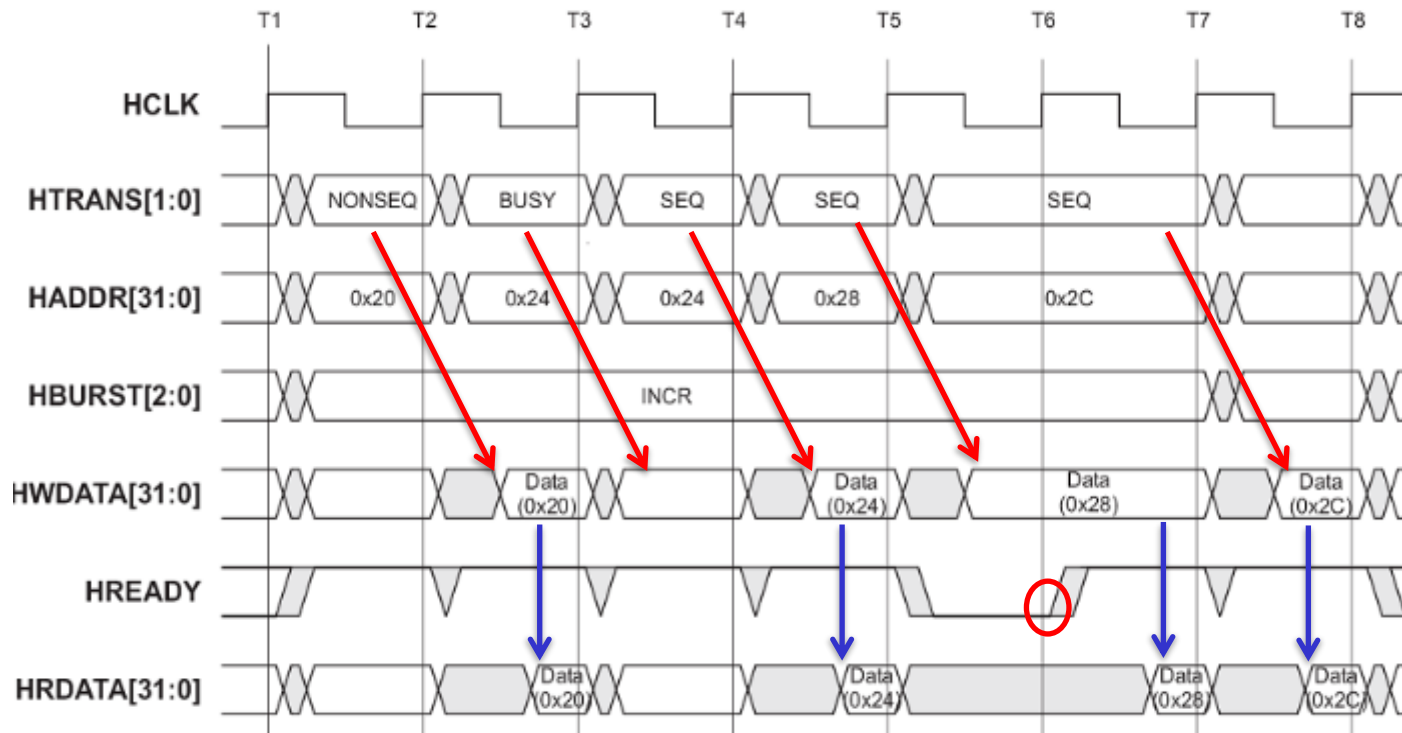
More Detail about Master Signal

Transfer type

HTRANS[1:0]	Type	Description
00	IDLE	Indicates that no data transfer is required.
01	BUSY	The BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers.
10	NONSEQ	Indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer.
11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer.

More Detail about Master Signal

Transfer type example



More Detail about Master Signal

Transfer Size

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size	Description
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

More Detail about Master Signal

Burst Signal encoding

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Arbitration

Arbiter的仲裁機制可以自訂，這邊介紹兩個常用的機制：

- ⊕ **Fixed Priority**：每個Master的priority在設計系統時就先決定好了，不會因為任何因素改變優先順序，優點是在優先權高且極需用到bus的master可以容易拿到bus。
- ⊕ **Round Robin Priority**：拿到bus使用權的Master，在下次仲裁時priority會變最低，這個優點是每一個Master有較平均的bus使用權。

Fixed Priority example

```
assign Request = {Request_3, Request_2, Request_1, Request_0};
```

```
// The default scheme is:
```

```
// Request(3) is the highest priority
// Request(0) is the second highest priority - This must only be connected to
// a Pause input.
// Request(2) is the middle priority.
// Request(1) is the lowest priority and default bus master - This input is
// usually used for an uncached ARM core.
```

