Contents

ASIC Logic

1. Overview

This lab gives a brief introduction to digital IP authoring techniques. The ability of writing hardware description language is essential in this lab. In this lab, you will design your own IP with hardware description language. Next, the design will be checked by lint tools and coverage verification tool. Finally, the design is to be synthesized and downloaded to FPGA on ARM Logic Module for verification.

2. Background Information

2.1. Rapid Prototyping with Logic Module

The ARM Integrator/LM provides a platform for developing digital IPs on the AMBA-based SoC [1]. In this lab, the LM will work with the core module (CM) and the application platform (AP). The CM is the master device on the system bus while some slave devices reside in the LM. The AP serves as a communication channel between CM and LM. There are three other ways to use the LM. First, the logic module can also work in standalone mode like a traditional FPGA test board. In addition, the LM can also function as a CM on AP if a synthesized ARM core is programmed into the FPGA. The last option is to stack several LMs together without an AP if one of the LMs provides the system controller functions of a motherboard.

The LM contains several components to facilitate rapid prototyping (Figure 1). The primary component is a FPGA from Altera or Xilinx. A configuration PLD and a flash memory are presented to store the FPGA configurations. A ZBT SSRAM of 1MB 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 details.

[1] The Advanced Microcontroller Bus Architecture (AMBA) includes the Advanced High-performance Bus (AHB), the Advanced System Bus (ASB) and the Advanced Peripheral Bus (APB).

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.

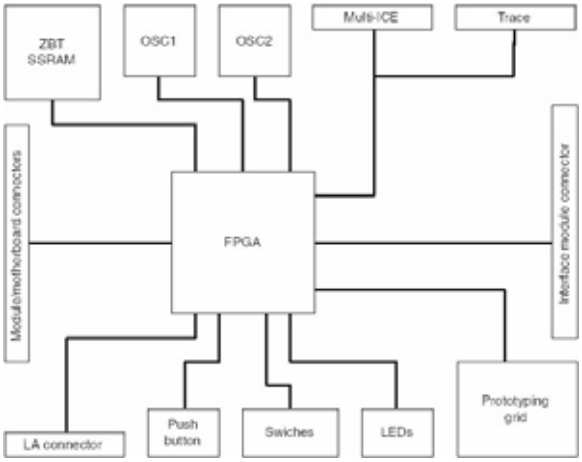


Figure 1. The architecture of a Logic Module.

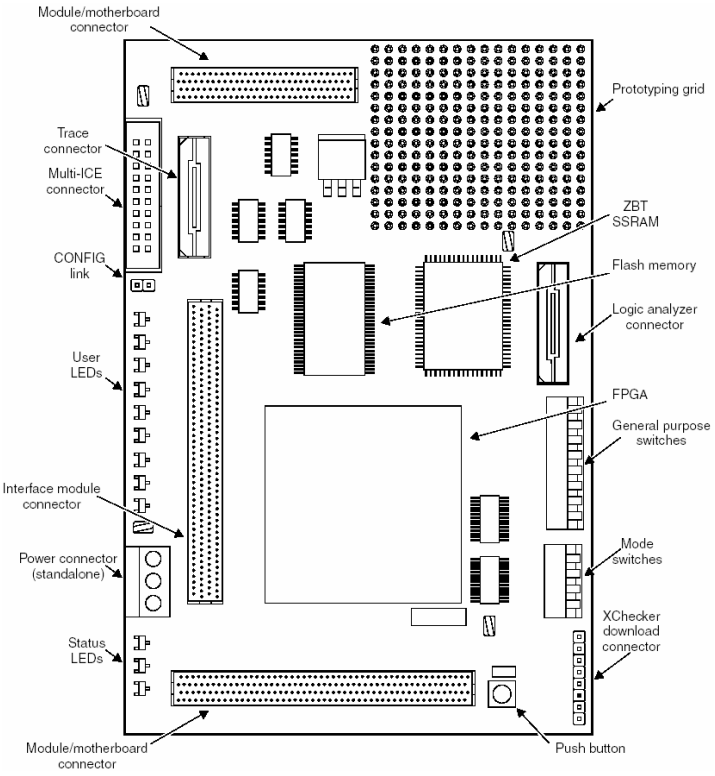


Figure 2. Logic Module's board layout.

2.2. 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 3 supports the first configuration, and Figure 4 supports the second one.

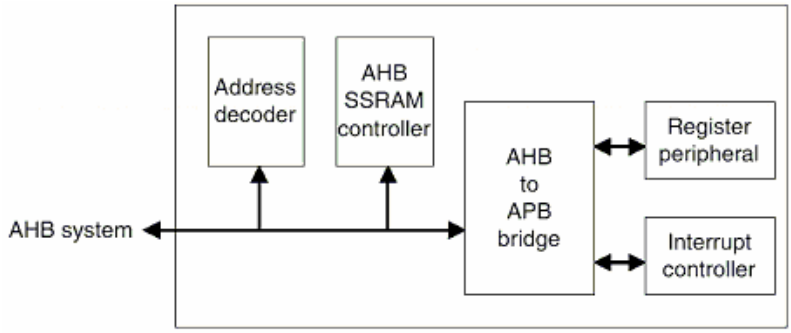


Figure 3. Implementation the support AHB system.

