

Lab 4- IP Core Design

C. W. Jen 任建葳

cwjen@twins.ee.nctu.edu.tw

Lab TA: Tzung-Shian Yang

VLSI Signal Processing Group

Department of Electronics Engineering

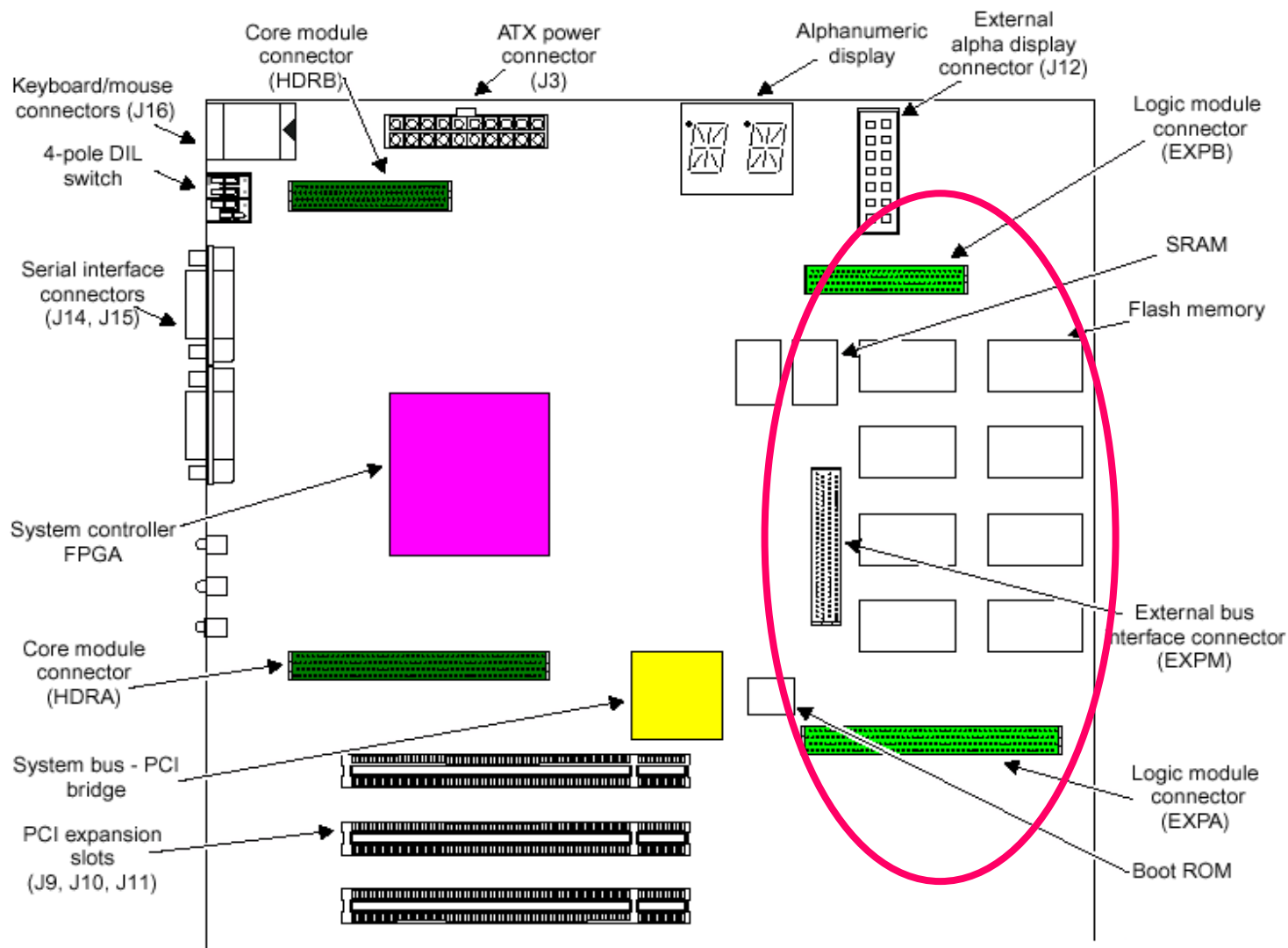
National Chiao Tung University

Outline



- Introduction of LM-XCV600E+
- FPGA tools
- Example 1
- Example 2
- Exercise

AP Layout



AP System Architecture

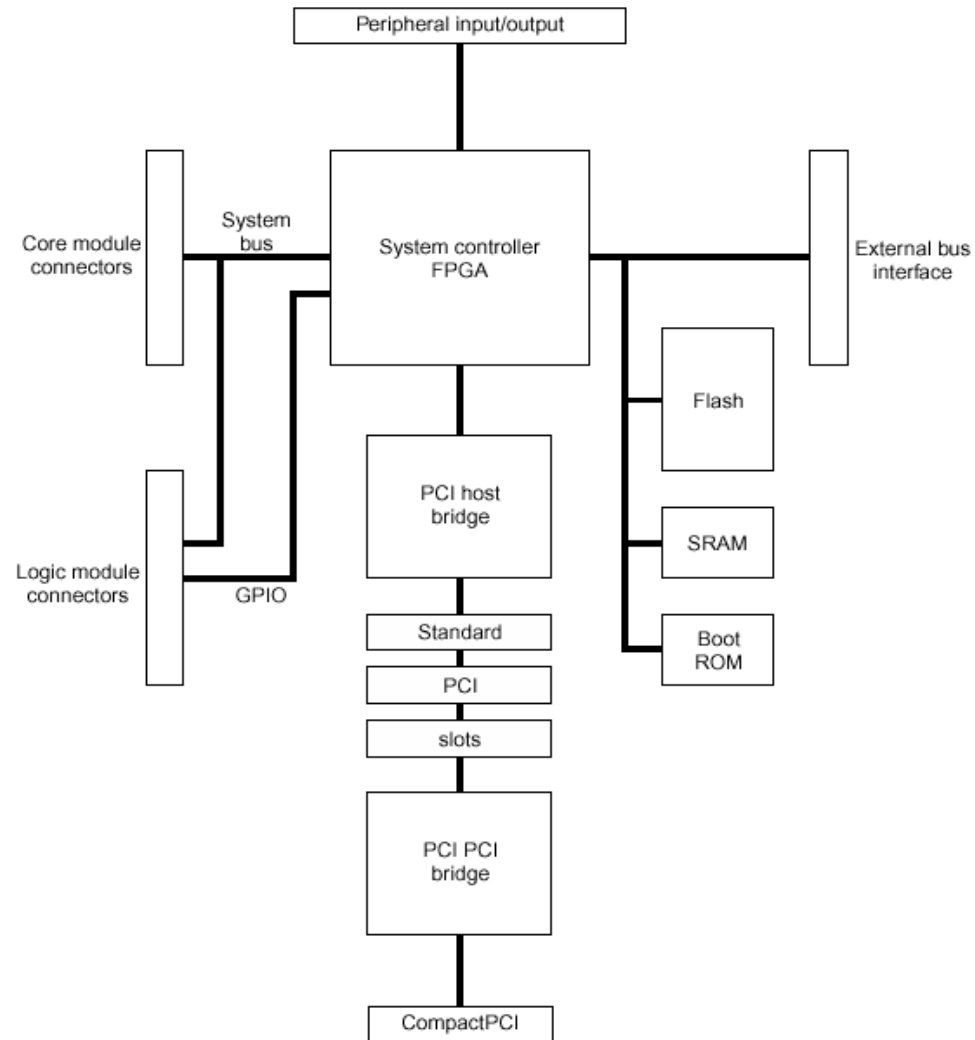


Figure 1-2 ARM Integrator/AP block diagram

What is LM



- *Logic Module*
- A platform for developing **Advanced Microcontroller Bus Architecture** (AMBA), **Advanced System Bus**(ASB), **Advanced High-performance Bus**(AHB), and **Advanced Peripheral Bus**(APB) peripherals for use with ARM cores.

