Contents

9. ASIC Logic

9.1. 實驗目的

了解如何運用 ARM Logic Module 及 ARM Integrator 驗證自己設計的 IP。

9.2. 實驗原理

9.2.1. Introduction

While designing an IP, it is important to make sure the design could work as part of the system. The simplest and the most direct way is to port our design to an FPGA and verify the results.
Note that prototyping using FPGA cannot be done before the designer starts the HDL hardware design procedure, therefore the designer can only verify the design after the design's HDL coding has been completed. Yet using virtual prototyping, the designer would have more design space to explore. It allows the designer to know how their IP might work with the system, and know their options and limitations more before detailed architectural design, especially from the system's point of view.
ARM Integrator's Logic Module can work alone like most FPGA development board provided by ALTERA or XILINX. Still it can also be attached to ARM Integrator's Application Platform and operates with Core Module together. This configuration provides a complete system which represents as a basic platform model in SOC design.

9.2.2. Basics and Work Flow for Prototyping with Logic Module

ARM Logic Module (LM) provides a platform for developing Advanced Microcontroller Bus Architecture (AMBA), Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB), and peripherals for the use with ARM-based system.

The LM can be used as a standalone system like an FPGA test board, or used with a Core Module (CM) and an Application Platform (AP). It can also work as a CM with AP if a synthesized ARM core is programmed into the FPGA. The last option is to stack several LMs together without an AP motherboard if one of the LMs provides the system controller functions of a motherboard.

The LM contains several components as shown in ***Figure 1***. An LM has an ALTERA or XILINX FPGA, our LM uses XILINX FPGA. It has a configuration PLD and a flash memory for storing FPGA configurations. A 1MB ZBT SSRAM is provided for local storage. There's a prototyping grid where the user can attach small circuits to LM. The system bus connector provides connection to AP motherboard or to other modules. The LM also incorporates with several peripherals such as LEDs, user-definable push button and switches. The layout of the LM is illustrated in ***Figure 2***. Please refer to "Integrator/LM-XCV600E+ User Guide" for further architecture details.

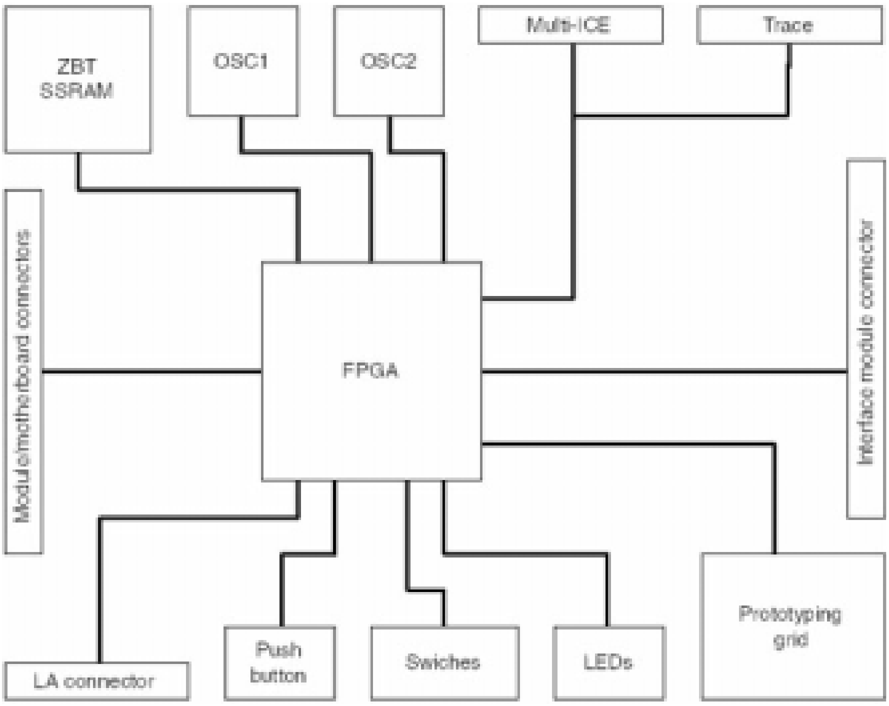

Figure 1. The architecture of a Logic Module.

