Lab Exercise 4: A Basic SoC Platform

# Overview

In this lab, we build a basic SoC platform. Work includes:

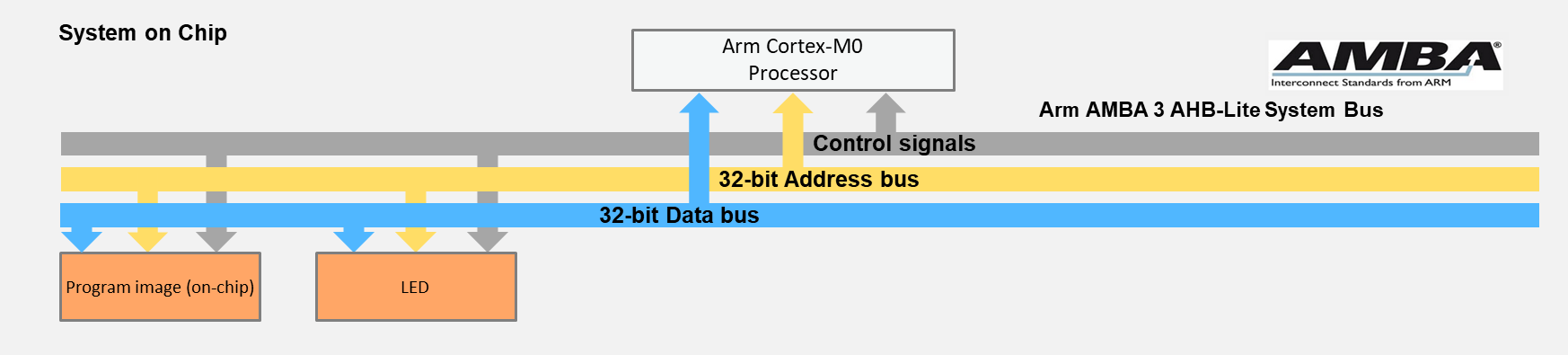
* Hardware implementation:
* Implement the hardware framework that has been written in Verilog.
* Prototype the SoC framework onto an FPGA.
* Software programming:
* Program the Cortex-M0 processor using assembly language.
* Simulate the program using Keil μVision Simulator.
* Demonstrate the SoC:
* Toggle the 8-bit LEDs at a given frequency.
* Analyze the behavior of the AHB bus using an on-chip hardware debugging tool.

# Details

## Hardware

The hardware components of the SoC include:

* An Arm Cortex-M0 microprocessor
* An AHB-Lite system bus
* Two AHB peripherals
  + Program memory (implemented using on-chip memory blocks)
  + A simple LED peripheral



**SoC Peripherals**

* Arm Cortex-M0 microprocessor:

The logic of the Arm Cortex-M0 processor is written in Verilog code, and thus can be prototyped (synthesized and implemented) on an FPGA platform. In this set of teaching materials, we use a simplified version of the Cortex-M0 processor, called Cortex-M0 DesignStart. The Cortex-M0 DesignStart has almost the same functionality of an industry-standard Cortex-M0 processor, except that some features are reduced; e.g., the number of interrupts is reduced from the original 32 to 16 interrupts.

* 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. For example, the block RAM (BRAM) is one type of on-chip memory on Xilinx FPGAs.

Normally, in order to load your program into the on-chip memory of an FPGA, the program image needs to 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., “code.hex”) needs to be referred to in your Verilog code, using syntax such as:

initial begin

$readmemh("code.hex", memory);

end

In the following labs, the on-chip memory will be replaced by an off-chip memory, which needs you to develop a memory controller that interfaces with an off-chip memory. After that, your program file could be separately loaded without reconfiguring the FPGA.

* 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-M0 processor.

The files needed in this lab are listed below:

FILES TO BE USED IN THIS LAB

|  |  |  |
| --- | --- | --- |
| **Components** | **File name** | **Description** |
| Cortex-M0 processor | cortexm0ds\_logic.v | Cortex-M0 DesignStart processor logic level Verilog file: The DesignStart is a simplified version used for education, or providing fast and efficient access to industry usages. |
| CORTEXM0INTEGRATION.v | Cortex-M0 DesignStart processor macro cell level |
| 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) used for the program memory of the processor |
| AHB LED peripheral | AHB2LED.v | The LED peripheral module |
| Top module | AHBLITE\_SYS.v | The top-level module |
| User constraint file | .xdc/.ucf | The user constraint file (depending on board type) |

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** |
| MEM | 0x0000\_0000 | 0x0000\_FFFF | 16MB |
| LED | 0x5000\_0000 | 0x50FF\_FFFF | 16MB |

## Software

In this lab, we will use the assembly language to program the Cortex-M0 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** |
| cm0dsasm.s | The assembly code used in this lab |

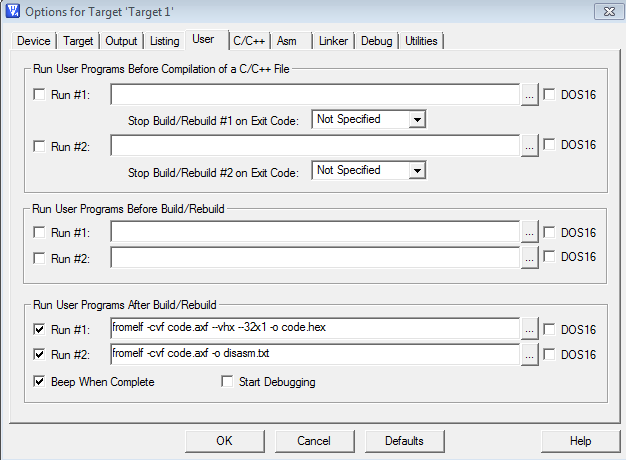
The main code should perform the following:

* 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 a 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 code.hex



Copy the executable file “code.hex” to the FPGA project directory.

Synthesize the hardware using FPGA synthesis tools from the FPGA vendors, e.g., Xilinx ISE, Vivado and Altera Quartus.

After reset (controlled by a button), LEDs should start flashing.

# HARDWARE debugging

## hardware logic simulation

Before downloading the hardware to the FPGA, we can use hardware simulation tools to analyze the system behavior, such as MentorGraphic ModelSim and Xilinx Isim.

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

* HADDR[31:0]
* HWDATA[31:0]
* HRDATA[31:0]
* HWRITE
* HREADY
* HSIZE[2:0]
* HTRANS[1:0]
* HRESP

## on-chip debugging

After the FPGA is configured, the live signals can be sampled and analyzed at run-time, which is different from the hardware simulation.

To sample the signals at run-time, on-chip debugging tools are required, for example, ChipScope from Xilinx and SignalTap from Altera.

Analyze AHB bus behavior by sampling the following signals:

* HADDR[31:0]
* HWDATA[31:0]
* HRDATA[31:0]
* HWRITE
* HREADY
* HSIZE[2:0]
* HTRANS[1:0]
* HRESP

The on-chip debugging tool will also be useful in the following developments of hardware peripherals.

# Extension work

Here are some extra things that you can do:

* Add additional registers to the LED peripheral. For example, add a mask register that can mask out certain bits when writing the LEDs.
* Add another peripheral “AHB switch” to input the status of the 8-bit switches. For example, use the switch to control the LEDs.