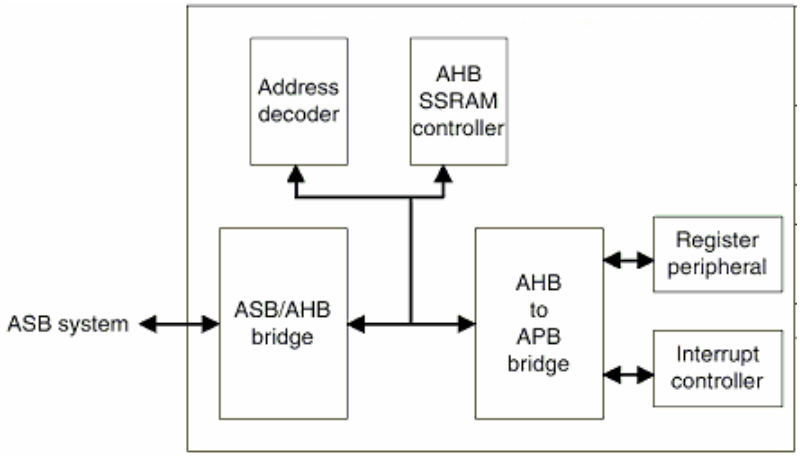


Figure 4. Implementation that supports ASB system.

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 5. The blocks inside the dashed bounding box represent the architecture to be programmed into the LM's FPGA.

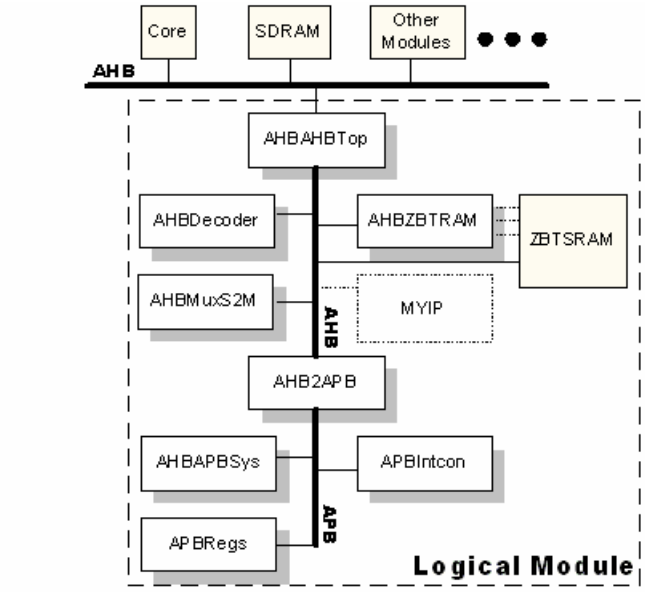


Figure 5. The AHB platform and its block diagram used in this course.

2.3. Logic Module Registers

The memory space within a LM and its relation with Integrator's system memory space is illustrated in Figure . 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. The Integrator's system memory map is shown in Figure 8.

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 push button

Table 2. Push button interrupt register.

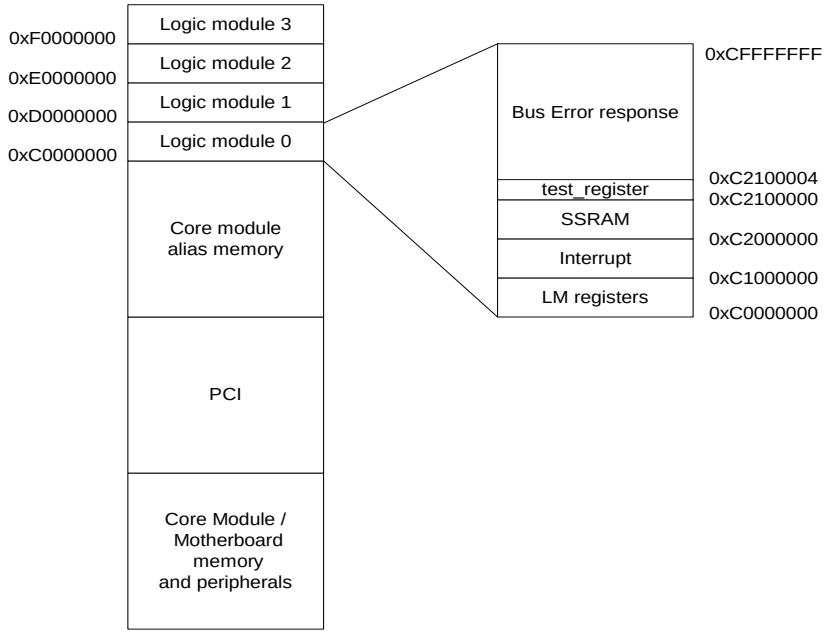


Figure 7. Relations between LM's memory space and the Integrator system's memory space