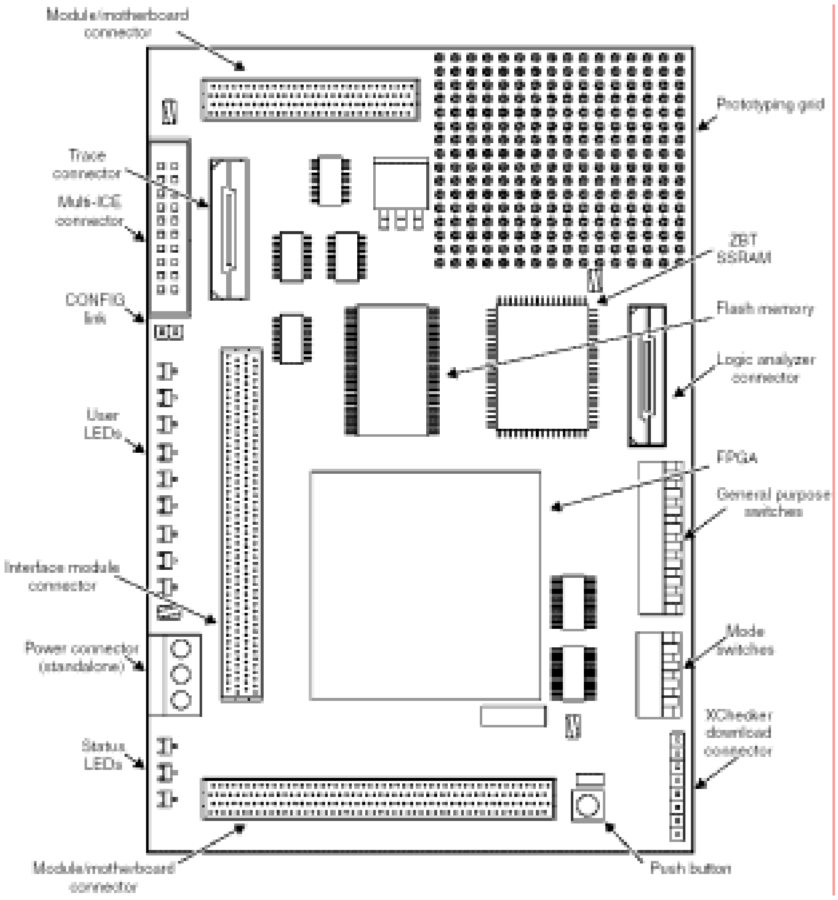

Figure 2. Logic Module's board layout.

The LM can be linked with JTAG, Trace, or logic analyzer connectors. There is a **configuration mode**, which changes the JTAG signal routing and is used to download new PLD or FPGA configurations.

9.2.3. FPGA tools

FPGA Compiler II (FCII) is a GUI synthesis tool for Xilinx FPGA. ***Figure 3*** illustrates the general synthesis flow for using FCII. The HDL design is imported into the GUI synthesis tool, and the synthesis tool generates the EDIF netlist. Then Xilinx GUI tool will perform place and route to generate the FPGA binary bit data the EDIF netlist as input. The FPGA binary bit data is used to program the FPGA on the LM.

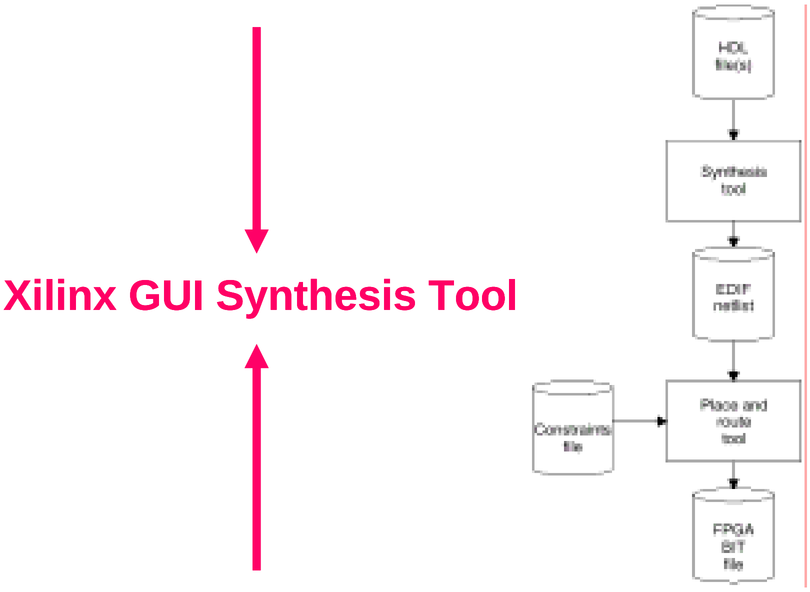

Figure 3. Xilinx FPGA synthesis flow.

9.2.4. Basic Platforms: AHB and ASB

The example contains two versions of implementation which support the following two configurations:

- AHB MB and AHB peripherals
- ASB MB and AHB peripherals

Figure 4 supports the first configuration, and ***Figure 5*** supports the second one.

