***Intro to System-on-Chip Design Course***

**LAB 4**

**Cortex M0 and AHB-lite implementation**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817334)

[1.1 Lab overview 1](#_Toc83817335)

[1.1.1 Part 1: Hardware implementation: 1](#_Toc83817336)

[1.1.2 Part 2: Software programming 1](#_Toc83817337)

[2 Learning Objectives 1](#_Toc83817338)

[3 Requirements 2](#_Toc83817339)

[4 Project files 2](#_Toc83817340)

[5 Hardware 3](#_Toc83817341)

[5.1 Overview of the SoC hardware 3](#_Toc83817342)

[5.1.1 Arm Cortex-M0 microprocessor 3](#_Toc83817343)

[5.1.2 On-chip program memory: 3](#_Toc83817344)

[5.1.3 LED peripheral 4](#_Toc83817345)

[5.2 Implementing the System-on-Chip 4](#_Toc83817346)

[6 Compiling assembly code and programming it into Basys 3 board 5](#_Toc83817347)

[6.1 Creating and compiling a program in assembly language 5](#_Toc83817348)

[6.2 Merging bitstream 6](#_Toc83817349)

[6.3 Programming FPGA board 6](#_Toc83817350)

[7 Hardware Debugging 7](#_Toc83817351)

[7.1 Hardware logic simulation 7](#_Toc83817352)

[7.2 On-chip debugging 7](#_Toc83817353)

[8 Extension work 8](#_Toc83817354)

[8.1 Extra tasks for this lab: 8](#_Toc83817355)

# Introduction

## Lab overview

In this lab, we will begin designing a simple SoC platform that contains a Cortex-M0 microprocessor, an LED peripheral, and an on-chip memory. The Cortex-M0 can be downloaded from the DesignStart webpage found in <https://developer.arm.com/ip-products/designstart/eval>.

You will learn how to integrate the processor and other peripherals into a system using the AHB-Lite bus interconnect and write/modify assembly programs to control the peripheral. This lab is divided into two parts as shown below

### Part 1: Hardware implementation:

The processor, bus interface, on-chip memory and peripheral hardware are written in Verilog and provided for you, with some modification/additions needed to make it work. The SoC will be implemented in an FPGA.

### Part 2: Software programming

The program targeted at the Cortex-M0 processor is written in assembly language and will be used to control the LEDs. The program is provided for you. You will need to compile it in Keil to generate a ***code.hex*** file which will be copied to the FPGA project directory.

# Learning Objectives

At the end of this lab, you should be able to:

* Implement a simple SoC which consist of Cortex-M0 processor, AHB-Lite bus and AHB peripherals (Program memory and LED) on an FPGA.
* Modify and compile a simple assembly program using Keil to control the LED.

# Requirements

This lab requires the following hardware and software:

* **Hardware:**
  + **Diligent BASYS 3** FPGA board connected to computer via **MicroUSB cable.** A constraints file for this board is also provided.
* **Software**
  + **Xilinx Vivado**
    - Xilinx Vivado.

|  |  |
| --- | --- |
| Lights On | This lab was tested using Vivado 2019.1. The *tcl* script provided to auto add the compiled program to the generated bitstream works with this version.  The *tcl* scripts may not work with later versions of Vivado. In this case, you will need to rerun synthesis, implementation and bitstream generation when a new code.hex file is generated due to change made to the program. |

* + **Keil uVision**

# Project files

The following files are provided/needed for this Lab:

|  |  |  |
| --- | --- | --- |
| **Components** | **File name** | **Description** |
| Cortex-M0 processor | cortexm0ds\_logic.v | Cortex-M0 DesignStart processor logic level Verilog file |
| CORTEXM0INTEGRATION.v | Cortex-M0 DesignStart processor macro cell level |
| AHB bus component | AHBDCD.v | The address decoder of the AHB bus |
| AHBMUX.v | The subordinate 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 |
| Program executed by FPGA board | cm0dsasm.s | The assembly code used in this lab |

# Hardware

In this task, you will implement the Cortex-M0 processor core, AHB-lite bus, memory and LED peripheral on an FPGA board.

## Overview of the SoC hardware

The hardware components of the SoC include:

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

Diagram, timeline

Description automatically generated

Figure 1: 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 code, which contains the instructions to be executed by the processor. The physical memory used to store these instructions is called the 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, to load your program into the on-chip memory of an FPGA, the program image needs to be merged into your hardware design during synthesis. 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

However, this approach requires complete regeneration of the bitstream for minor changes in code, which can be very time-consuming for larger designs. In the labs for this course, we have provided a post-implementation *tcl* script which merges the program code to the bitstream. This method is much quicker for simple testing of different program images and the process will be described later.