Use the LM



- It can be used in the following ways:
 - As a standalone system
 - With an CM, and a AP or SP motherboard
 - As a CM with either AP or SP motherboard if a synthesized ARM core is programmed into the FPGA
 - Stacked without a motherboard, if one module in the stack provides system controller functions of a motherboard

LM Architecture

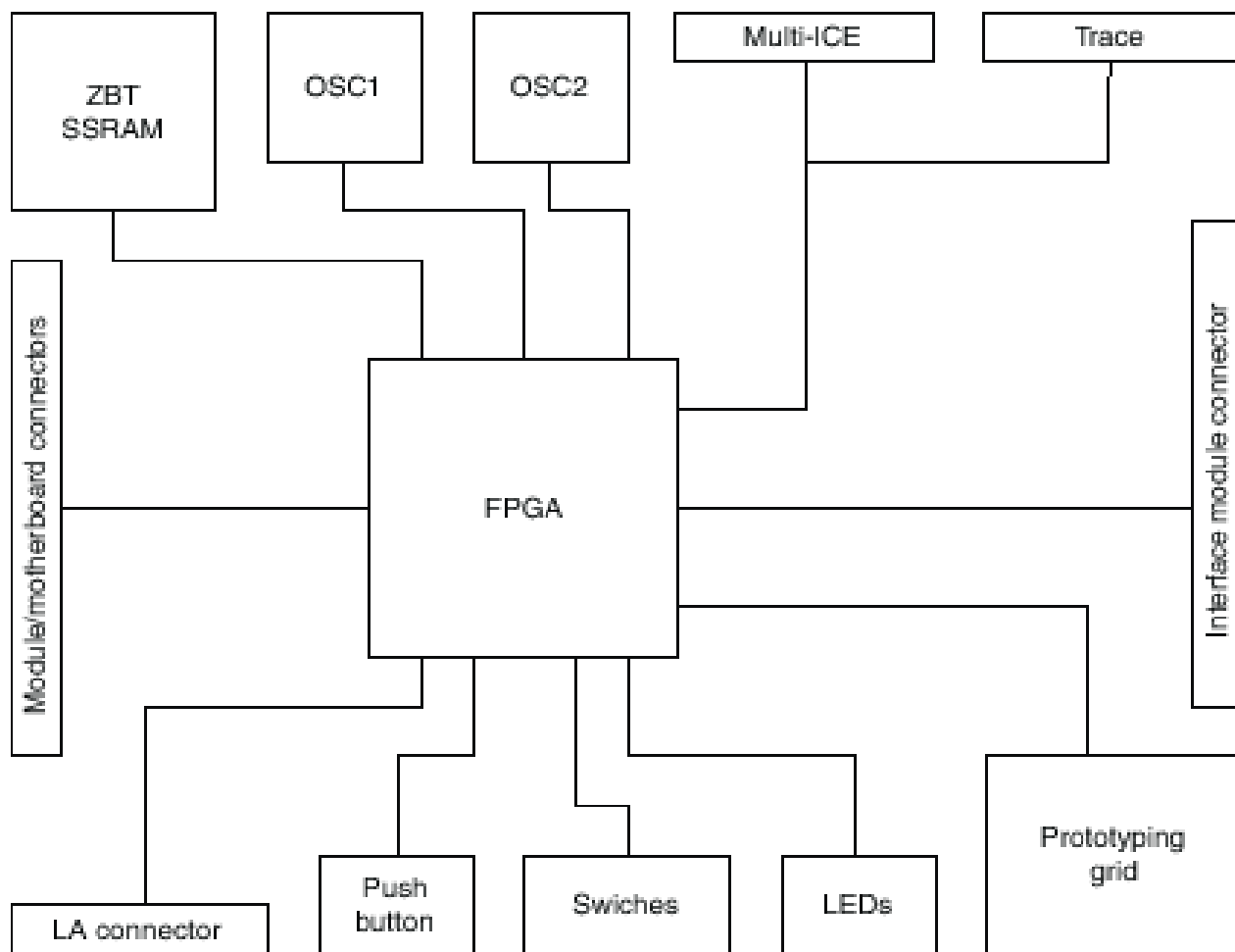


Figure 1-3 System architecture

Components of LM



- Altera or **Xilinx** FPGA
- Configuration PLD and flash memory for storing FPGA configurations
- 1MB ZBT SSRAM
- Clock generators and reset sources
- 4-way mode switch and 8-way user definable switch
- 9 user-definable surface-mounted LEDs (8G1R)
- User-definable push button
- Prototyping grid
- System bus connectors to a motherboard or other modules