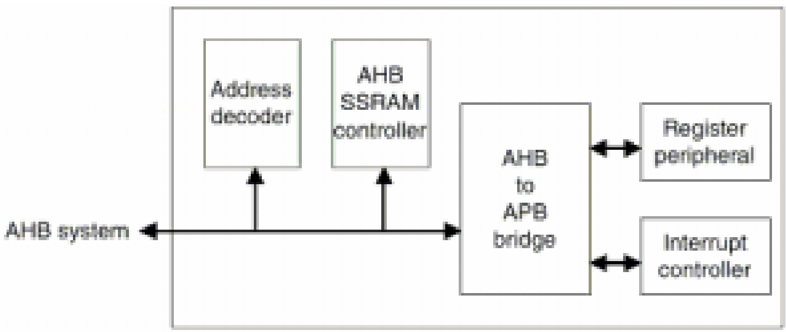

Figure 4. Implementation the support AHB system.

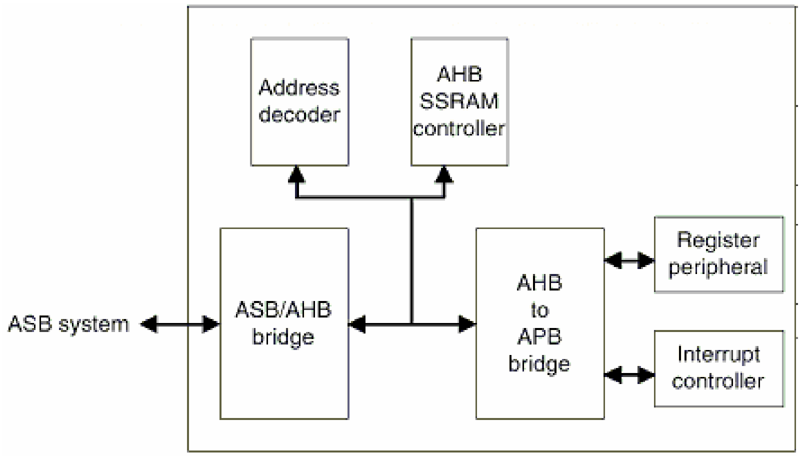

Figure 5. Implementation that supports ASB system.

The alphanumerical LED display on the Integrator AP motherboard can show whether it is AHB or ASB. The letter shown corresponds to either of the two systems, which will be shown below:

- **H**: AHB
- **S**: ASB

In this course, our configuration is illustrated in ***Figure 6***. The blocks inside the dashed bounding box represent the architecture to be programmed into the LM's FPGA.

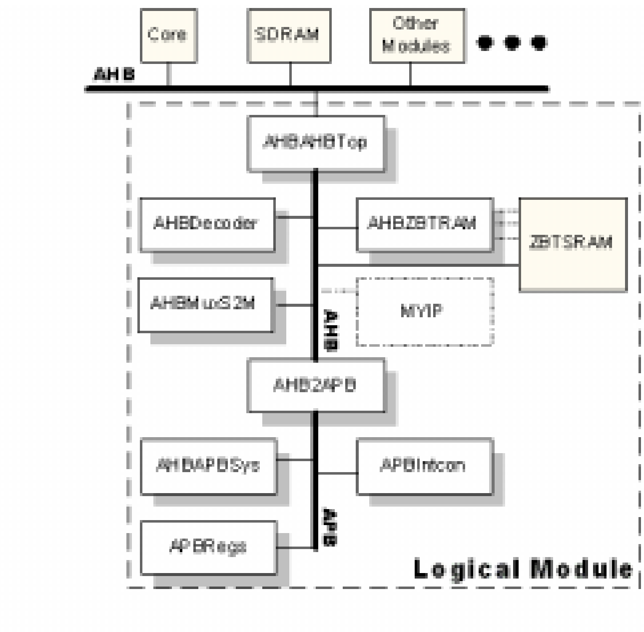

Figure 6. The AHB platform and its block diagram used in this course.

9.2.5. Logic Module Registers

The memory space within a LM and its relation with Integrator's system memory space is illustrated in ***Figure 7***. The custom IP design should use the address space from 0xC2100000 to 0xCFFFFFFF. The description of each LM registers is described in ***Table 1***. The offset address represents the register's offset from the base address.