事先訂好
優先權

```
// Bus master 0 is reserved for the dummy bus master, which never performs
// real transfers. This master is granted when the default master is
// performing a locked transfer which has received a split response.
```

```
always @ (Request or SplitMask)
```

```
begin
```

```
if ( Request[3])
```

```
    TopRequest = 4'b0011;
```

```
else if ( Request[0])
```

```
    TopRequest = 4'b0000;
```

```
else if ( Request[2])
```

```
    TopRequest = 4'b0010;
```

```
else if ( Request[1])
```

```
    TopRequest = 4'b0001;
```

圖2-8

優先權 : Master3 > Master0 > Master2 > Master1

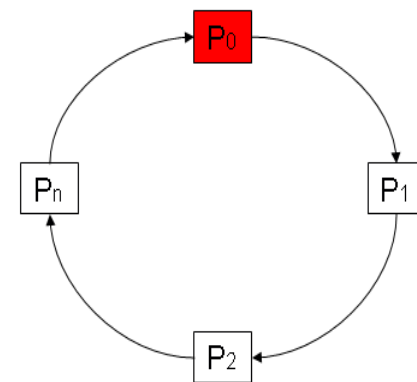
Round-Robin Scheduling

- ⊕ In OS, Round-Robin Scheduling是特別為 **Time-sharing** 設計的。在這種設計中，把一小段時間定義為時間配額(Time Quantum)或時間分槽(Time Slice)，每個 process 皆有固定時間量來使用 CPU，當 process 使用 CPU 超過一個 Time Slice 時，則計數器(Timer)會產生中斷促使此 process 從 Running state 回到 Ready state。

- ⊕ In the hardware arbiter:

- ✓ 假設有 $a > b > c > d$ 順序的priority

我們要做的**Round-Robin**機制是當上一個**AddrMaster**是**b**(**b**搶到當下的Bus所有權)的話，下次的priority會變成 **$c > d > a > b$** ，依此類推....



認識 Generic I/Os : Timers 、 UART

Timer 的運作機制:

- ⊕ 系統計算時間的方式常常是基於內部的某些硬體設備（如振盪器的振動頻率），在由系統控制這些設備的行為所代表的意義。

Timer interrupt 的意義:

- ⊕ 根據 timer 而產生的中斷就叫做 timer interrupt，OS 會在每次 timer 中斷來時做 scheduling 的動作。當 timer interrupt 的頻率過低時，系統的反應會比較遲鈍，因為 context switch 的頻率相對變低了，但當 timer interrupt 過高時，系統則會因為不斷處理 timer interrupt 而變得沒有效率。