LM Layout

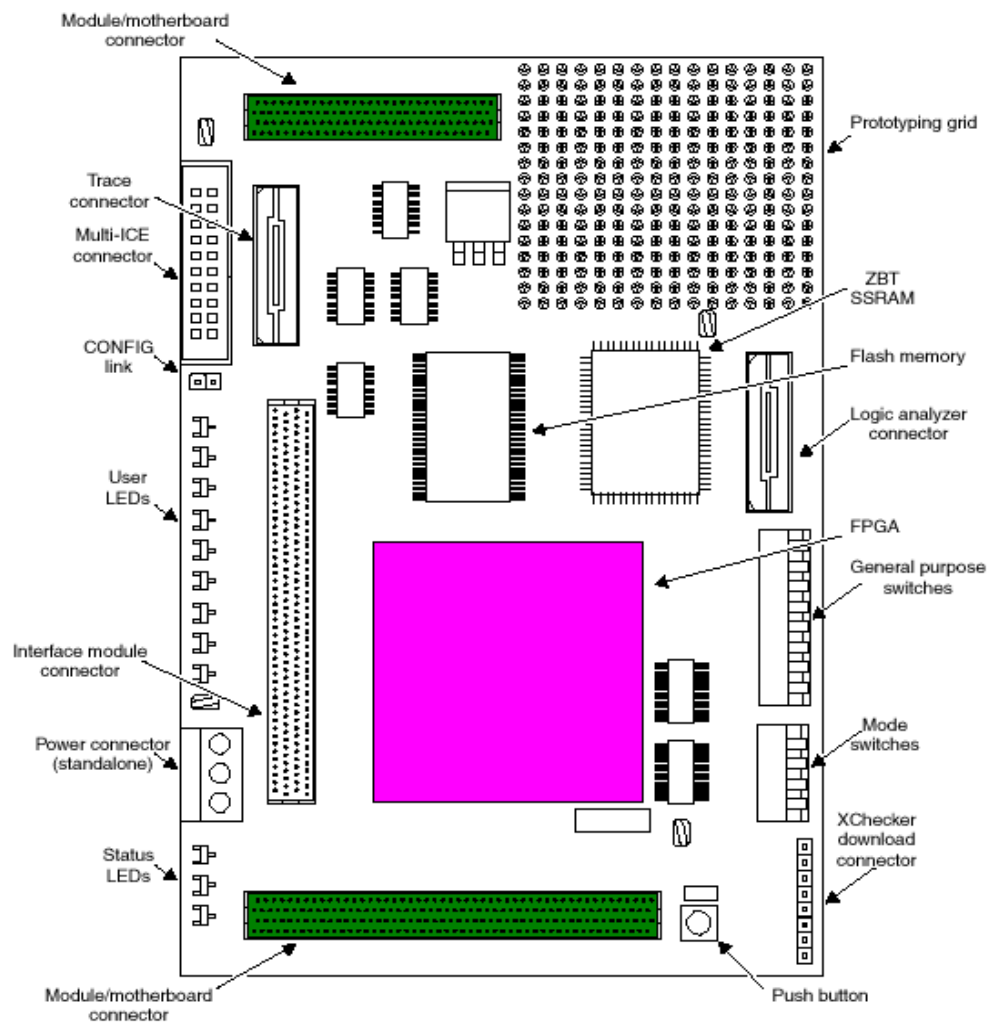
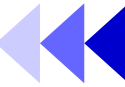
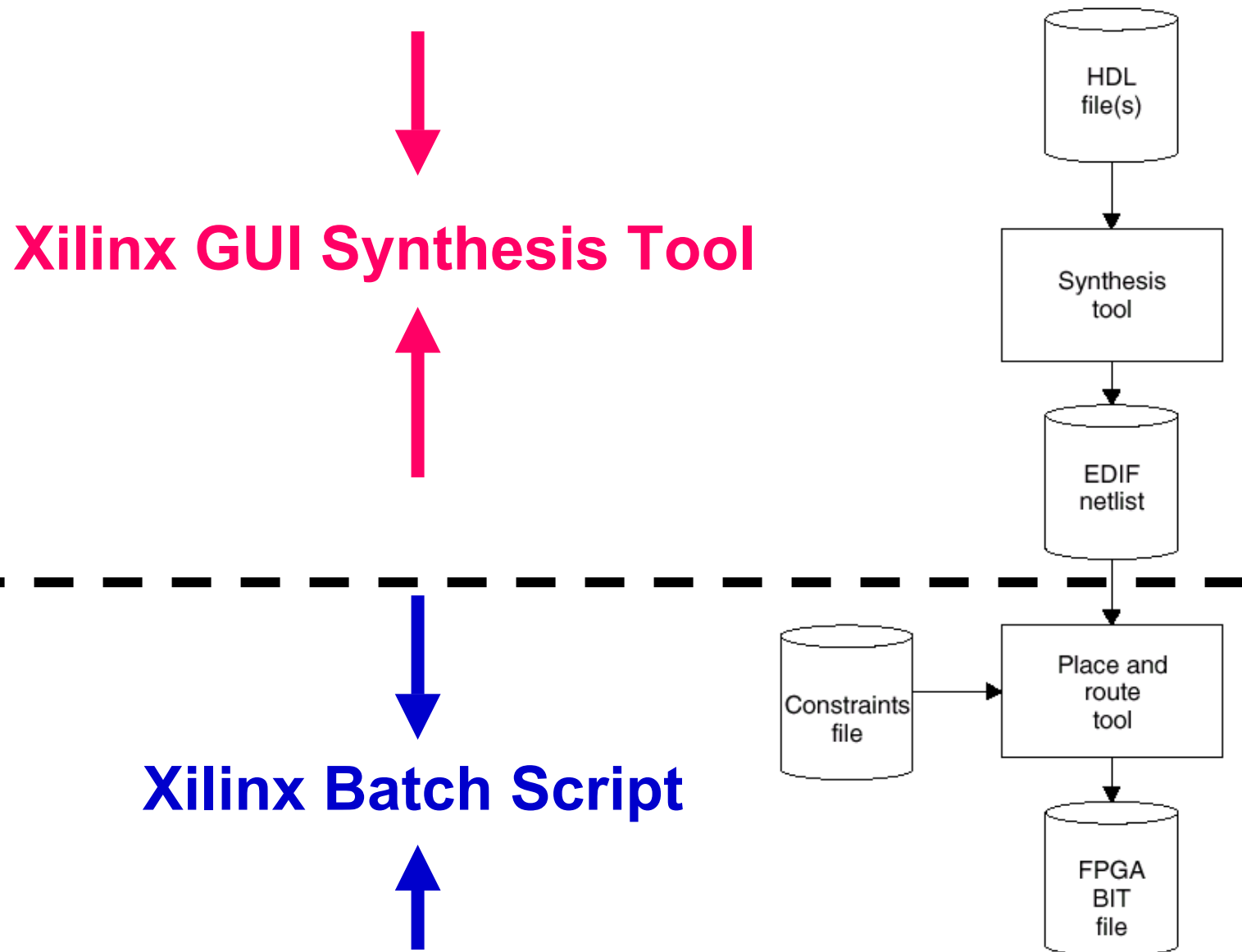


Figure 1-1 Integrator/LM-XCV600E+ layout



- CONFIG link
 - Enable ***configuration mode***, which changes the JTAG signal routing and is used to ***download new PLD or FPGA configurations***.
- JTAG, Trace, and logic analyzer connectors
- Other links, switches, and small ICs can be added to the prototyping grid if required.

FPGA tools



Xilinx GUI Synthesis Tool (1/3)



0. Extract lab4.zip to c:\ and c:\ipcore will be created
Extract v2000e.zip to %xilinx%\virtexe\data\
1. Execute the *Project Manager* of *Xilinx Foundation*
2. Create a New Project
 - Input Name (assume: **example1**)
 - Flow: *HDL*
3. Project -> add source File(s) (**example1\example1.v**)
 - Analyzing...
4. Synthesis -> synthesize..
 - Top level
 - Target device
 - Family: *VirtexE*
 - Device: *v2000efg680*
 - Speed: -6
 - Run and create **example1.edf**

Xilinx Batch Script (2/3)



1. cd c:\ipcore\lab4\example1\
2. copy %xilinx%\active\projects\example1\
example1.edf
3. replace all '<>' with '()' of example1.edf by
UltraEdit or other editor else
4. replace the '*filename*' of pc_par.bat with
example1
5. execute pc_par.bat to generate example1.bit