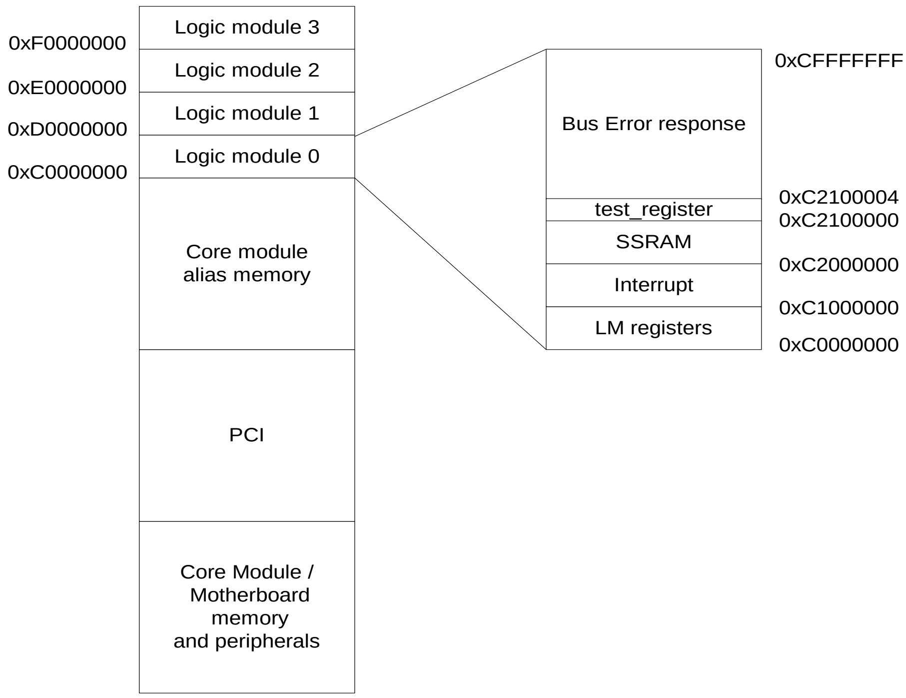

Figure 7. Relations between LM's memory space and the Integrator system's memory space

Offset Address	Name	Type	Size	Function
0x0000000	LM_OSC1	R/W	19	Oscillator divisor register 1
0x0000004	LM_OSC2	R/W	19	Oscillator divisor register 2
0x0000008	LM_LOCK	R/W	17	Oscillator lock register
0x000000C	LM_LEDS	R/W	9	User LEDs control register
0x0000010	LM_INT	R/W	1	Push button interrupt reg.
0x0000014	LM_SW	R	8	Switches register

Table 1. Register map of an LM.

Bits	Name	Name	Function
0	LM_INT	Read	This bit when SET is a latched indication that the push button has been pressed.
		Write	Write 0 to this register to CLEAR the latched indication. Writing 1 to this register has same effect as pressing the puchs button

Table 2. Push button interrupt register.

9.2.6. Interrupt Controller

The interrupt controller in LM manages the IRQs from the user's design and the peripheral devices on LM. The Integrator system treats the LM as a single slave device, therefore there's only one IRQ signal connected from LM to the motherboard.
Figure 8 shows the basic bit-slice structure of the interrupt controller. The Set-Clear register and the "AND" gate can perform interrupt enable masking, so that only the enabled interrupt requests are allowed. The corresponding control registers for interrupt controller are listed in ***Table 3***.

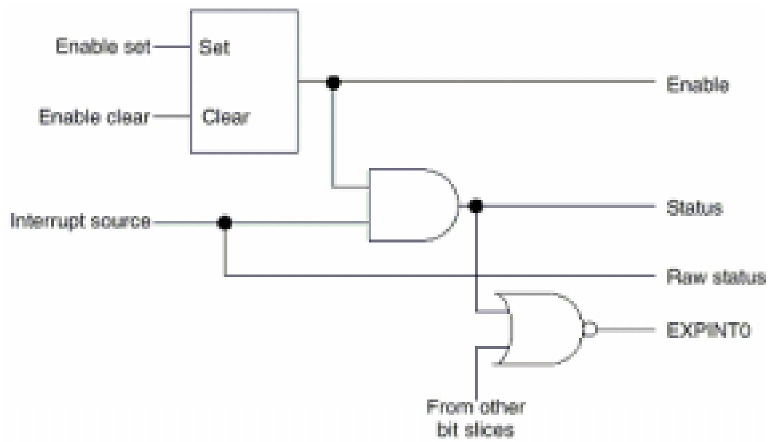

Figure 8. Bit-slice of LM's interrupt controller's structure.

Offset Address	Name	Type	Size	Function
0x10000000	LM_ISTAT	R	8	Interrupt status register
0x10000004	LM_IRSTAT	R	8	Interrupt raw status reg.
0x10000008	LM_IENSET	R/W	8	Interrupt enable set
0x1000000C	LM_IENCLR	R	8	Interrupt enable clear
0x10000010	LM_SOFTINT	R	4	Software interrupt register

Table 3. Interrupt controller's registers.

9.3. 引導實驗

This lab has two examples. The first example demonstrates how to program the Logic Module by programming into the FPGA or into the flash. The second example demonstrates the basics to implement a design prototype by writing the FPGA into the flash on the Logic Module.

The features of each example are shown below:

Example 1:

- Flashes the LEDs on the Logic Module from left to right.
- The speed of flashing the LEDs from left to right can be set by changing the configuration of the 8-way switch.
- FPGA version: programs the FPGA by writing the bitstream image into the FPGA directly. The image will start running right after programming into the FPGA.
- Flash version: programs the FPGA by writing the bitstream image into the flash. The image will start after next power up of the development system.
-

9.3.1. File Descriptions: Example 1

There's only one Verilog HDL file for this example, namely, *example1.v*. The rest are used for bitstream generation and downloading.