**Note: The TCL script were tested on Vivado released in 2019. We cannot guarantee that it will work for other versions of Vivado. If it does not work, you may need to modify the TCL script, or synthesize the code manually each time code.hex is updated.**

### 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.

## Implementing the System-on-Chip

Start Vivado and follow the steps in Section 4.2 of the Getting Started Guide to create a new project:

1. In the Add Constraints page click create new file and name it ***basys\_3\_constraints***.
2. On the Device screen chose XC7A35Tcpg236-1

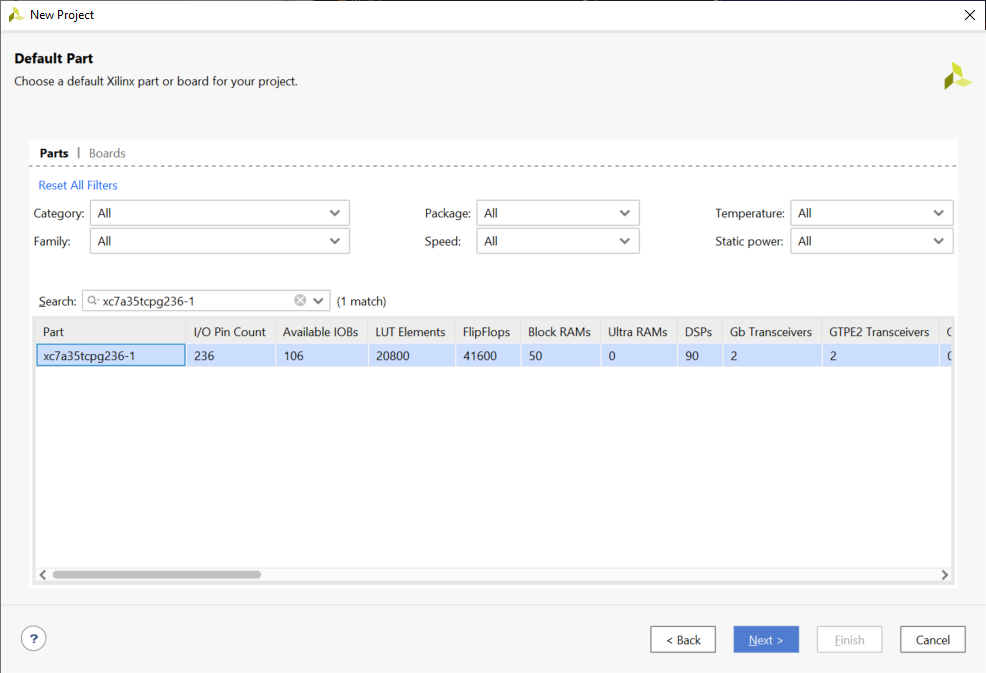


Figure 2: Select FPGA part

Hierarchy of files in complete project should look like this:

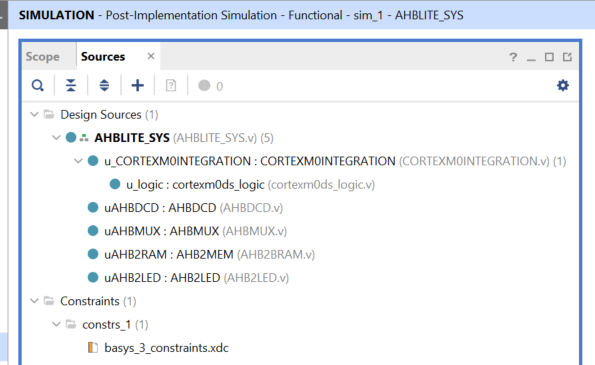


Figure 3: Project hierarchy

Run synthesis, implementation and generate bitstream if you are using **Vivado 2019.1**. You can also connect a board now but note that there is no program in a memory yet!

Our next step will be to compile an assembly code and merge it with the generated bitstream (effectively putting it in the memory in the already generated bitstream file in Vivado).

# Compiling assembly code and programming it into Basys 3 board

Please see Keil uVision getting started Guideprovided with this lab for how to setupf Keil uVision for this course.

## Creating and compiling a program in assembly language

In this lab, we will use assembly language to program the Cortex-M0 processor. This will be edited in Keil. The assembly language allows low-level access to the registers, giving us a better understanding of the low-level hardware mechanism.

The main code does 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.

A working code in ***cm0dsasm.s*** has been provided with this lab. Study the code in the file. Check each label (Reset\_Handler, AGAIN, Loop and Loop1) and study the corresponding code. Note the register calling convention and how the instructions are used.

Once you have created/added an assembly program (please see Section 3.4 in the getting Started Guide), compile it into .hex file (please see Section 3.5 in the getting Started Guide).

## Merging bitstream

We have also provided a TCL script that can be used for the post-implementation flow. This script, ***update\_bitstream.tcl***, can be called from the Vivado TCL console and it will generate a new bitstream file, ***reflash.bit***, in the project directory. The script requires the following:

* A valid data file named **code.hex** in the project directory.
* An up-to-date bitstream file called **AHBLITE\_SYS.bit** generated in the implementation directory. This file is generated by Vivado using “Generate bitstream” option.
* The implemented design is open in Vivado.
* Both parts of the script, ***update\_bitstream.tcl*** and ***update\_bitstream\_header.tcl***, exist in the project directory.

**Note: The TCL script were tested on Vivado released in 2019. We cannot guarantee that it will work for other versions of Vivado. If it does not work, you may need to modify the TCL script, or synthesize the code manually each time code.hex is updated.**

On the *Tcl* Console Ensure you are in the Vivado working directory

The command “source ***update\_bitstream.tcl***” should be called from the project directory in the TCL console using following commands:

source update\_bitstream.tcl

* The new bitstream is saved as ***reflash.bit***. After it is generated, this file should be found exist in the project directory.

## Programming FPGA board

Program the board using the generated ***reflash.bit*** file. After reset, on the Basys 3 controlled by button **BTNR**, the 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 behaviour, 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 the AHB bus behaviour 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

# Extension work

## Extra tasks for this lab:

* 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.