認識Generic I/Os : Timers 、 UART

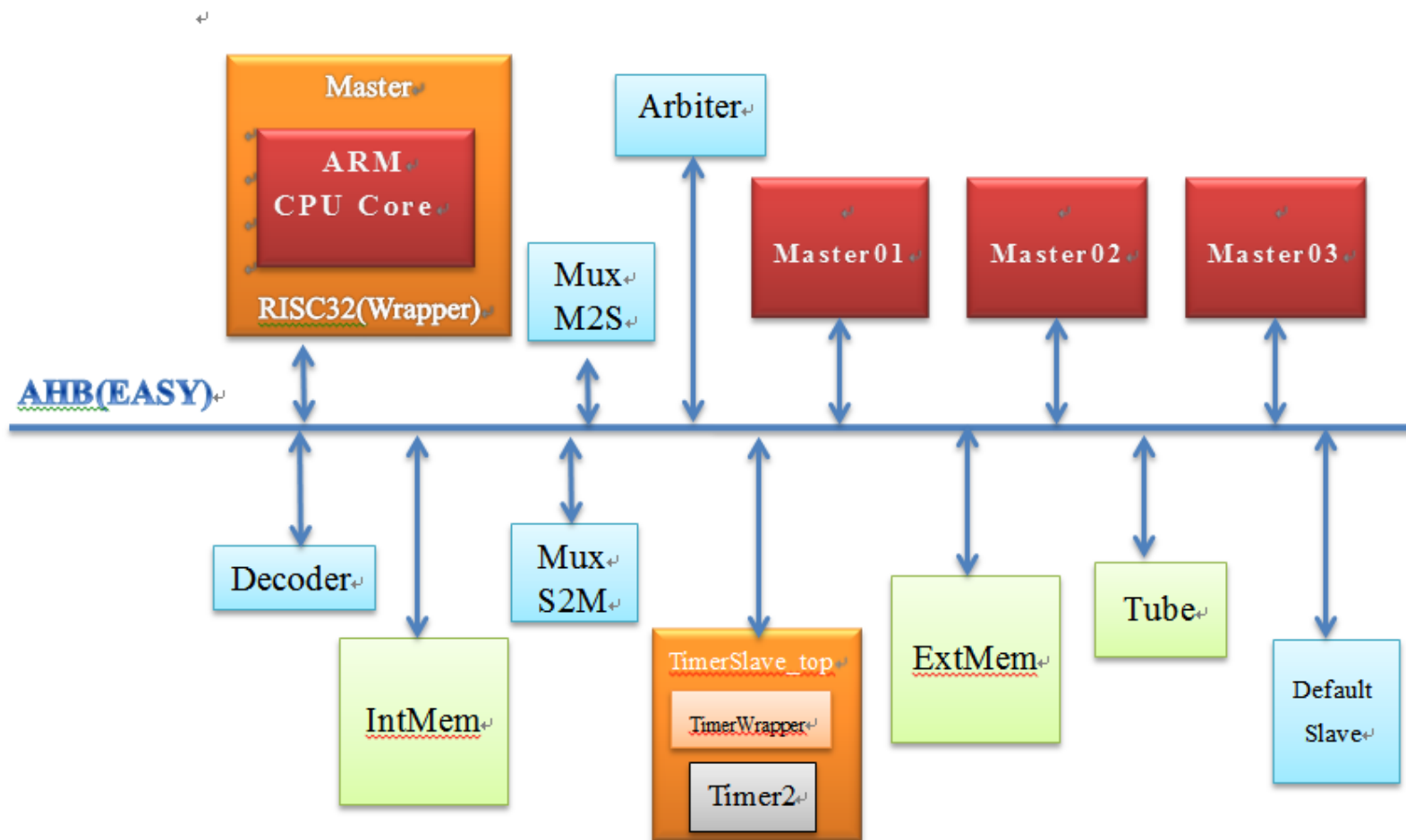
- ⊕ UART (Universal Asynchronous Receiver / Transmitter) 是通用非同步收發器（非同步串列通信口）的英文縮寫，它包括了RS232、RS449、RS423、RS422和RS485等介面標準規範和匯流排標準規範。現在PC的COM 接口均為RS232。若配有多個非同步串列通信口，則分別稱為COM1、COM2，COMxxx。
- ⊕ UART的工作就是從 CPU 一次接收8 bits的資料(parallel)，然後將這些資料 1次 1bit 的送往周邊設備(serially)。同時，UART還可以接收周邊設備傳送來的資料，當組成 8 bits時，再將資料送往 CPU。



實驗系統架構

Example **A**MBA **S**ystem (EASY)

EASY平台 (ARM processor)



實驗環境

⊕ Tool used :

1. ARM GNU Toolchain overview

- 利用GNU ARM Cross Compiler將C轉成arm assembly和machine code

2. Mentor Modelsim

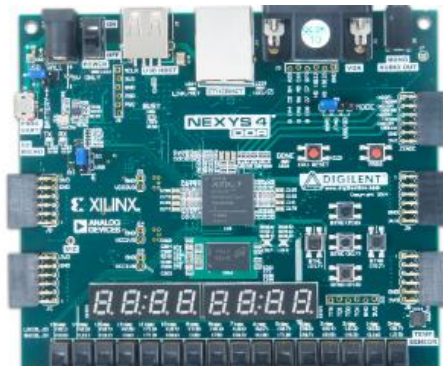
- 看平台上的wave來驗證CPU和AHB的行為

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)

Multi-Master Platform

實作(一)

- ◆ 請同學將Master03加到EASY平台上與其他Master爭奪Bus所有權，並使Master03有最高的priority，若當Master03得到所有權時，在Tube印出'**G**'字元

1. 目前平台上的Master01.v和Master02.v是一樣的Master，他們的動作是不停地搶BUS所有權，當搶到所有權的時候就對Tube寫入'1'和'2'字元，請同學參考Master01完成Master03.v以下空白的Master輸出訊號。

```
assign HBURSTM = 3'b000; //SINGLE
assign HSIZEM = 3'b010; //WORD
assign HWRITEM =
assign HADDRM =           //Tube
assign HTRANSM = 2'b10; // NONSEQ
assign HBUSREQM =
assign HWDATAM =           //ASCII "G"
assign HPROTM = 4'b0000;
```

實作(一)