File	Description
exampl1.v	Verilog HDL for example1.
pc_par.bat	The batch script for running Xilinx FPGA PnR utilities
Example1.ucf	This is the constraint file defining the pin I/Os.
Example1.ncf	This gives the timing constraints.
map.ncd	This provides general mapping directives.
Example1.ncd	This is the file with specific mapping directives.
bit_gen.ut	Bitstream generation utility
enter	A simple file with a single enter

Table 4. Files for example1.

9.3.2. File Descriptions: Example 2

HDL Files Descriptions

Each block in the LM is described with an HDL design file. The description of each HDL files is provided in ***Table 5***.

File	Description
ASBAHBTop	These files are the top-level HDL that instantiate all of the high-speed

AHBAHBTop	peripherals, decoder, and all necessary support and glue logic to make a working system. The files are named so that, for example, ASBAHBTop.vhd is the top level for AHB peripherals connected to an ASB system bus.
ASB2AHB	This is the bridge required to connect AHB peripherals to an ASB Integrator system.
AHBDecoder	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines and the module ID (position in stack) signals from the motherboard. The decoder blocks also contain the default slave peripheral to simplify the example structure. The Integrator family of boards uses a distributed address decoding system
AHBMuxS2M	This is the AHB multiplexor that connects the read data buses from all of the slaves to the AHB master(s).
AHBZBTRAM	High-speed peripherals require that SSRAM controller block supports word, halfword, and byte operations to the SSRAM on the logic module.
MYIP	A simple IP template with only one single register wrapped with simple AHB slave interface. This file is modified from AHBZBTRAM.
AHB2APB	This is the bridge blocks required to connect APB peripherals the the high-speed AMBA AHB bus. They produce the peripheral select signals for each of the APB peripherals.
AHBAPBSys	The components required for an APB system are instantiated in this block. These include the bridge and the APB peripherals. This file also multiplexes the APB peripheral read buses and concatenates the interrupt sources to feed into the interrupt controller peripheral.
APBRegs	The AOB register peripheral provides memory mapped registers that you can use to: Configure the two clock generators Write to the user LEDs Read the user switch inputs. It also latches the pressing of the push button to generate an expansion interrupt.
APBIntcon	The APB interrupt controller contains all of the standard interrupt controller registers and has an input port for four APB interrupts. Four software interrupts are implemented.

Table 5. The description of each HDL file.

Software File Descriptions

There are four software files in this example. The description of each software file is provided in ***Table 6***.

File	Description
Logic.c	These files are the top-level HDL that instantiate all of the high-speed peripherals, decoder, and all necessary support and glue logic to make a working system. The files are named so that, for example, ASBAHBTop.vhd is the top level for AHB peripherals connected to an ASB system bus.
Logic.h	This is the bridge required to connect AHB peripherals to an ASB Integrator system.
Platform.h	The decoder block provides the high-speed peripherals with select lines. These are generated from the address lines and the module ID (position in stack) signals from the motherboard. The decoder blocks also contain the default slave peripheral to simplify the example structure. The Integrator family of boards uses a distributed address decoding system
Rw_support.s	This is the AHB multiplexor that connects the read data buses from all of the slaves to the AHB master(s).

Table 6. The description for each software file.

Bitstream Geratation File Desicriptions

These files are required to generate the bitstream to be downloaded into Logic Module's flash.

File	Description
pc_par.bat	The batch script for running Xilinx FPGA PnR utilities
example2.ucf	This is the constraint file defining the pin I/Os.
example2.ncf	This gives the timing constraints.
map.ncd	This provides general mapping directives.
example2.ncd	This is the file with specific mapping directives.
bit_gen.ut	Bitstream generation utility
enter	A simple file with a single enter

Table 7. The description for each files needed to generate the bitstream

9.3.3. 實驗步驟

Example1:

