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

**LAB 02**

**AHB VGA Peripheral**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817474)

[1.1 Lab overview 1](#_Toc83817475)

[1.1.1 Hardware design and implementation: 1](#_Toc83817476)

[1.1.2 Software programming: 1](#_Toc83817477)

[1.1.3 Demonstrate the SoC: 1](#_Toc83817478)

[2 Learning Objectives 1](#_Toc83817479)

[3 Requirements 2](#_Toc83817480)

[4 Project files 2](#_Toc83817481)

[5 Hardware design 3](#_Toc83817482)

[5.1 Overview of the SoC hardware 3](#_Toc83817483)

[5.2 VGA Peripheral 3](#_Toc83817484)

[5.2.1 VGA Peripheral Block diagram 3](#_Toc83817485)

[5.2.2 VGA Peripheral Memory map 4](#_Toc83817486)

[6 Software Design 4](#_Toc83817487)

[6.1 Main code tasks 4](#_Toc83817488)

[7 Hardware Debugging 5](#_Toc83817489)

[7.1 On-chip debugging 5](#_Toc83817490)

[8 Extension work 6](#_Toc83817491)

[8.1 Extra tasks for this lab: 6](#_Toc83817492)

# Introduction

## Lab overview

In this lab, we will implement an AHB VGA peripheral and write simple program for the processor to display images onto a VGA monitor. The steps include:

### Hardware design and 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.

### Software programming:

The program targeted at the Cortex-M0 processor is written in assembly language and will be used to access the VGA peripheral. 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.

### Demonstrate the SoC:

* Display texts and images on a VGA display.
* Analyze the behavior of the peripheral using an on-chip hardware debugging tool.

# 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, VGA) on an FPGA.
* Modify and compile an assembly code to initialize the interrupt vector.
* Modify and compile an assembly code to display text on a VGA.

1. **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 in Keil due to change made to the C program. |

* + **Keil uVision**

# Project files

You will need the files from the previous lab along with the following files which are provided with this Lab:

|  |  |
| --- | --- |
| **File name** | **Description** |
| AHBLITE\_SYS.v | Top level module |
| AHBVGASYS.v | The top module of the VGA, includes the AHB interface |
| vga\_image.v | The frame buffer in the image region |
| vga\_console.v | Used to display characters in the console region |
| vga\_sync.v | Used to generate VGA synchronization signals |
| font\_rom.v | The ROM used to store the pixels of a character |
| dual\_port\_ram\_sync.v | A dual-port synchronized on-chip RAM |
| counter.v | A generic counter |

# Hardware design

In this task, you will implement the Cortex-M0 processor core, AHB-lite bus, memory and VGA 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 VGA peripheral to interface with a VGA monitor

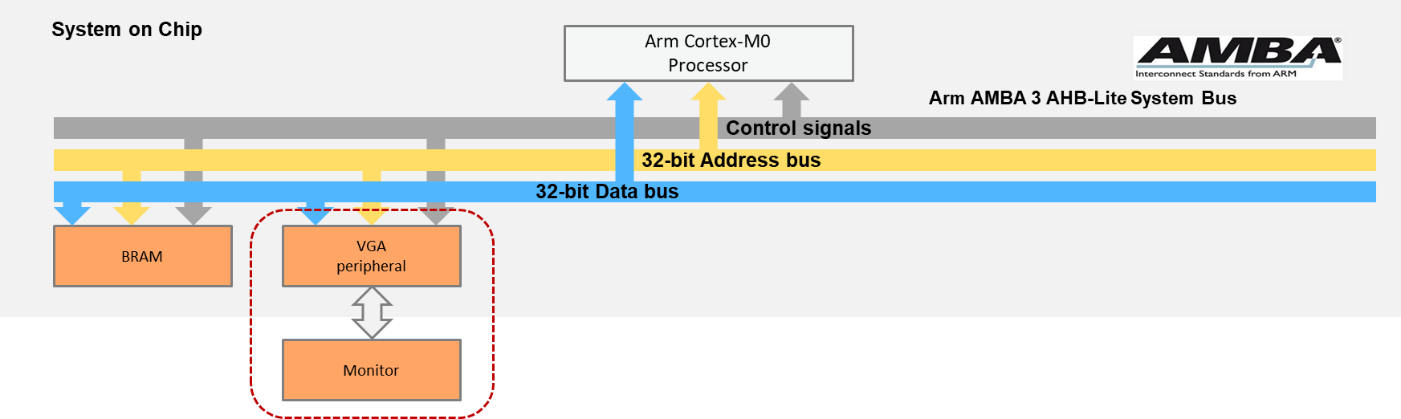


Figure :SoC Peripherals

## VGA Peripheral

The VGA peripheral is used to interface the AHB bus with a VGA monitor. The VGA peripheral has an AHB bus interface, which allows the VGA to be controlled by the Cortex-M0 processor.