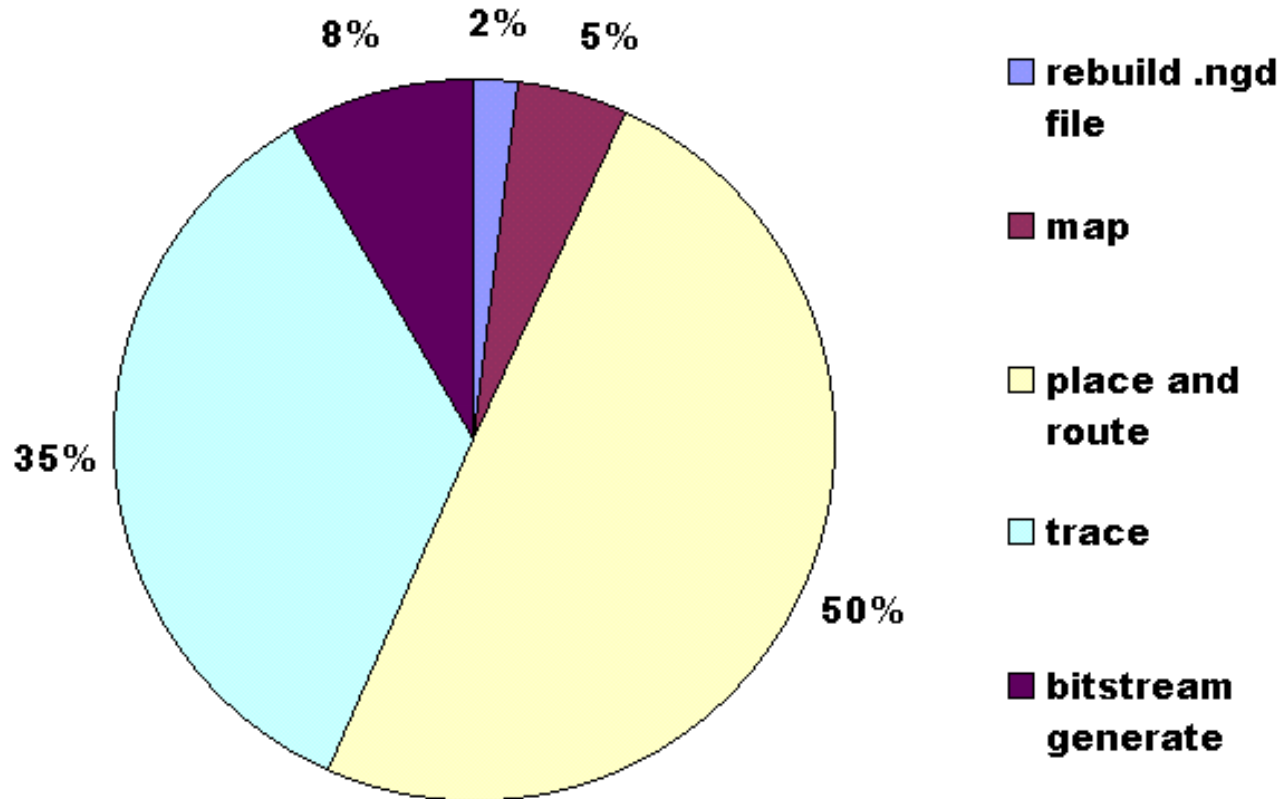
Download the Bitstream (3/3)



- Modify the ex1.brd
 - Step1File = example1.bit
 - Remove **Step2Address = 200000** if possible
 - Remove **Step3Address = 200000** if possible
 - # Addr. 0x200000 saves test image of LM (p.22)
 - Avoid to modify image 1 in 0x200000 !!
- Execute *progcards.exe*
 - Only search the .brd file in the same directory
 - If only one .brd file exists, the download work would be achieved directly

A Timing Information Example

Execution Time Distribution



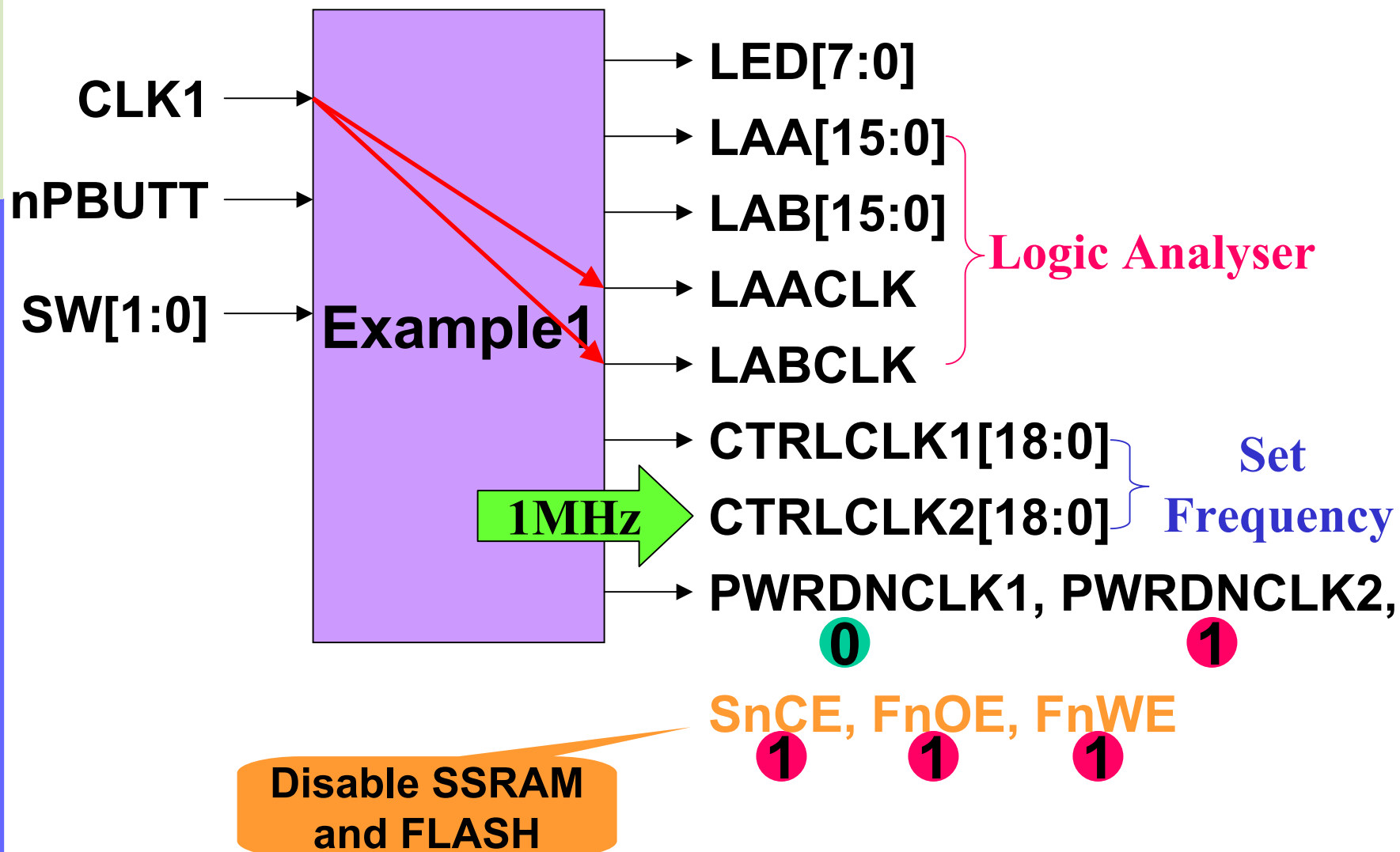
We strongly suggest you to perform place-and-route operation on a better PC, it's a very time-consuming work!