Steps for Synthesis with Xilinx ISE 5.1: Example1

0. Extract **example1.zip** to %xilinx%\virtexe\data\

1. Start Xilinx ISE 5.1 from the start menu.

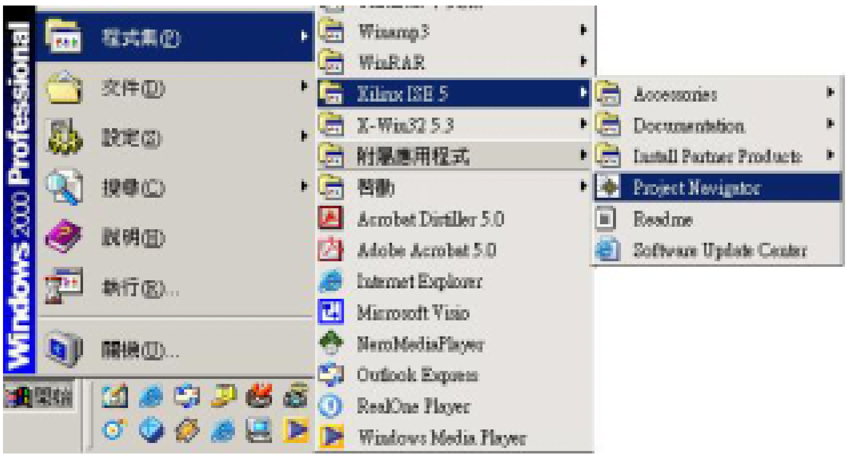

Figure 9. Starting Xilinx FPGA CompilerII's from the Start-Up menu.

2. Create a **New Project**
   – Input the project name (assume: ***example1***)
   – Assign the project location (assume: ***LAB09***)
   – Enter the project target device options:
     (a) Device Family: VirtexE
     (b) Device: xcv2000e
     (c) Package: fg680

(d) Speed Grade: -6
(e) Design Flow: XST Verilog

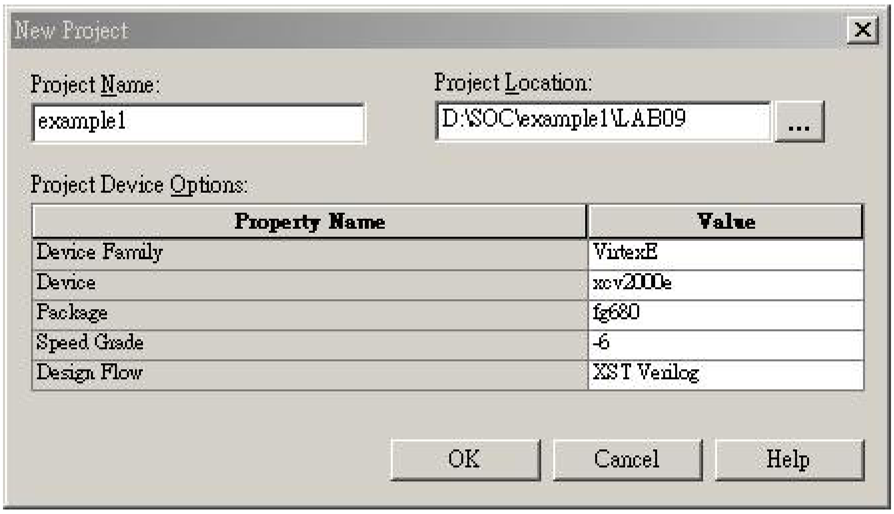

Figure 10. Getting Started window in Project Manager.

3. ***Project→add source File(s)***(*..\LAB\example1\Verilog\example1.v*)
   – Project Manager will analyze the added source file for syntax check.

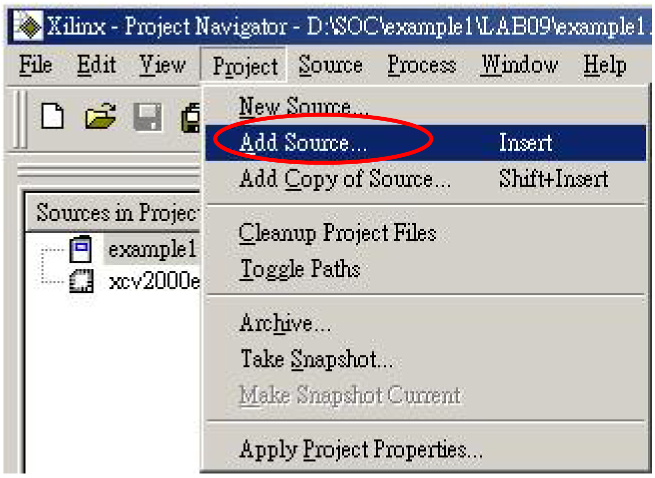

Figure 11. Add Source File(s) to the Project.