### VGA Peripheral Block diagram

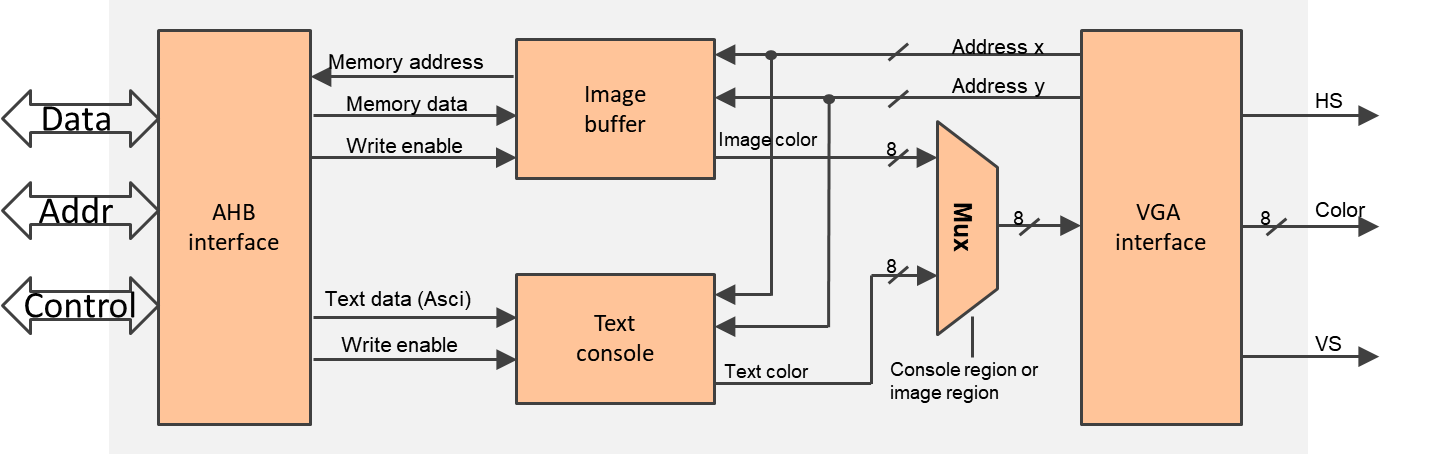


Figure :VGA Peripheral Block Diagram

* VGA interface
  + Generates synchronization signals to the VGA port
  + Is directly connected to external pins of the VGA port
  + Outputs the address of the current pixel
* Image buffer
  + Stores the color information of all pixels in the image region
  + Is implemented on a dual-port memory
* Text console
  + Displays texts in the text region
  + Is implemented on hardware logics

### VGA Peripheral Memory map

The following table shows the suggested memory map for peripherals and registers:

MEMORY MAP OF PERIPHERALS

|  |  |  |  |
| --- | --- | --- | --- |
| **Peripheral** | **Base address** | **End address** | **Size** |
| BRAM | 0x0000\_0000 | 0x00FF\_FFFF | 16MB |
| VGA | 0x5000\_0000 | 0x50FF\_FFFF | 16MB |

PERIPHERAL REGISTERS

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Base address** | **End address** | **Size** |
| Console text | 0x5000\_0000 | 0x5000\_0000 | 4 byte |
| Image buffer | 0x5000\_0004 | 0x50FF\_FFFF | (16M-4) byte |

* Console text: 1 word (4 byte) space to print a character
* Image buffer: the rest of the space is used to store pixels in the image region.

# Software Design

## Main code tasks

Create a new Keil project and include the assembly file provided for this lab.

The main code is written in assembly and performs the following:

* Initialize the interrupt vector.
* Display a text string (e.g., “TEST”) at the console region.
* Plot four pixels at the four corners of the image region.

An example of the demo:



Figure : Demo Example

# Hardware Debugging

## On-chip debugging

Use an on-chip debugging tool to sample and analyze the signals at run-time. Suggested signals are:

Towards AHB bus:

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

Towards the VGA interface:

* Horizontal\_Sync
* Vertical\_Sync
* VGA\_Color
* Address\_x
* Address\_y

# Extension work

## Extra tasks for this lab:

* Draw lines, rectangles, or even a picture at the image region.
* Add configuration registers to the VGA peripheral, whereby the processor can configure the peripheral by modifying its configuration registers, for example
  + Change the image resolution.
  + Adjust the size of the text region and the image region.
* Display live videos at the image region.
* Change languages at the console region.