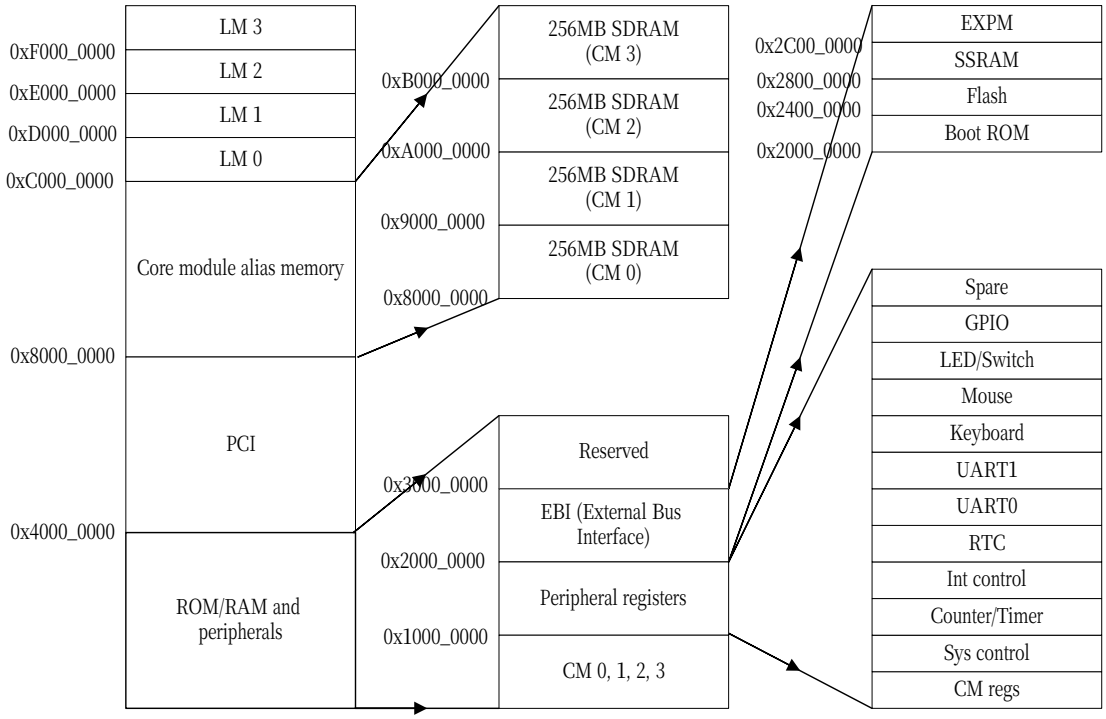


Figure 8. System memory map

2.4. 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 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.

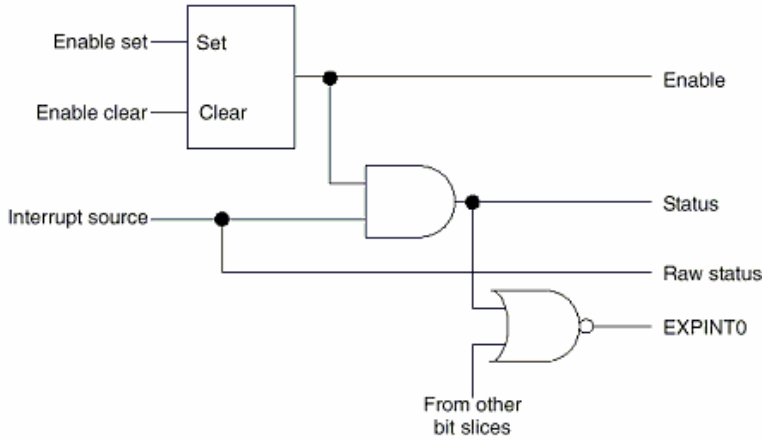


Figure 9. Bit-slice of LM's interrupt controller's structure.

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.

2.5 Rapid Prototyping Tools

Static Lint Checking

The Novas nLint is used for static lint checking. A lint tool can find errors and warnings in many aspects including naming, synthesis, simulation and DFT issues. Common syntax errors, such as typing errors, unmatched bus width, undeclared objects, can be quickly located. Some logical errors like unreachable state can be found. The lint tool also indicates bad coding styles that may lead to poor reusability. By using the lint tool, many un-necessary iterations of simulation can be saved.

Dynamic Coverage Verification

The Verification navigator from TransEDA is used for coverage verification. The coverage verification gives metrics on how well are the design being verified. For example, statement coverage shows the number of times each HDL statement is exercised. After extensive verification, un-exercised

statements are likely to be redundancies in the design. Similarly, a state coverage inspects if there exist any unreachable states in the FSM.