2. 在EASY平台上依照Master01的規格去新增 Master03所需要用到的訊號線，並把Master03的訊號接到Arbiter和MuxM2S的**第4個位置**

```
Master03 testMaster03
( .HCLK      (HCLK),
  .HRESETn   (HRESETn),
  .HREADYM   (HREADY),
  .HRESPM    (HRESP),
  .HGRANTM   (),

  .HADDRM    (),
  .HTRANS    (),
  .HWRITEM   (),
  .HSIZEM    (),
  .HBURSTM   (),
  .HPROTM    (),
  .HWDATAM   (),
  .HBUSREQM  ()
);
```

```
// The AHB system arbiter
Arbiter uArbiter (
  .HCLK      (HCLK),
  .HRESETn   (HRESETn),
  .HTRANS    (HTRANS),
  .HBURST    (HBURST),
  .HREADY    (HREADY),
  .HRESP     (HRESP),

  .HSPLIT    (5'b00000),

  .HBUSREQ0  (Pause),
  .HBUSREQ4  (),
  .HBUSREQ3  (HBUSREQM_testM02),
  .HBUSREQ2  (HBUSREQM_testM01),
  .HBUSREQ1  (HBUSREQM_RISC32),

  .HLOCK0    (Logic0),
  .HLOCK4    (Logic0),
  .HLOCK3    (Logic0),
  .HLOCK2    (Logic0),
  .HLOCK1    (HLOCKM_RISC32),

  .HGRANT0    (HGRANTdummy),
  .HGRANT4    (),
  .HGRANT3    (HGRANTM_testM02),
  .HGRANT2    (HGRANTM_testM01),
  .HGRANT1    (HGRANTM_RISC32),

  .HMASTER   (HMASTER), // Current bus master
  .HMASTLOCK  (HMASTLOCK)
);
```

```
// Central multiplexer - masters to slaves
MuxM2S uMuxM2S (
  .HCLK      (HCLK),
  .HRESETn   (HRESETn),
  .HMASTER   (HMASTER),
  .HREADY    (HREADY),

  .HADDR_M1   (HADDRM_RISC32),
  .HTRANS_M1  (HTRANSM_RISC32),
  .HWRITE_M1  (HWRITEM_RISC32),
  .HSIZE_M1   (HSIZEM_RISC32),
  .HBURST_M1  (HBURSTM_RISC32),
  .HPROT_M1   (HPROTM_RISC32),
  .HWDATA_M1  (HWDATAM_RISC32),

  .HADDR_M2   (HADDRM_testM01),
  .HTRANS_M2  (HTRANSM_testM01),
  .HWRITE_M2  (HWRITEM_testM01),
  .HSIZE_M2   (HSIZEM_testM01),
  .HBURST_M2  (HBURSTM_testM01),
  .HPROT_M2   (HPROTM_testM01),
  .HWDATA_M2  (HWDATAM_testM01),

  .HADDR_M3   (HADDRM_testM02),
  .HTRANS_M3  (HTRANSM_testM02),
  .HWRITE_M3  (HWRITEM_testM02),
  .HSIZE_M3   (HSIZEM_testM02),
  .HBURST_M3  (HBURSTM_testM02),
  .HPROT_M3   (HPROTM_testM02),
  .HWDATA_M3  (HWDATAM_testM02),

  .HADDR_M4   (),
  .HTRANS_M4  (),
  .HWRITE_M4  (),
  .HSIZE_M4   (),
  .HBURST_M4  (),
  .HPROT_M4   (),
  .HWDATA_M4  (),

  .HADDR      (HADDR),
  .HTRANS     (HTRANS),
  .HWRITE     (HWRITE),
  .HSIZE      (HSIZE),
  .HBURST     (HBURST),
  .HPROT      (HPROT),
  .HWDATA     (HWDATA)
);
```

實作(一)

3. 接好線後重新編譯EASY後，在Modelsim下即可看到以下Bus被Master03搶死的情況發生。

[illegible]

- 4.請同學試著解釋看看為什麼會有這樣的結果?

實作(二)

Timer & Wrapper

實作(二)

- ◆ 請同學練習將Timer(Slave)用 wrapper 包住掛在EASY平台上，並讓CPU去設定Timer在1000個cycle後才叫醒Master01到Master03 的Request

1. 右圖是Timer的Memory Mapping位置。Timer有兩個暫存器用於存放ENABLE和 VALUE，分別位於右表的位置，當ENABLE被寫入1，Timer就根據VALUE內的數值開始倒數，當倒數到0時，便會發出Interrupt叫醒Master

Timer1	
0x50000000	ENABLE
0x50000004	VALUE
0x50000008	空白
0x5FFFFFFF	空白

實作(二)