Example 1

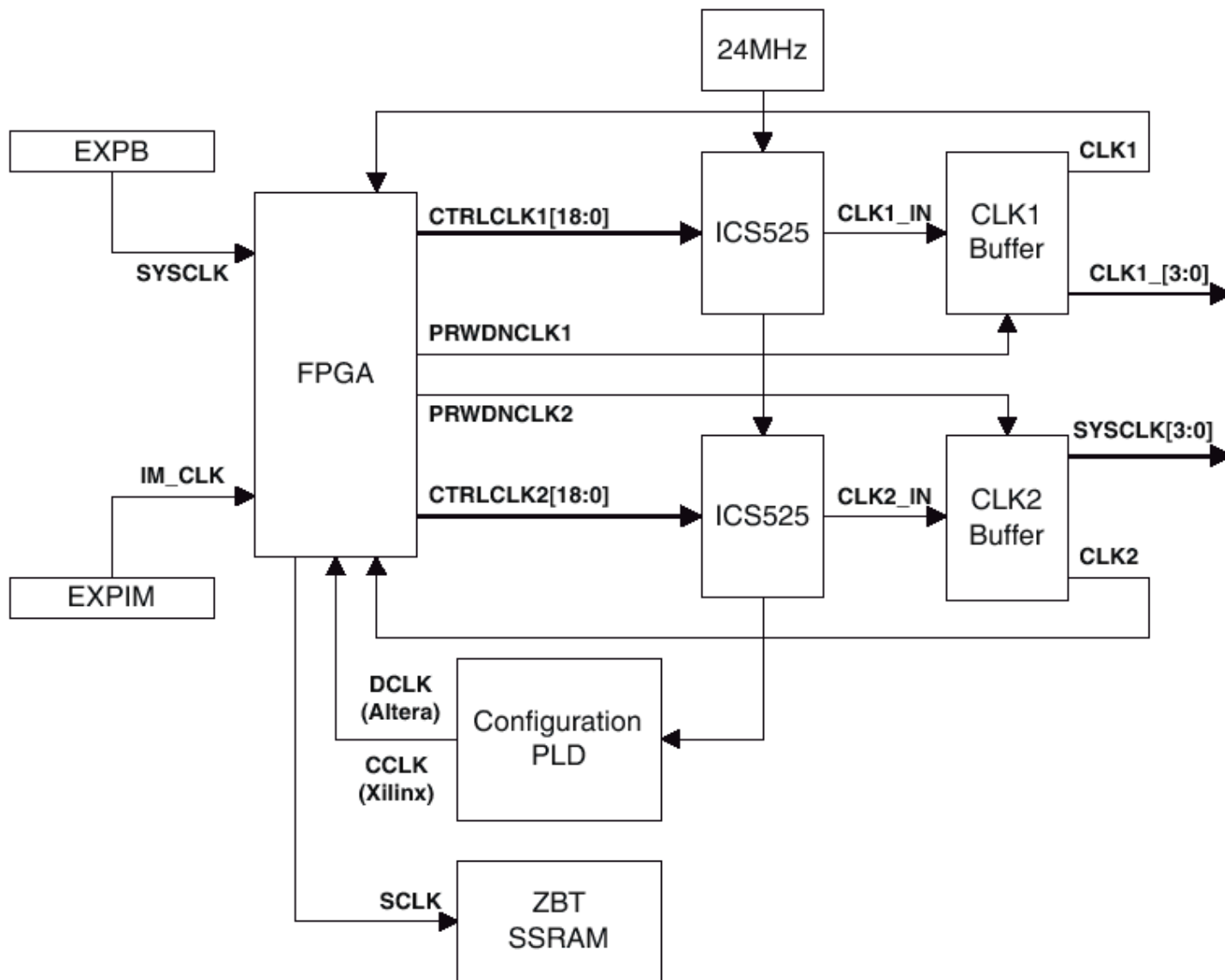


- C:\ipcore\lab4\example1\
 - Count up on logic analyzer channel a
 - Count down on logic analyzer channel b
 - Reset by pushbutton
 - Switches [0:1] (brown:red) control the clock frequency (***CTRLCLK1***)

Example 1 (cont.)



On-board Clock Generators



Clock Signal Summary



Clock name	Clock source
SYSCLK	Motherboard system clock
CLK1	On-board clock generator (programmable)
CLK2	On-board clock generator (programmable)
IM_CLK	Clock supplied from an interface module
CCLK (Xilinx) DCLK (Altera)	Configuration clock supplied by the PLD to the FPGA during FPGA configuration
SCLK	This signal provides a clock signal to the ZBT SSRAM
PWRDNCLK1	This signal can be used to enable or disable the CLK1[3:0] and CLK1 outputs
PWRDNCLK2	This signal can be used to enable or disable the SYSCLK[3:0] outputs to HDRB

Programming the Clock



Signals	Control parameter	Label
CTRLCLKx[18:16]	Output divider	S[2:0]
CTRLCLKx[15:9]	Reference divider	R[6:0]
CTRLCLKx[8:0]	VCO divider	V[8:0]

S	S[2:0]	Frequency = 48MHz · $\frac{(V[8:0] + 8)}{(R[6:0] + 2) \cdot S}$
2	001	
4	011	
5	100	
6	111	
7	101	
8	010	
9	110	
10	000	

1MHz: CTRLCLKx=19'b1100111110000000100
2MHz: CTRLCLKx=19'b1100011110000000100
5MHz: CTRLCLKx=19'b1100001110000000111
10MHz: CTRLCLKx=19'b1100000110000000111

Constraint:

$$10\text{MHz} < 48\text{MHz} \cdot \frac{(V[8:0] + 8)}{(R[6:0] + 2)}$$

$$R[6:0] < 118$$

Execution of Example 1



- Remember to connect the Multi-ICE onto **LM**
(Be SURE under the stand-by mode!!)...\$
- Set the LM in **Config mode** by fitting the CONFIG link, and the CFGLED is lit as an indication that configure mode is selected.
- C:\Program Files\ARM\Logic Modules\LM-XCV600E\configure\progcards.exe
(need Multi-ICE software, reading *.brd file)
 - Step 1: 3. example1 XCV2000E -> fpga
about 1 minutes, PNP Version, Simple
 - Step 2: 2. example1 XCV2000E -> flash(addr 0x0)
about 3 minutes, be patient...zzz

Execution of Example 1 (cont.)



- Load the FPGA from FLASH:
 1. Remove the CONFIG link
 2. Power the LM down (***stand-by button***)
 3. Power the LM up again (***stand-by button***)
(***Flash image x?***)
- Observe the result on the LM

Only active on power up

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

Example 2

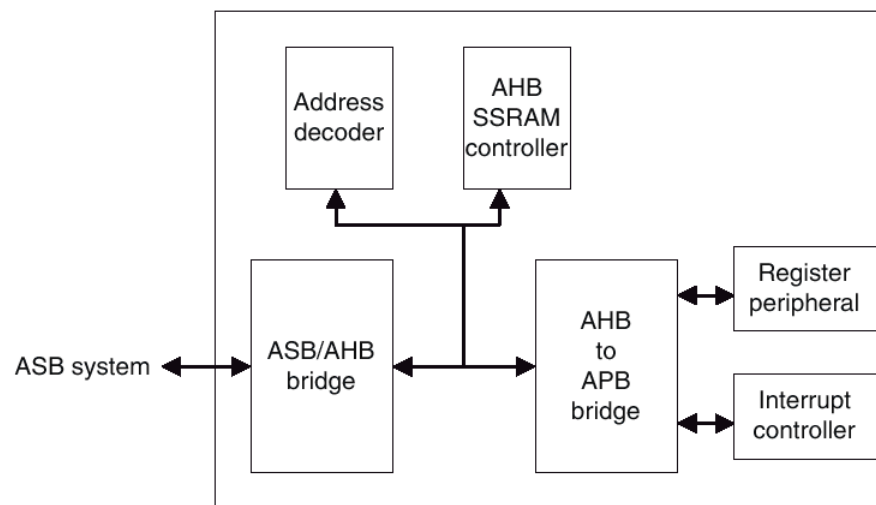
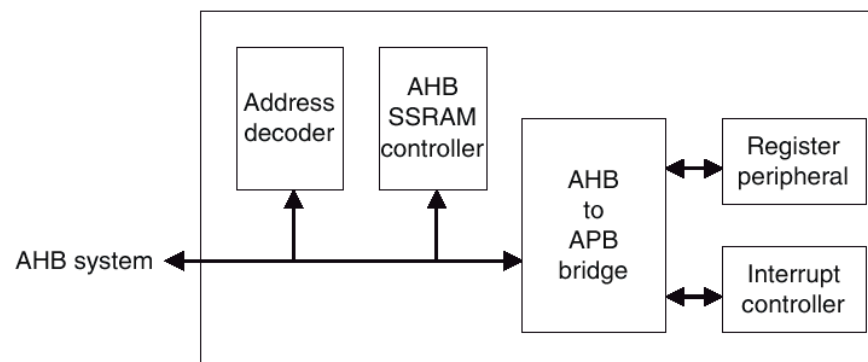