According to the Semiconductor Reuse Standards, the statement, branch and state coverage of a design should achieve 100%. However, the lower bound for trigger and condition coverage are not listed.

The coverage also suggests if the test vectors applied to the design under verification are sufficient. It's easier for the developers to decide when the simulation tasks could be done.

Implementation

The Xilinx ISE is a GUI implementation tool for Xilinx FPGA. This tool handles all the work from RTL to downloadable bit-stream. In addition, the Core Generator utility of ISE can create useful building blocks such as SRAM, ROM, and so on. The implementation flow can general be divided into 5 stages: synthesys, translate, map, place-and-route and bit-stream generation.

3. Instructions

In this lab, you have to work on the RGB2YUV module. First, lint the design with NOVAS nLint. Fix all possible errors and warnings. For the rest, explain the reasons why they are not fixed. Second, run the coverage verification with Verification Navigator. Report the statement coverage. If the coverage is not 100%, try to improve the coverage by either apply more test vectors, or remove redundant code from the RGB2YUV module. The last thing to do with the RGB2YUV module is to implement the design with Xilinx ISE. Run the implementation script and check the results.

3.1. Before You Start

You have to set up the files for this lab. A few steps to follow:

Login to the workstation in ED415.
Unpack the file "lab7_code.tgz" to your home directory. A directory named "lab7" should now appear in your home directory.

```
% tar xvfz lab7_code.tgz
% cd lab7
```

Directory Name	Descriptions
~/lab7/RGB2YUV/rtl	RTL source files for RGB2YUV module and the AHB bus wrapper
~/lab7/RGB2YUV/nlint	Working directory for nLint
~/lab7/integrator/hw	Bit-stream file for hardware module
~/lab7/integrator/sw	Software for controlling hardware module
~/lab7/integrator/configure	Utility for bit-stream downloading

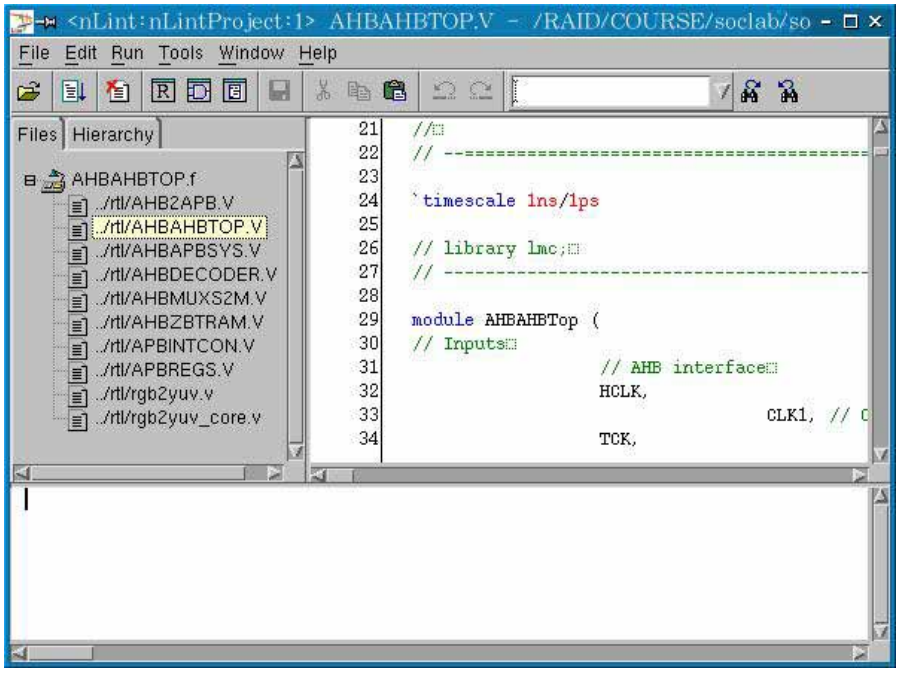
3.2. Lint Checking

To run lint checking on the RGB2YUV module, please follow these simple instructions.

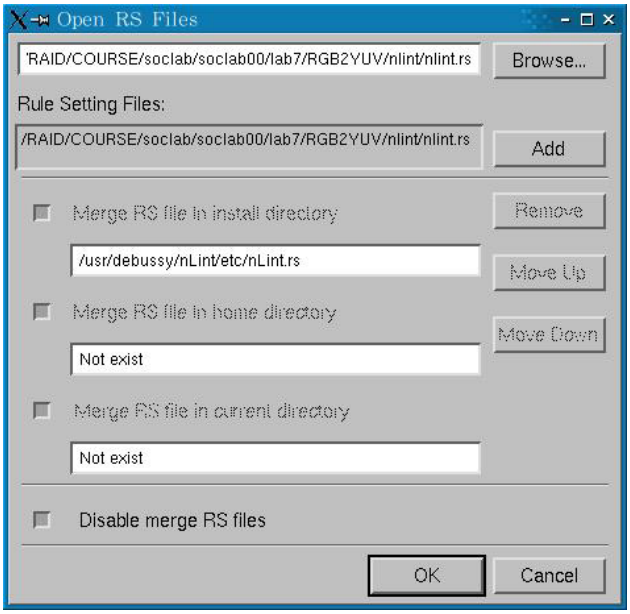
1. Switch to the working directory for nLint. Then, launch nLint with graphic user interface. The main window should show up right away.

```
% cd ~/lab7/RGB2YUV/nlint
% nLint –gui &
```