2. 因此Timers.v 中的 input 和 output如下，需有兩個input，分別是 ENABLE_D 和 VALUE_D，另有timer_itr 的output用於叫醒Master

```
module Timers ( HCLK, HRESETn, HSELT , ENABLE_D, VALUE_D, HADDR, ACK,timer_itr);
```

```
    input      HCLK ;
    input      HRESETn;
    input      HSELT;
    input [31:0] ENABLE_D;
    input [31:0] VALUE_D;
    input [31:0] HADDR ;
    input      ACK;
    output     timer_itr ;
```

3. 而Slave的input和output需要一個 wrapper來把Slave的訊號(IP_開頭)轉換成AHB上的訊號(H_開頭)

TimersWrapper

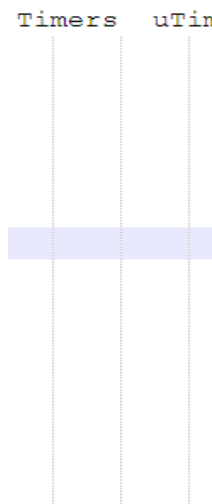
BUS

```
    uTimers_Wra (
//AHB Bus Input
        .HCLK            (HCLK),
        .HRESET_n        (HRESET_n),
        .HADDR            (HADDRS),
        .HTRANS           (HTRANS),
        .HWRITE           (HWRITES),
        .HSIZE            (HSIZES),
        .HWDATA           (HWDATAS),
        .HSEL_slave       (HSELS),
        .HREADY_in        (HREADY),
//AHB Bus Output
        .HRDATA           (HRDATAS),
        .HREADY_out       (HREADYOUTS),
        .HRESP            (HRESPS),
//To IP (Timers)
        .IP_SEL           (IP_SEL),
        .IP_ADDR          (IP_ADDR),
        .IP_WRITE         (IP_WRITE),
        .IP_MASK          (IP_MASK),
        .IP_WDATA         (IP_WDATA),
//From IP (Timers)
        .IP_RDATA         (IP_RDATA),
        .IP_READY         (IP_READY),
        .IP_ERROR         (IP_ERROR)
    );
```

Timer

實作(二)

4. 請在TimersSlave_top.v 中，
將TimersWrapper的訊號拉到
uTimers中沒有接線的地方，
並把Timer的timer_itr接到
TimersSlave_top的output IRQ



```

Timers  uTimers (
    //AHB Bus Input
    .HCLK      (HCLK),
    .HRESETn   (HRESET n),
    .HSELT     (IP_SEL),
    .ENABLE_D  ( ),
    .VALUE_D   ( ),
    .HADDR     (IP_ADDR),

    //Timer input
    .ACK       (ACK),
    //Timer output
    .timer_itr ( )
);
  
```

5. 在Master01.v中，新增input腳 **wake**
並 assign HBUSREQM = (wake==1'b1) ? 1'b1 : 1'b0 ;
6. 最後在EASY.v中加入TimersSlave_top，把Slave的訊號線與Decoder和
MuxS2M接起，並把output **IRQ**用線Timer1_IRQ接到所有Master01~03
的input **wake**。(EASY.v空的訊號都要接)

實作(二)

7. 掛好Timer在0x50000000之後，用C code讓CPU去寫入Timer1的
ENABLE=1和VALUE=1000

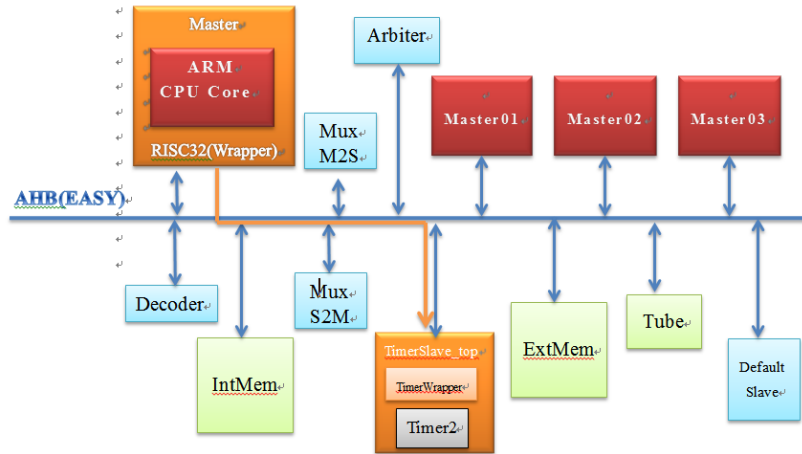
```
void main_function ()
{
    volatile int *tube = (int*) 0x20000000;
    volatile int* Timer1;
    Timer1 = (int*) 0x50000000;
    Timer1[0] = 1;
    Timer1[1] = 1000;
    while(1){}
```