4. ***Add example2.ucf file to this project***

5. ***Generate the Binary Bitstreams***
   – Select Top module,
   – Double click the ***Generate Programming File***

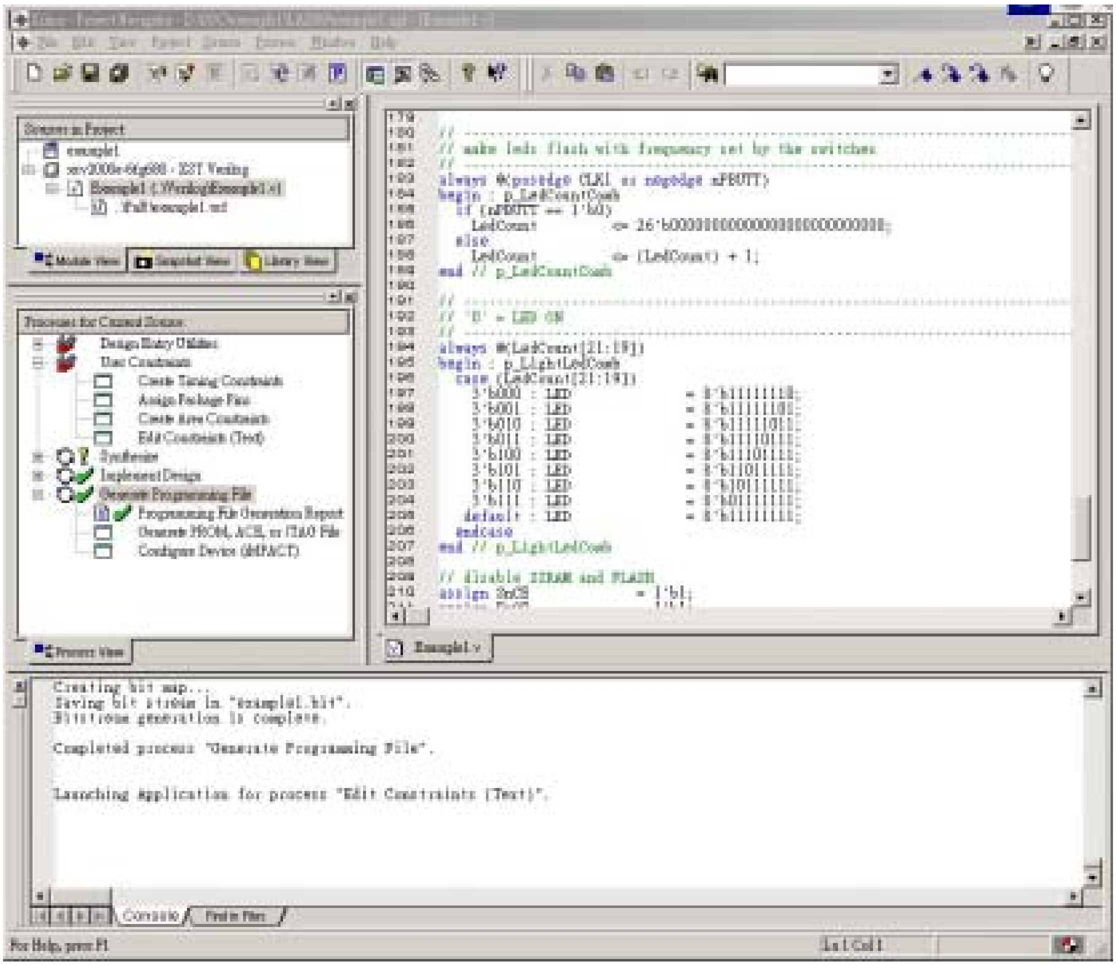

Figure 12. Generate Bitstreams.

Downloading the Binary Bitstreams

1. Open the download setting files .\Lab9\Codes\HW\example1\ **example1_to_flash.brd** and **example1_to_fpga.brd**. These two files are shown in Figure 13 and Figure 14.

```
[General]
Name = example1 XCV2000E -> fpga
Priority = 1

[ScanChain]
TAPs = 2
TAP0 = XCV2000E
TAP1 = XC9572XL

[Program]
SequenceLength = 1
Step1Method    = Virtex
Step1TAP       = 0
Step1File      = example1.bit
```

Figure 13. Example1_to_fpga.brd

```
[General]
Name = example1 XCV2000E -> flash (addr 0x0)
Priority = 1

[ScanChain]
TAPs = 2
TAP0 = XCV2000E
TAP1 = XC9572XL

[Program]
SequenceLength = 3
Step1Method     = Virtex
Step1TAP        = 0
Step1File                                                   =
lmxcv600e_72c_xcv2000e_via_reva_build0.bit
Step2Method     = IntelFlash
Step2TAP        = 0
Step2File       = example1.bit
Step3Method     = IntelFlashVerify
Step3TAP        = 0
```

Figure 14. Example1_to_flash0.brd

Modify the content of example1_to_fpga.brd and example1_to_ flash.brd as shown in Figure 13 and Figure 14. Remove **Step2Address = 200000** or **Step3Address = 200000** if possible. (#Address 0x200000 saves the test image of LM, avoid modifying image 1 at 0x200000)

2. Connect ARM MultiICE onto LM. (**Be SURE to power down first!! …$$**)

3. Set the LM in Config Mode by shorting the ***CFGLNK*** jumper on the LM board. The ***CFGLED*** on the LM is lit as an indication for configure mode. LM's FPGA can only be detected by MultiICE Server in configure mode. Yet CM cannot be found by MultiICE Server while LM is in configure mode.