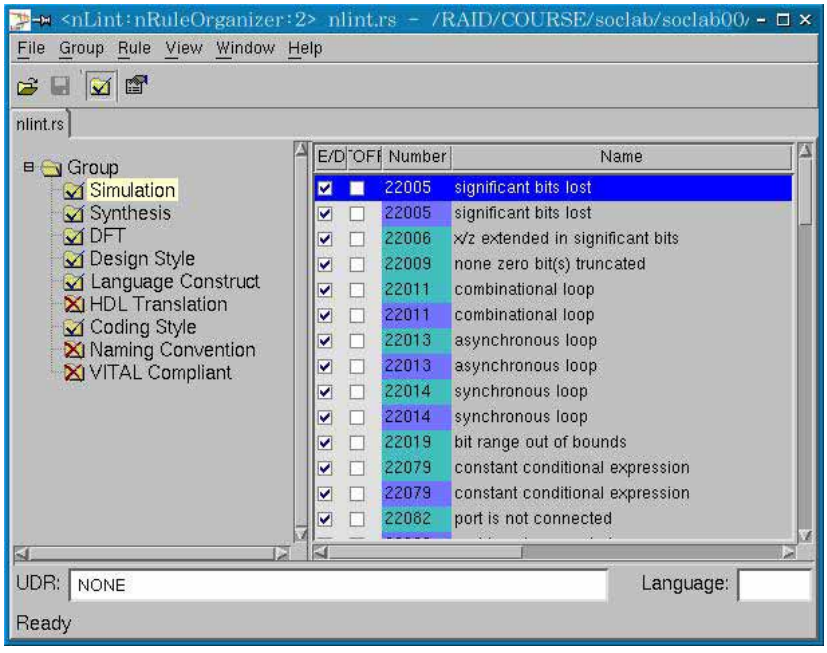
2. Load the RTL source files. Just select the from the menu bar: File => Import Design. A dialog pops up. In the import file dialog, select From File and choose AHBAHBTOp.f. Click the Add button and the click OK. The RTL source file will now be loaded.



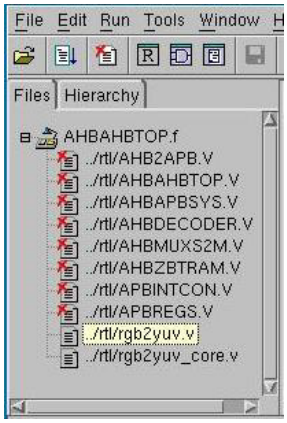
3. Now, we want to load the rule set files. Select from menu bar: Tools => Rule Organizer. Then, choose the rule file “~/lab7/RGB2YUV/nlint/nlint.rs” and click Add. You have to select the “Disable merge RS files” at the bottom of the dialog box.



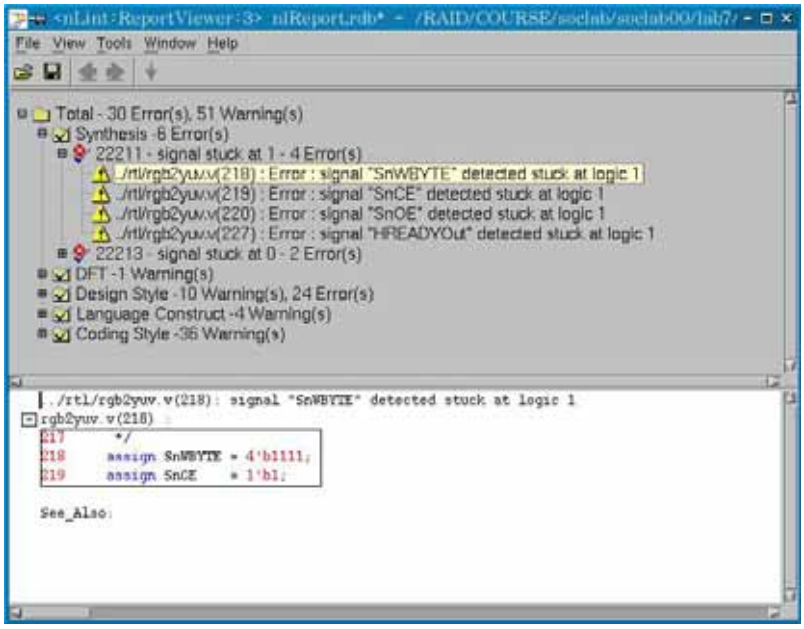
4. The rule set is now loaded. The rules are classified into several categories. You can modify the rules as you wish. To enable/disable a whole category, left click on the folder icon and toggle the “enable” option. To enable/disable an individual rule, toggle the check box under the “E/D” column. The “T/OFF” column next to the “E/D” column means rules are ignored for code segments within the “synopsys translate_off” directive.