- The example code operates as follows:
 1. Determines DRAM size on the core module and sets up the system controller
 2. Checks that the logic module is present in the AP expansion position
 3. Reports module information
 4. Sets the logic module clock frequencies
 5. Tests SSRAM for word, halfword, and byte accesses.
 6. Flashes the LEDs
 7. Remains in a loop that displays the switch value on the LEDs

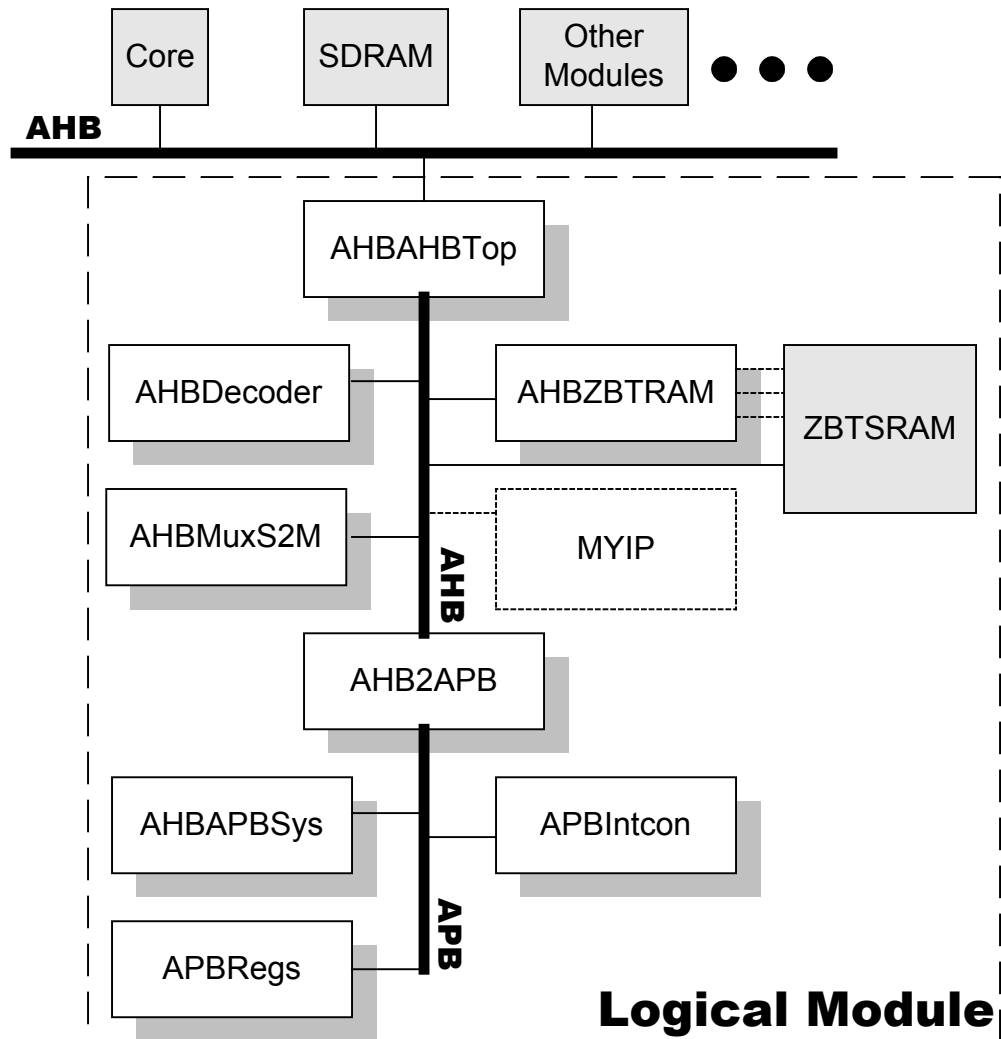
Two Platform – AHB & ASB



- Two versions of ex2 are provided to support the following implementations:
 - AHB MB and AHB peripherals
 - ASB MB and AHB peripherals
- Which AMBA has been downloaded on board can be observed by the alphanumeric display
 - **H**: AHB
 - **S**: ASB



AHB Platform



HDL Files Descriptions



File	Description
ASBAHBT AHBAHBT	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, ASBAHBT.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 (see <i>Example 2 memory map</i> on page 6-6).
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.

HDL Files Descriptions (cont.)



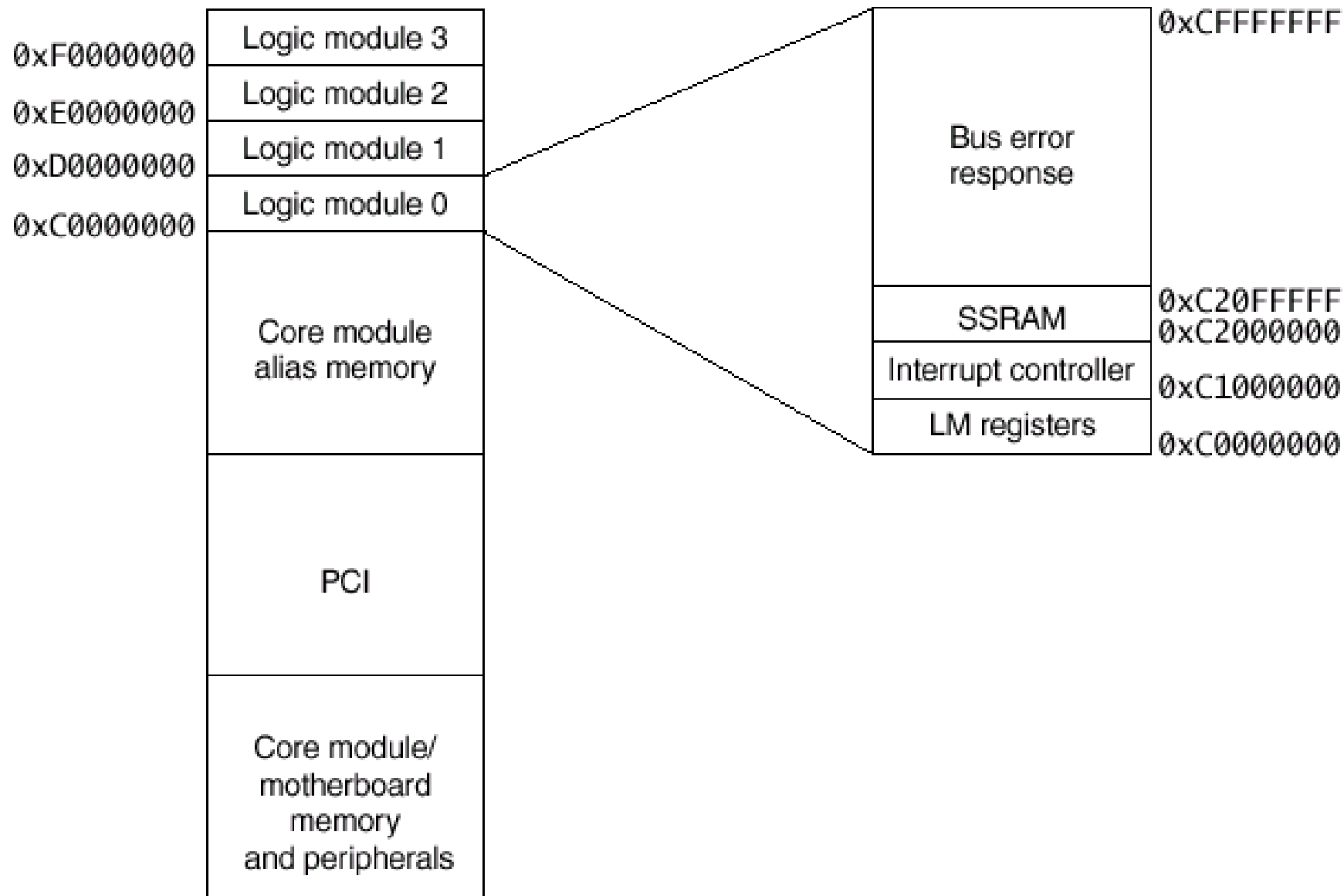
File	Description
AHB2APB	This is the bridge blocks required to connect APB peripherals to the high-speed AMBA AHB bus. They produce the peripheral select signals for each of the APB peripherals.
AHBAPBSys	The components required for an APB system are instantiated in this block. These include the bridge 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	<p>The APB register peripheral provides memory-mapped registers that you can use to:</p> <ul style="list-style-type: none">• configure the two clock generators (protected by the LM_LOCK register)• write to the user LEDs• read the user switch inputs. <p>It also latches the pressing of the push button to generate an expansion interrupt.</p>
APBIntcon	The APB interrupt controller contains all of the standard interrupt controller registers and has an input port for four APB interrupts. (The example only uses one of them. The remaining three are set inactive in the AHBAPBSys block.) Four software interrupts are implemented.