8. 最後用ModelSim模擬約30 us，得到以下結果及代表成功用Timer喚醒Master01到Master03了！

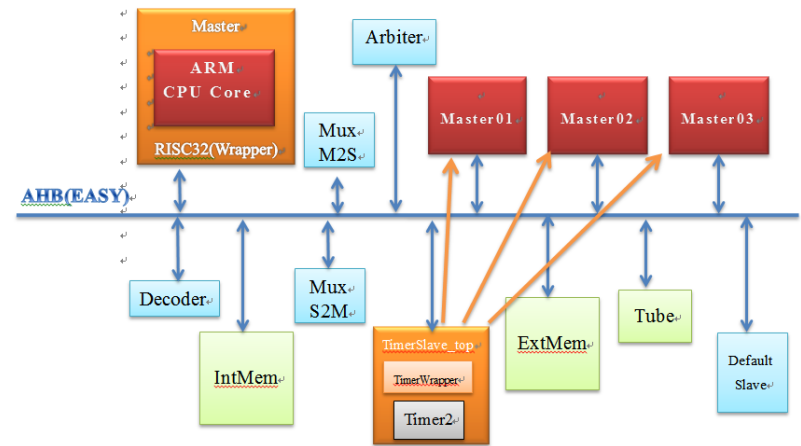
[illegible]

實作(二)

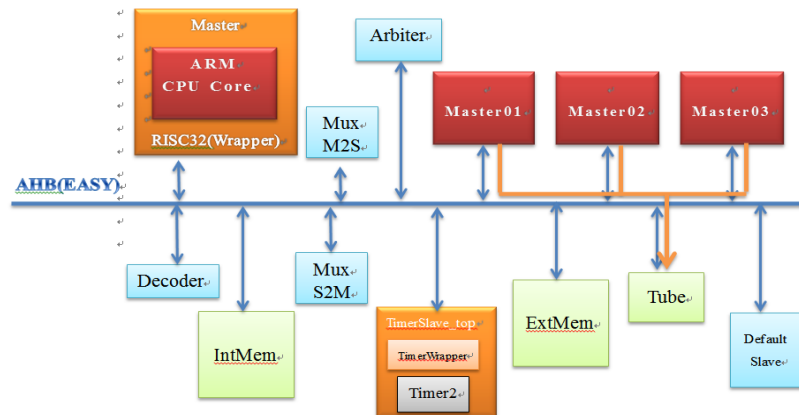
1. CPU enable Timer1



2. Timer1 Interrupt Master01~03(wake up)



3. Master01~03 fight for writing on Tube



實作(三)