5. Now, the nLint is ready to run lint checking on the RTL files. However, let’s focus on the rgb2yuv.v and rgb2yuv_core.v files first. Please select every file except the above two, and click the “Check/Uncheck” button from the toolbar for each file. A red cross will appear on the file marked uncheck.



6. Click the “Lint” button on the toolbar to start lint checking. A report window will soon appear.



7. Now, expand each category to see the violations. A double click on the violation brings you to the editor window to modify the problematic statement under concern. If you need any further explanations on the rule, left click on the rule and select “Search Rule”. A help window with explanation and example will show up.

8. Based on your knowledge on HDL design flow, please decide which violations are vital and which are not. Justify your decision.

9. Enable checking on all source files and re-run lint checking. Review all the violations.

10. Please fix all violations you consider vital. You are required to work on the “rgb2yuv.v” and “rgb2yuv_core.v”. Do not forget to save the modifications. Fixing other source files are not required but encouraged.

3.3. Downloading Hardware Modules to LM

This example demonstrates the basics to implement a design prototype by writing the FPGA into the flash on the Logic Module.

This example determines the DRAM size on the Core Module and sets up the system controller. It tests the SSRAM for word, half-word, and byte accesses. Then, it flashes the LED. Finally, it remains in a loop that displays the 8-way switch value of the Logic Modules on its LEDs.

Software File Descriptions

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

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.
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 5. The description for each software file

.

Downloading the Binary Bitstreams

1. Open the download setting files. (C:\lab7\integrator\configure\ lmxcv600e_72c_xcv2000e_example2_ahb_to_flash0.brd) The file is shown in Figure 69.

```
[General]
Name = example2 AHB XCV2000E -> flash 0 (addr 0x0)
Priority = 1

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

[Program]
SequenceLength = 3
Step1Method    = Virtex
Step1TAP       = 0
Step1File      = ..\hw\lmxcv600e_72c_xcv2000e_via_reva_build0.bit
Step2Method    = IntelFlash
Step2Address   = 0
Step2TAP       = 0
Step2File      = ..\hw\lmxcv600e_72c_xcv2000e_example2_ahb.bit
Step3Method    = IntelFlashVerify
Step3Address   = 0
Step3TAP       = 0
Step3File      = ..\hw\lmxcv600e_72c_xcv2000e_example2_ahb.bit
```

Figure 69. lmxcv600e_72c_xcv2000e_example2_ahb_to_flash0.brd

2. Connect ARM Multi-ICE 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. Remove the CONFIG link after downloading.

6. Power down the LM.

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

8. 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.

Working Hardware & Software Together

1. Connect the Multi-ICE onto CM (Be SURE to power off!!!)

2. Start Multi-ICE server program and press Auto-Configure.

3. Execute the CodeWarrior.

4. Choose Open from the tool panel (or press Ctrl + O).

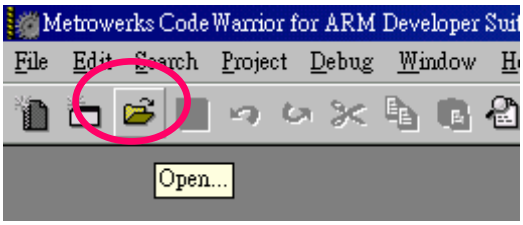


Figure 22. Open icon in the CodeWarriorIDE.

5. Choose C:\lab7\integrator\sw\sw.mcp

6. Make the project

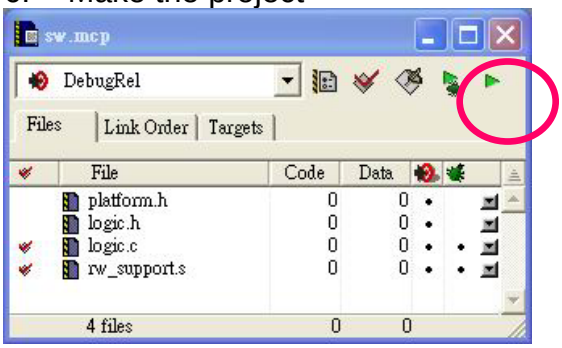


Figure 25. Press the Make button on the Project window.

