LAB PRACTICE:
A BASIC SOC PLATFORM

OVERVIEW

In this module, we will build a basic SoC platform, work includes:

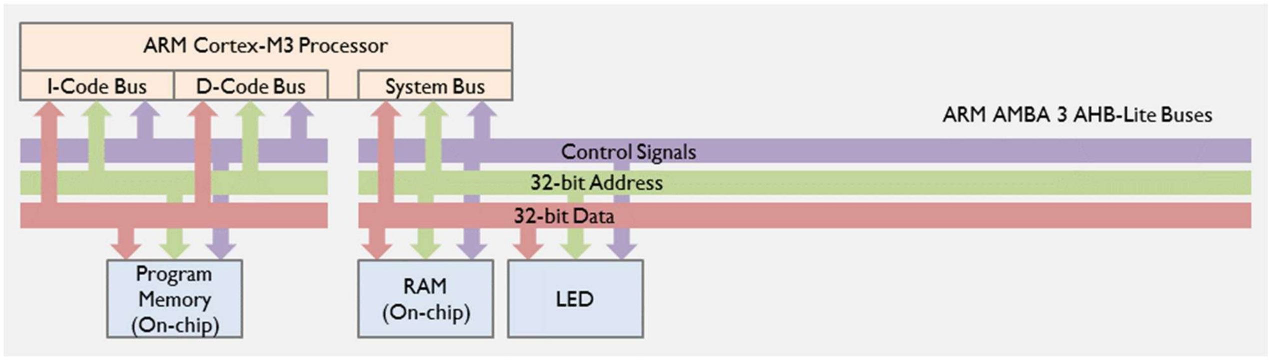
- Hardware implementation:
    - Implement the hardware framework which has been written in Verilog
- Software programming:
    - Program the Cortex-M3 processor using assembly language
    - Simulate the program using Keil µVision Simulator
- Demonstrate the SoC:
    - Toggle the 8-bit LEDs at a given frequency
    - Analyze the behavior of the AHB bus using on-chip hardware debugging tool
- Modify the hardware/software to implement new funcitons:
    - Modify the hardware and software code to add new functions to the module.

DETAILS

HARDWARE

The hardware components of the SoC include:

- An ARM Cortex-M3 microprocessor
- AHB-Lite buses
- Three AHB slaves
    - Program memory and RAM (implemented using on-chip memory blocks)
    - A simple LED peripheral



SoC Peripherals

- ARM Cortex-M3 Microprocessor:

The logic of the ARM Cortex-M3 processor is written in Verilog code. In this set of teaching material, we use a simplified version of the Cortex-M3 processor, called Cortex-M3 DesignStart. The Cortex-M3 DesignStart has almost the same functionality of an industry-standard Cortex-M3 processor, except that some configurations are fixed.

- On-chip program memory:

To program a processor, your software code needs to be compiled to machine codes, which are the instructions to be executed by the processor. The physical memory used to store these instructions is called a program memory. In this basic SoC platform, the program memory is implemented using the on-chip memory blocks, rather than off-chip memories.

The program memory can be accessed by both I-Code AHB bus, i.e. when reading instructions, and D-Code AHB bus, i.e. when reading constants or initialization value of variables. Usually a 2-master-1-slave bus arbiter is needed at least to arbitrate between the two AHB buses. Cortex-M3 DesignStart has integrated such an arbiter to ensure there is only one master working at any moment and the D-Code bus has a higher priority than the I-Code bus. Therefore, we can use the higher bit of signal HTRANSD to multiplex between the two buses.

The program image can be merged into your hardware design during synthesizing. For example, if you need to preload a program file into the hardware, the program file (e.g. "lab1.hex") needs to be referred in your Verilog code, using syntax such as:

```
initial begin
$readmemh("lab1.hex", memory);
end
```

- On-chip RAM:

Unlike the program memory, the RAM is another memory in SoCs usually to store volatile data, i.e. variables in C language. It is access by the CPU through the System AHB bus together with on-chip peripherals. But we will not use the RAM until we start to program in C language in the following labs.

- LED peripheral:

The LED peripheral is a simple module used to interface with the 8-bit LEDs. It has an AHB bus interface, which allows the LED to be connected to the system AHB bus, and controlled by the Cortex-M3 processor.

The files needed in this lab are listed below:

FILES TO BE USED IN THIS LAB

Components	**File name**	**Description**
Cortex-M3 processor	CORTEXM3INTEGRATIONDS.v	Cortex-M3 DesignStart processor macro cell level.
	cortexm3ds_logic.v	Cortex-M3 DesignStart processor logic level Verilog file. The DesignStart is a simplified version used for education, or providing fast and efficient access to industry usages.
AHB bus component	AHBDCD.v	The address decoder of the AHB bus.
	AHBMUX.v	The slave multiplexor of the AHB bus.
AHB on-chip memory peripheral	AHB2BRAM.v	The on-chip memory (BRAM) that used for the program memory of the processor.
AHB LED peripheral	AHB2LED.v	The LED peripheral module.
Top module	ARMSOC_TOP.v	The top-level module. Process most of the control signals of Cortex-M3 DesignStart and implement a simple multiplexor between D/I-Code buses.