Software Description



- There are 4 source files included in Ex2
 - logic.c : the main C code
 - logic.h : constants
 - platform.h : constants
 - rw_support.s : assembly funcs for SSRAM testing

Integrator Memory Map



Logic Module Registers



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

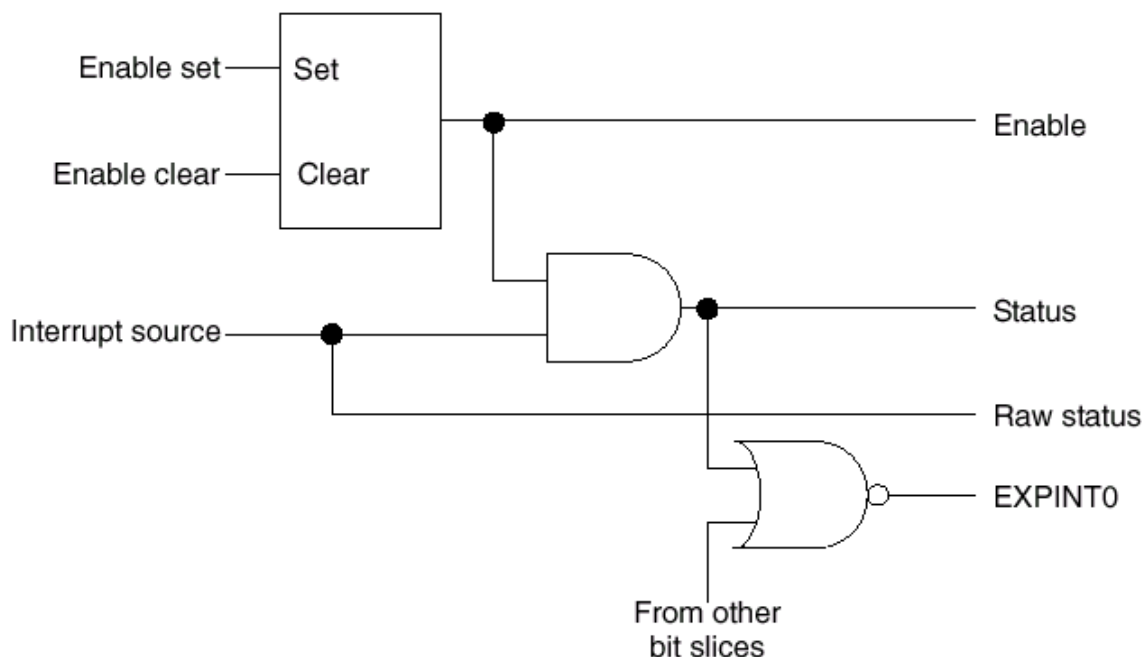
- The oscillator registers control the frequency of the clocks generated by the two clock generators. (p.18)
- Before writing to the oscillator registers, you must unlock them by writing the value *0x0000A05F* to the ***LM_LOCK*** register. After writing the oscillator register, relock them by writing any value other than *0x0000A05F* to the ***LM_LOCK*** register

Push Button Interrupt Register



Bits	Name	Access	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 the same effect as pressing the push button.

Interrupt controller



Register name	Address offset	Access	Size	Description
LM_ISTAT	0x1000000	Read	8 bits	Interrupt status register
LM_IRSTAT	0x1000004	Read	8 bits	Interrupt raw status register
LM_IENSET	0x1000008	Read/write	8 bits	Interrupt enable set
LM_IENCLR	0x100000C	Write	8 bits	Interrupt enable clear
LM_SOFTINT	0x1000010	Write	4 bits	Software interrupt register

Execution of Example2



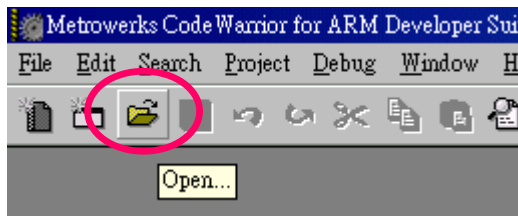
- Remember to connect the Multi-ICE onto **CM** (**Be SURE under the stand-by mode!!**)...\$\$
- Execute c:\ipcore\lab4\example2\runme.bat at first time. It would copy the new .brd to the proper location.
- Check the **LM** and be sure that flash image of **AHB** (*example2 AHB XCV2000E -> flash addr 0x0*), **not addr 0x200000!!** you can refer to the steps on P. 21~22) has been configured and the switch has been on the right position to select the respective FLASH image.
- Execute the **Code Warrior**



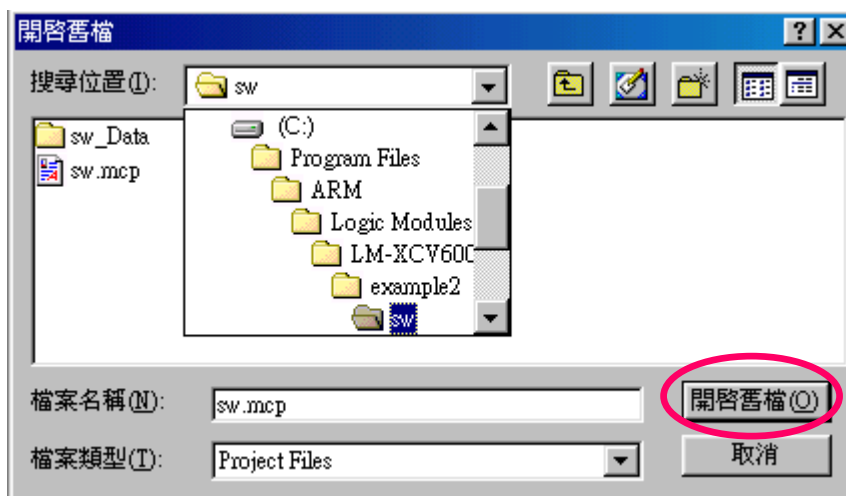
Execution of Example2 (cont.)