4. ***Auto-config*** again in the MultiICE Server program. Remember to auto-configure again each time the MultiICE link is modified.

5. Execute **progcards.exe** to download the bitstream to the FPGA. This download program only searches for the **.brd** files in the same directory. If only one .brd file exists, the downloading would start directly without any prompt.

Running the Downloaded Bitstream from FPGA

1. Execute **progcards.exe** and select **example1 XCV2000E->fpga.**

2. This will take about 1 minute. It will automatically start running the

programmed bitstream right after it finishes downloading into the FPGA.

Running the Downloaded Bitstream from Flash(0x00)

1. Execute **progcards.exe** and select **example1 XCV2000E->flash(addr 0x00)**.

2. This will take about 3 minutes. ***Remove the CONFIG link*** after downloading.

3. Power down the LM.

4. Select the flash image to be executed. Which flash image to be executed is selected by the position of the ***4-way switch S1*** on the LM. It is only active in power-up blink. Consult ***Table 8*** and change the position of ***S1*** while in ***stand-by mode***. The circled positions are better than the crossed ones because of being independent of **CFGSEL[1:0]**.

Flash image	Image base address	CFGSEL[1:0]	S1[1]	S1[2]	S1[3]	S1[4]
0	0x000000	xx	CLOSED	x	OPEN	x
1	0x200000	xx	OPEN	x	OPEN	x
0	0x000000	0x	CLOSED	x	CLOSED	x
1	0x200000	1x	OPEN	x	CLOSED	x

Table 8. The relation between the 4-way switch positions and the selected flash image.

5. Power the LM up again and observe the LM. You will see the LEDs on the LM flashing from left to right. The combination of the switch ***S1*** can changed the flashing frequency.

A Timing Information Example

Figure 15 shows the distribution of each stage during the FPGA working flow. The proportion of total execution time of each stage is also shown in this figure. As can be seen, place-and-route using the batch scripts occupies 50% of the total execution time. Therefore we strongly suggest users to perform place-and-route on a faster PC, since it's the most time-consuming work.

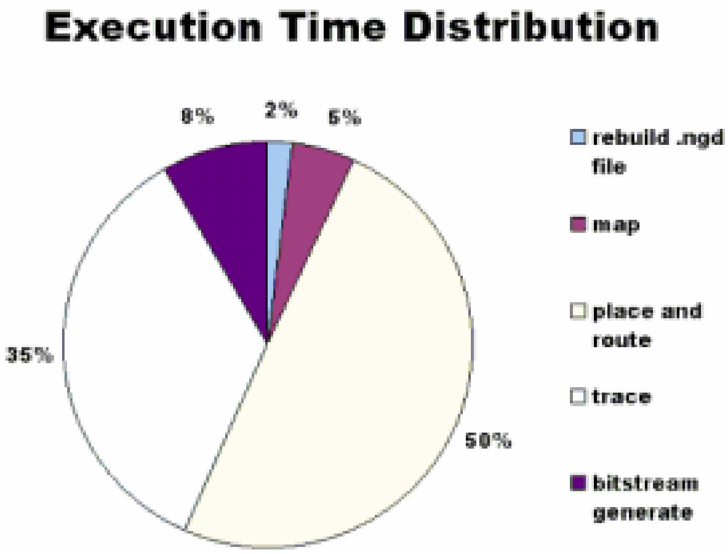

Figure 15. The execution time distribution of each stage in the FPGA working flow.

9.4. 實驗要求

Design an RGB-to-YUV converting hardware module that converts R, G, B values into Y, U, V values:

1. Implement the converter with pure software; you'll need to write the test program.
2. Implement the converter into hardware and program it into the FPGA on the LM, evaluate the improvement compared to pure software implementation.
   Hint: you may modify **AHBAHBTop.v**, **AHBDecoder.v**, **AHBMuxS2M.v**, and **AHBZBTram.v** in example 2.

Function:

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Figure 16. RGB to YUV transfer function.

9.5. 問題與討論

1. In example1, explain the differences between the flash version and the FPGA one.
2. In example1, explain how to move data from DRAM to registers in MYIP and how program access these registers.

3. In example2, draw the interconnect

9.6. 參考文件及網頁

- http://twins.ee.nctu.edu.tw/courses/ip_core_02/index.html
- http://twins.ee.nctu.edu.tw/courses/ip_core_01/index.html
- http://www.arm.com/
- Integrator ASIC Platform [DUI_0098B_AP_UG]
- System Memory Map [DUI_0098B_AP_UG 4.1]
- Counter/Timer [DUI_0098B_AP_UG 3.7, 4.6]
- Interrupt [DUI_0098B_AP_UG 3.6, 4.8]