FPGA Verification

實作(三)--挑戰題

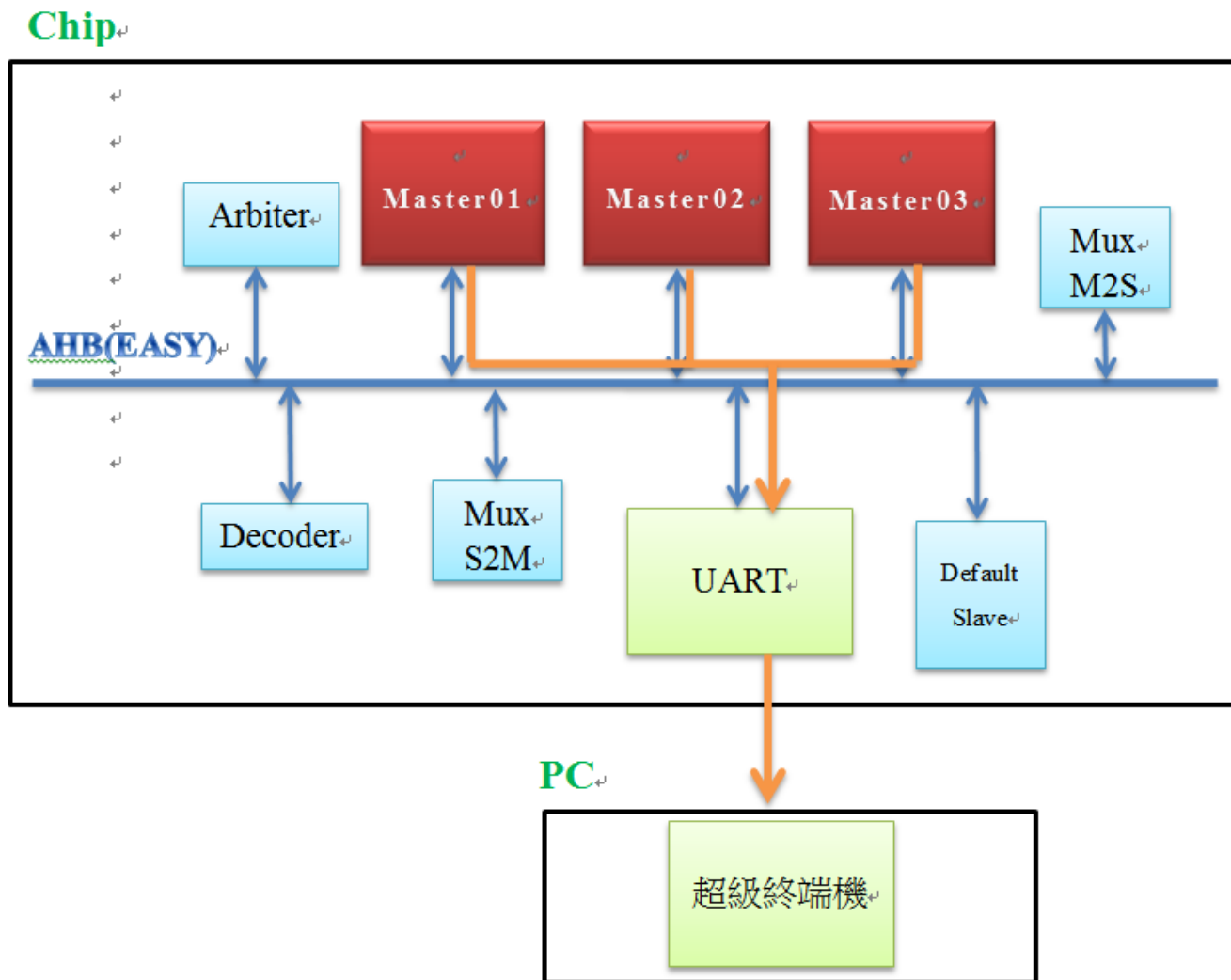
- ◆ 請同學將實作(一)中，Arbiter的Arbitration algorithm改成**Round-Robin**的形式，並用Vivado合成電路燒到FPGA板子之後，由**UART**回傳到電腦的超級終端機中，看3個Master分別對UART寫入的狀況。

- ✓ 提示：假設有 $a > b > c > d$ 順序的priority
我們要做的**Round-Robin**機制是當上一個**AddrMast**是**b**(**b**搶到當下的Bus所有權)的話，下次的priority會變成 **$c > d > a > b$** ，依此類推....

```
always @ (Request or AddrMaster)
begin
    if (
        )
    begin
        if ( Request[4])
            TopRequest = 5'b00100;
        else if ( Request[3])
            TopRequest = 5'b00011;
        else if ( Request[2])
            TopRequest = 5'b00010;
        else if ( Request[1])
            TopRequest = 5'b00001;
        else if ( Request[0])
            TopRequest = 5'b00000;
        else
            TopRequest = 5'b00000; // Dummy master
    end
    else if (
        )
    begin
        if ( Request[0])
            TopRequest = 5'b00000;
        else if ( Request[4])
            TopRequest = 5'b00100;
        else if ( Request[3])
            TopRequest = 5'b00011;
        else if ( Request[2])
            TopRequest = 5'b00010;
        else if ( Request[1])
            TopRequest = 5'b00001;
        else
            TopRequest = 5'b00000; // Dummy master
    end
    else if() //請同學完成剩下Round-Robin code

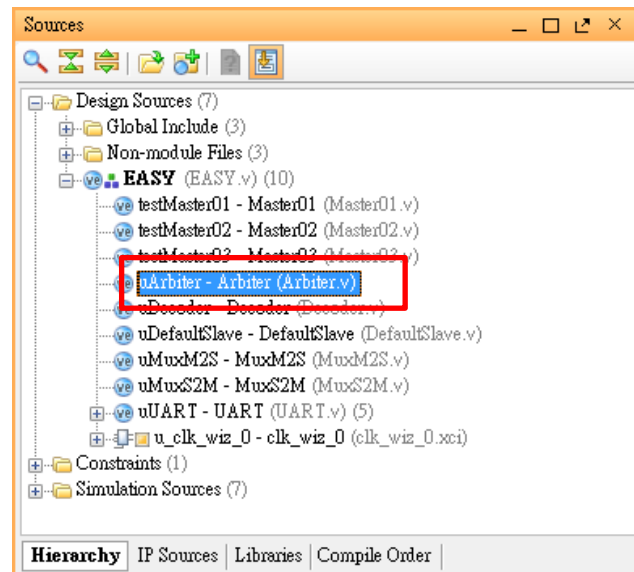
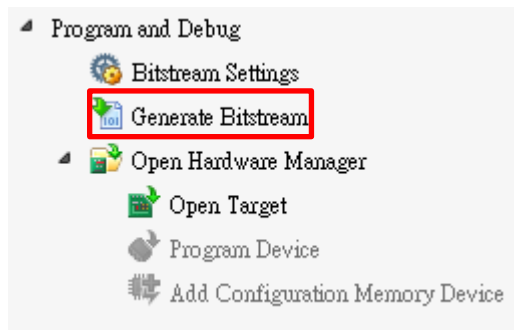
```