The memory map can be defined in the AHB bus address decoder, below is the default memory map of the two peripherals:

MEMORY MAP OF PERIPHERALS

Peripheral	**Base address**	**End address**	**Size**
Program Memory	0x0000_0000	0x0000_7FFF	32KB
RAM	0x2000_0000	0x2000_7FFF	32KB
LED	0x5000_0000	0x50FF_FFFF	16MB

SOFTWARE

In this lab, we will use the assembly language to program the Cortex-M3 processor. The assembly language allows us to access the registers in a low-level; hence we can have a better understanding of the low-level hardware mechanism.

File name	**Description**
cmdsasm.s	The assembly code used in this lab

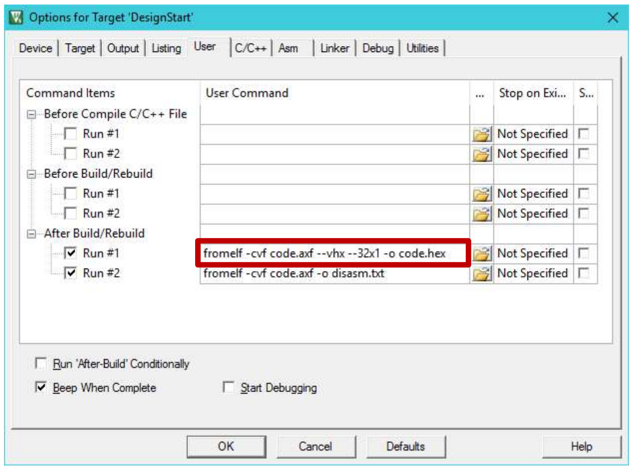
The main code should perform the following:

- Initialize the interrupt vector
- In the reset handler, repeat the following:
    - Turn on half of the 8-bit LEDs, e.g. LED [0, 2, 4, 6]
    - Set a counter, and use it to delay for short time
    - Turn on the other half of the LEDs, e.g. LED [1, 3, 5, 7]
    - Delay for another period

TESTING

Compile the software code, and generate the executable file in hex format using command:

```
fromelf -cvf code.axf --vhx --32x1 -o lab1.hex
```


Options for Target 'DesignStart'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
OK
Cancel
Defaults
Help

Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
✓ Run #1	fromelf -cvf code.axf --vhx --32x1 -o code.hex		Not Specified	☐
✓ Run #2	fromelf -cvf code.axf -o disasm.txt		Not Specified	☐

☐ Run 'After-Build' Conditionally
✓ Beep When Complete ☐ Start Debugging

Copy the executable file “lab1.hex” to the directory “Hardware”.

In the Hardware directory, you will see a file “run.do”:

```
restart
force -deposit top.clk_div 0
run 500ns
```

Its function is to initialize the clk_div in the top level design to 0, otherwise your clock will always be “X”.

Open Modelsim and use **vlog** to compile all Verilog source code to “Work” library. Load the simulation of top level model “testbench” by suing command **vsim**, and run the simulation by using the command “**do run.do**”.

The simulation tool allows you to analyze a set of signals, the suggested signals are:

- HADDRS[31:0]
- HWDATAS[31:0]
- HRDATAS[31:0]
- HWRITES
- HREADYS
- HSIZES[2:0]
- HTRANSS[1:0]
- HRESPS

WHAT YOU NEED TO DO

In this lab, you need to:

1. Run the simulation, take a look of the waveform and analyze the signal activities in the AHB transactions when the CPU writes to the LED register. Report the exact time of the address phase and data phase of the first 2 LED write transactions. Also report what signals are asserted (transmitted) during those two phases.
2. Instead of writing to address 0x5000_0000, write to 0x5000_0010, and observe the LED signals. What do you see?
3. The LED register is currently mapped to memory address 0x5000_0000. Modify the source code and hardware design to map this register to 0x5700_0000. Simulate the whole system again and report the waveform of your simulation. Include the waveform in the report. The waveform should contain all aforementioned system bus signals and the LED signals as a proof that the address mapping has taken place and the LED register is written correctly.
4. Based on (3), further modify the LED module and add an additional mask register that is read and writable by the CPU at memory address 0x5700_0004. A "0" in the mask register will prevent the corresponding bit in the LED register to be written (i.e. the corresponding bit in LED register will keep its previous value instead of being modified.) Modify your software program to test the mask function. You need provide a brief description of the steps that were taken in your test program, (e.g. step 1: write a 0Xff to LED register, step 2: write a 0x0F to the mask register, step 3: write a 0Xaa to LED register. Step 4: observe that the value in LED register is 0xfa.) Report the waveform for those steps.

Submit a report of the above tasks. You do not need to submit source code for hardware and software design.