7. After make completion, check the make report. 0 error and 1 warning is ok.

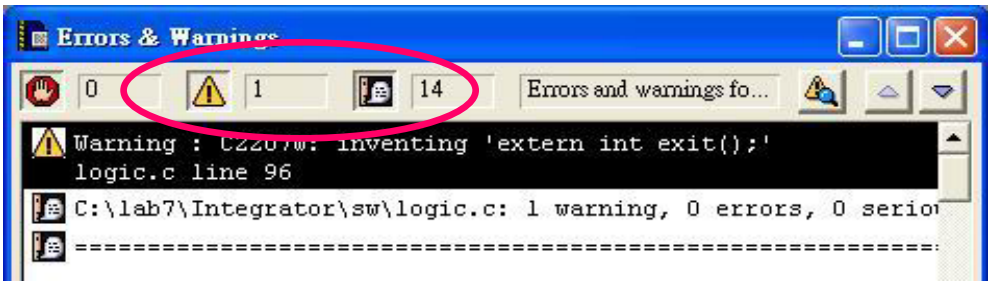


Figure 26. Errors & Warnings window after making the project.

8. Execute AXD (make sure all the equipments, including AP, CM, LM and Multi-ICE,etc; are all ready). If you encounter any error connecting to the server, click the configure button and select “Multi-ICE”.

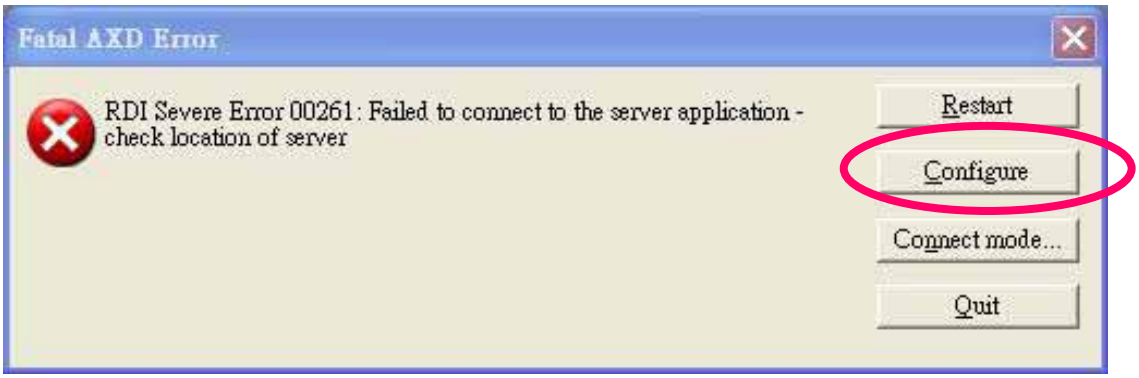


Figure 27. Click configure to select debug target

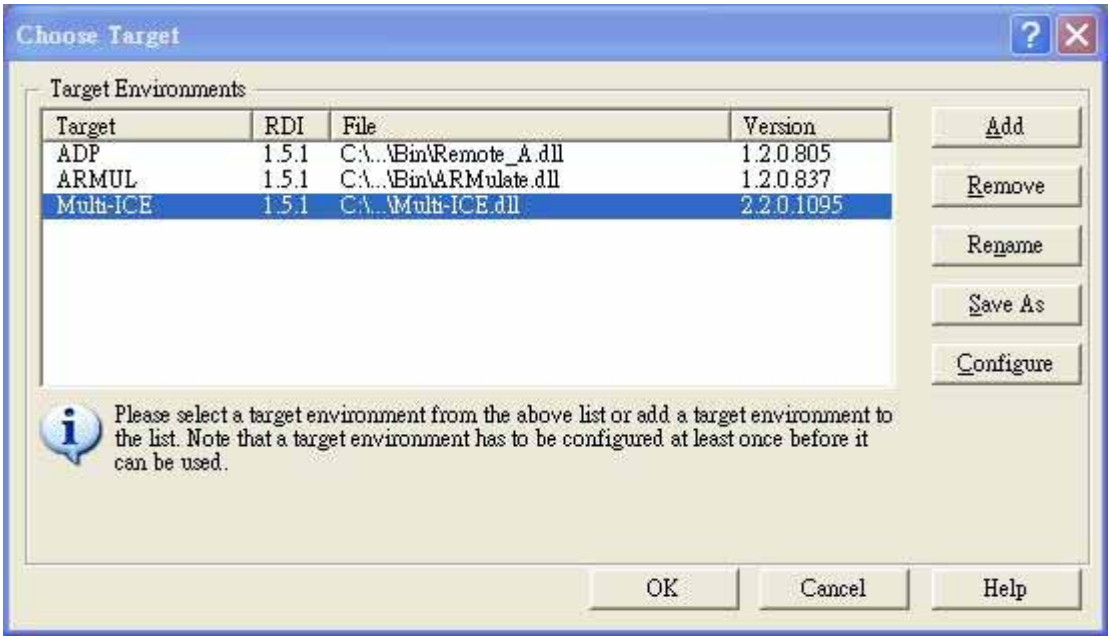


Figure 28. Select Multi-ICE as debug target

9. Go! (or press F5)

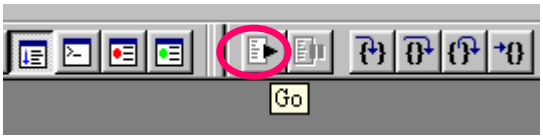


Figure 30. Press the Go button in the AXD window to run the image.