實作(三)



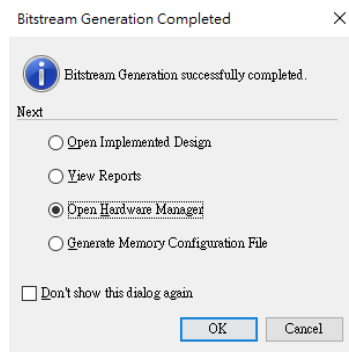
實作(三)

1. 開啟Vivado，File -> Open Project
開啟SYSLab2\vivado\lab9.xpr
2. 修改Arbiter.v的內容，實踐Round-Robin的機制
3. 按下Program and Debug下的Generate Bitstream

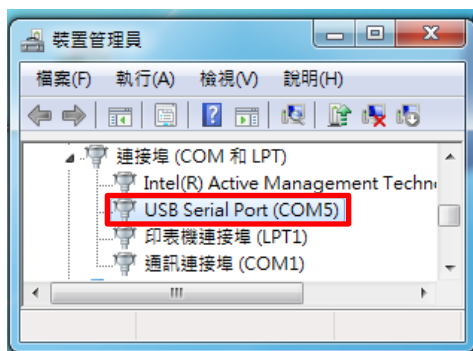


實作(三)

4. 完成後Open Hardware Manager並跟開發板連線，Program Device將資料燒入FPGA。產生的bit檔放置在./SYSLab2/vivado/lab9.runs/impl_1



5. UART透過USB傳輸資料至電腦，因此我們需先透過裝置管理員找出該USB的名稱，如下圖所示，此例USB Serial Port的名稱為COM5。

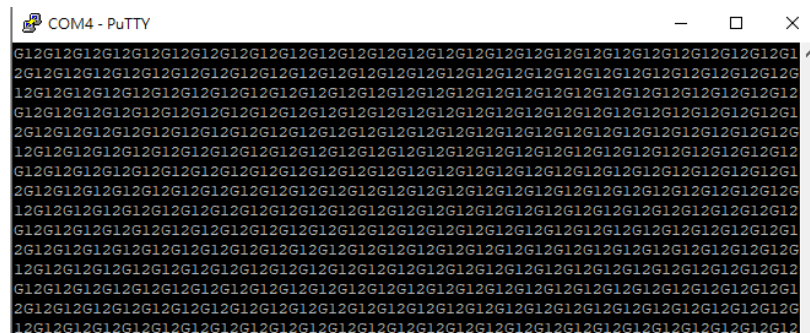
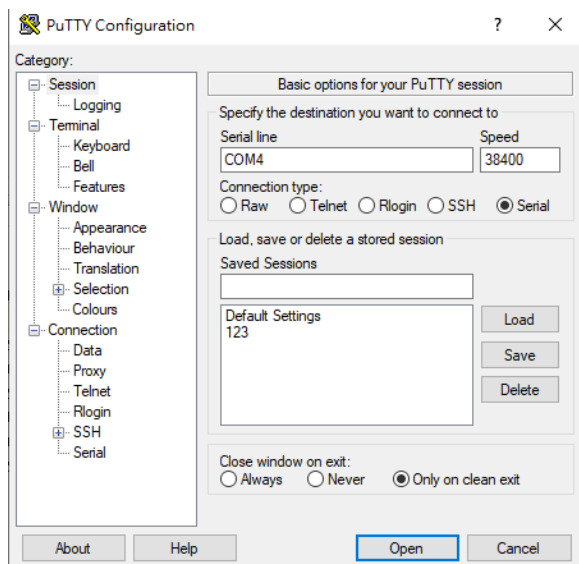


實作(三)

6. 利用終端機Putty建立電腦與板子的連線。

選取Serial -> Serial line(Ex: COM5) & Speed (38400) -> Open

最後我們可以觀察到UART的傳送值與前面的模擬結果相符。



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: 12/21 18:00pm

⊕ TA Contact Information:

- 助教信箱 : anita19961013@gmail.com
- Lab : 92617
- Office hour : (Tuesday)8:00pm~10:00pm