- Choose Open option (or press Ctrl-O)



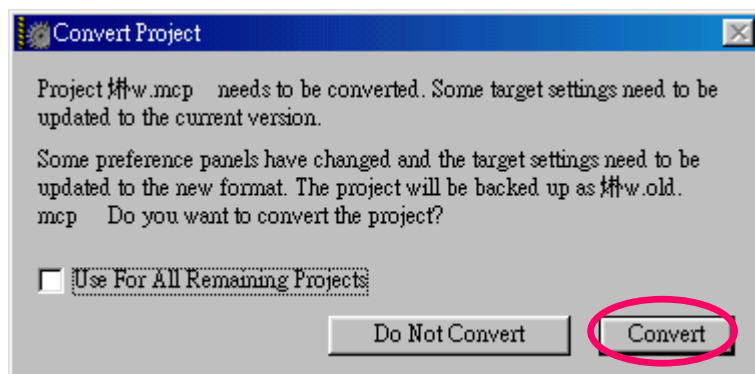
- Choose ***C:\Program Files\ARM\Logic + Modules\LM-CV600E\example2\sw\sw.mcp***



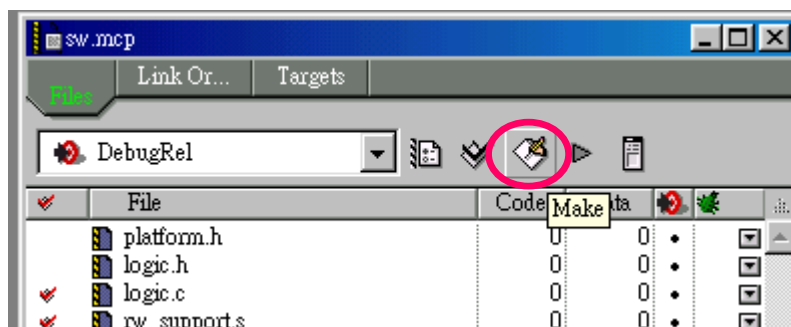
Execution of Example2 (cont.)



- Convert project



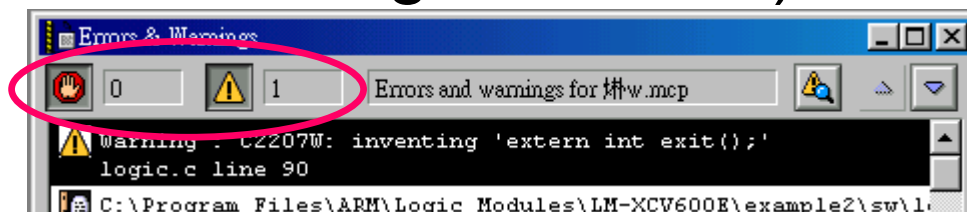
- Make file



Execution of Example2 (cont.)



- Make completion and check reports (0 error and 1 warning, that's ok!)



- Execute **AXD** (to be sure the equipments, including AP, CM, LM and Multi-ICE..etc, are all ready)



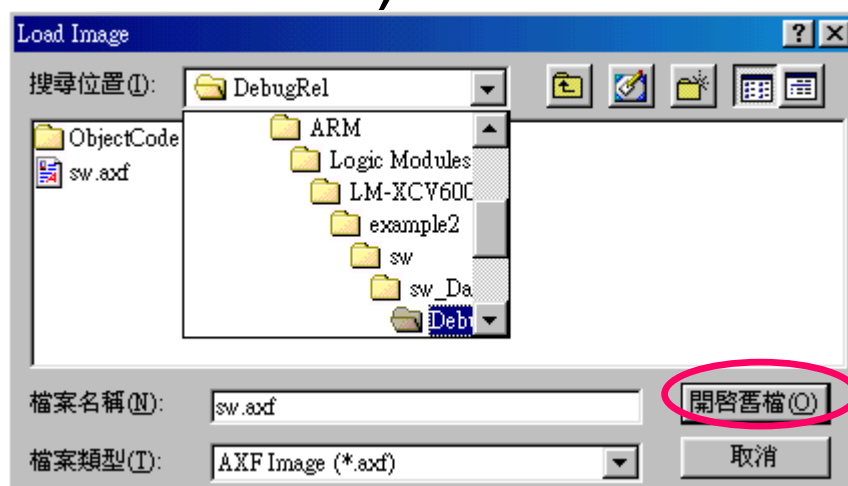
Execution of Example2 (cont.)



- Load image..



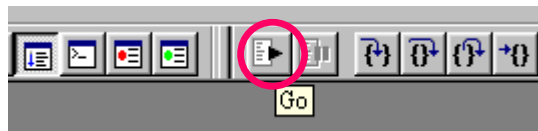
- Choose ***C:\Program Files\ARM\Logic + Modules\LM-V600E\example2\sw\sw_Data\DebugRel\sw.axf*** (be sure to configure target to Multi-ICE..)



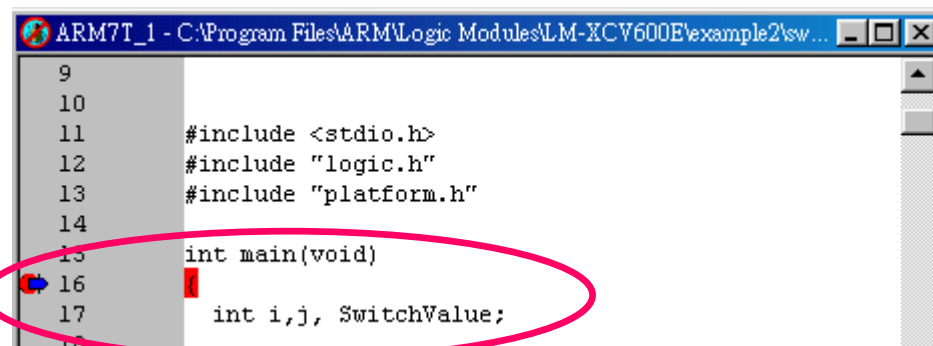
Execution of Example2 (cont.)



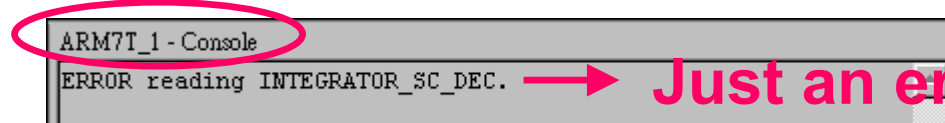
- Go! (or press F5)



- Another window bump up, program break at main function...



- Observe the result on the Console window and AP board...



→ Just an error message because
operate on ARMulator, Not normal result!

Exercise: RGB to YCrCb Converter



- Convert the *rgb2ycrcb()* into hardware module and implement it on the ARM development system. Evaluate the improvement.

- Hint: you may modify the
ahbahbtop.v
ahbdecoder.v
ahbmuxs2m.v
ahbzbtram.v
in example2

Function:

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

```
void main(void)
```

```
{  
    int a, b, c;  
    ...  
    rgb2ycrcb(a, b, c);  
    ...  
}
```

```
void rgb2ycrcb(int &a, int &b, int &c)
```

```
{  
    int y, cb, cr;  
    y=0.257*a+0.504*b+0.098*c+16;  
    cb=-0.148*a-0.291*b+0.439*c+128;  
    cr=0.439*a-0.368*b-0.071*c+128;  
    a=y; b=cb; c=cr;  
}
```