10. Another window pops up, program is halted at main function…

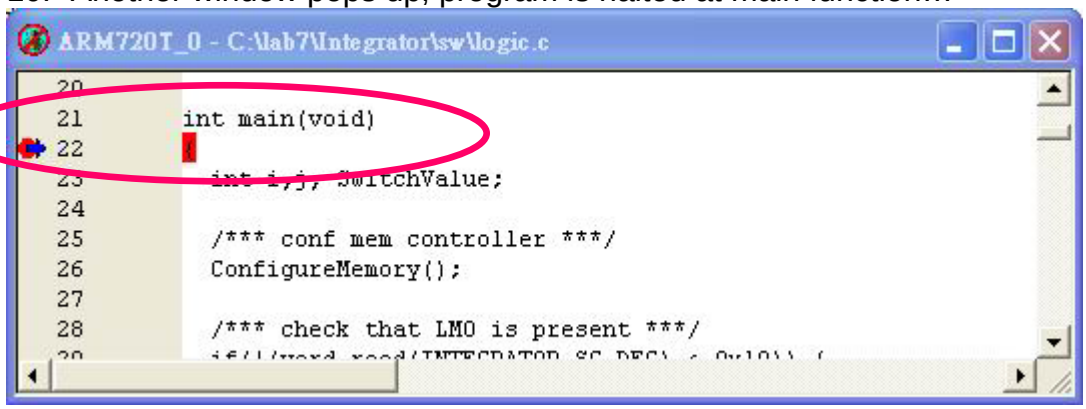


Figure 31. Source code window in AXD.

11. Observe the result on the Console window and AP board. If the example is running correctly, the AXD Console window will show the hardware information and the LM's SSRAM test results as shown in Figure 32. After the tests are done, the LEDs on LM would be flashed for several seconds, and then the LEDs will be mapped to the 8-way switch value.

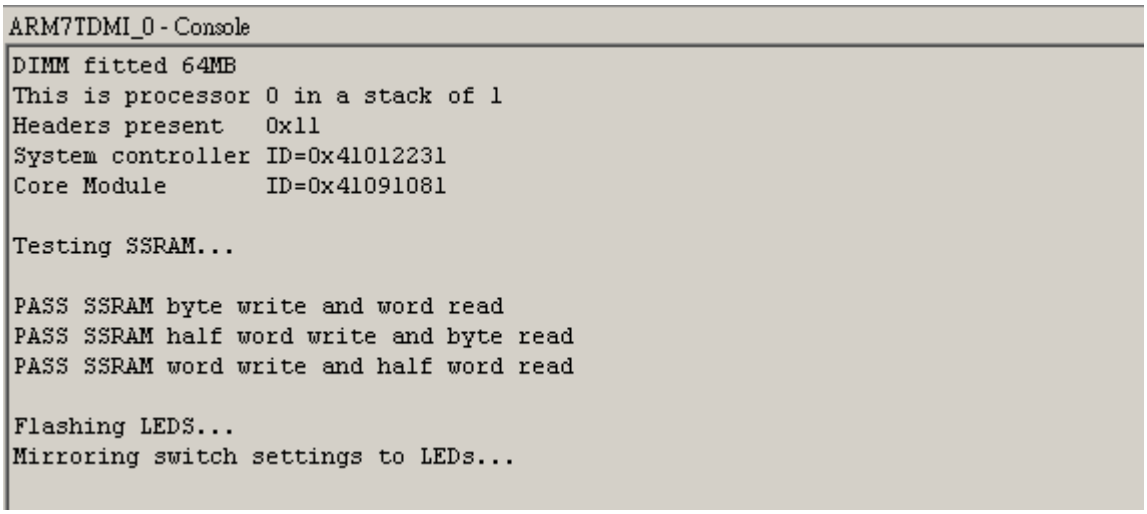


Figure 32 Correct results running the example with hardware.

4. References

1. http://twins.ee.nctu.edu.tw/courses/ip_core_02/index.html
2. http://twins.ee.nctu.edu.tw/courses/ip_core_01/index.html
3. http://www.arm.com/
4. Integrator ASIC Platform [DUI_0098B_AP_UG]

5. System Memory Map [DUI_0098B_AP_UG 4.1]
6. Counter/Timer [DUI_0098B_AP_UG 3.7, 4.6]
7. Interrupt [DUI_0098B_AP_UG 3.6, 4.8]