Lab Exercise 7: Timer, GPIO and 7-Segment Peripherals

# Overview

In this lab, we design and implement three AHB peripherals: an internal timer, a general-purpose input output (GPIO), and a 7-segment display. The work includes:

* Hardware design and implementation:
* Design and implement the hardware mechanism of the three peripherals.
* Design and implement the AHB bus interface for the three peripherals.
* Prototype the hardware onto a FPGA.
* Software programming:
* Program the Cortex-M0 processor; access the three peripherals using assembly language.
* Demonstrate the SoC:
* Display the timer on the 7-segment display.
* Use GPIO to interface with the switches and LEDs.
* Analyze the behavior of the peripheral using an on-chip hardware debugging tool.

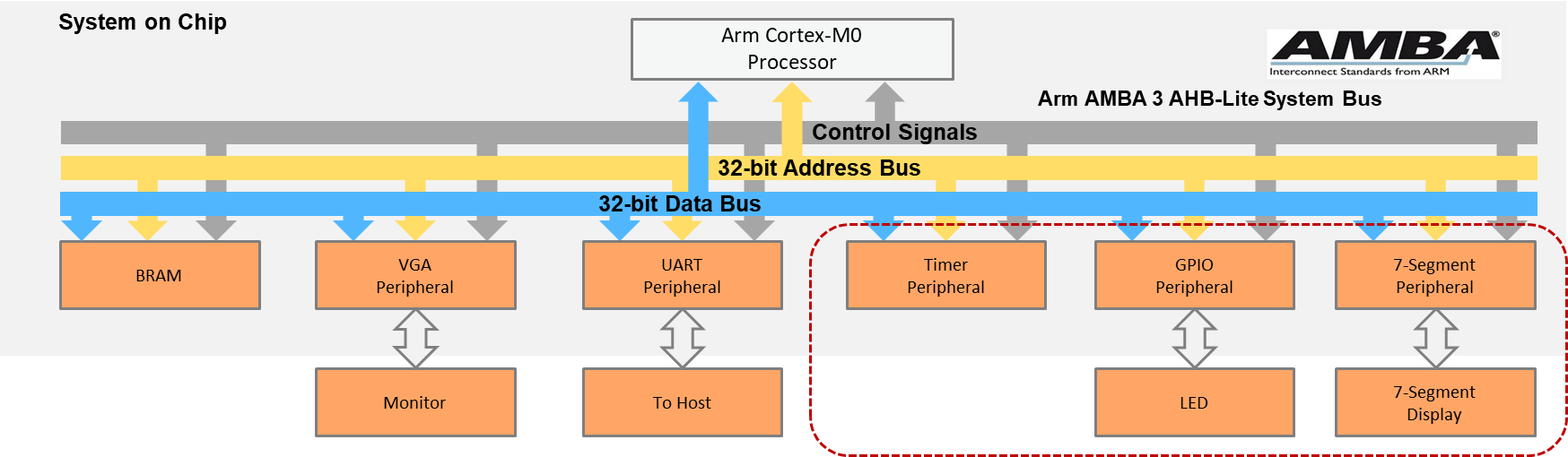
# Details

## Hardware

### system components

The hardware components of the SoC include:

* An Arm Cortex-M0 microprocessor
* An AHB-Lite system bus
* Six AHB peripherals
  + A BRAM memory module
  + A VGA peripheral
  + A UART peripheral
  + An internal timer peripheral
  + A GPIO peripheral
  + A 7-segment display peripheral



**SoC Peripherals**

### file structure

The files that will be used for the peripherals are listed below:

FILES USED BY VGA PERIPHERAL

|  |  |
| --- | --- |
| **File name** | **Description** |
| AHBTIMER.v | The AHB timer peripheral, including the AHB bus interface |
| prescaler.v | The prescaler is used to scale-down the frequency for the timer. |
| AHBGPIO.v | The AHB GPIO peripheral, including the AHB bus interface |
| AHB7SEGDEC.v | The AHB 7-segment display peripheral, including the AHB bus interface |

The structure of the files:

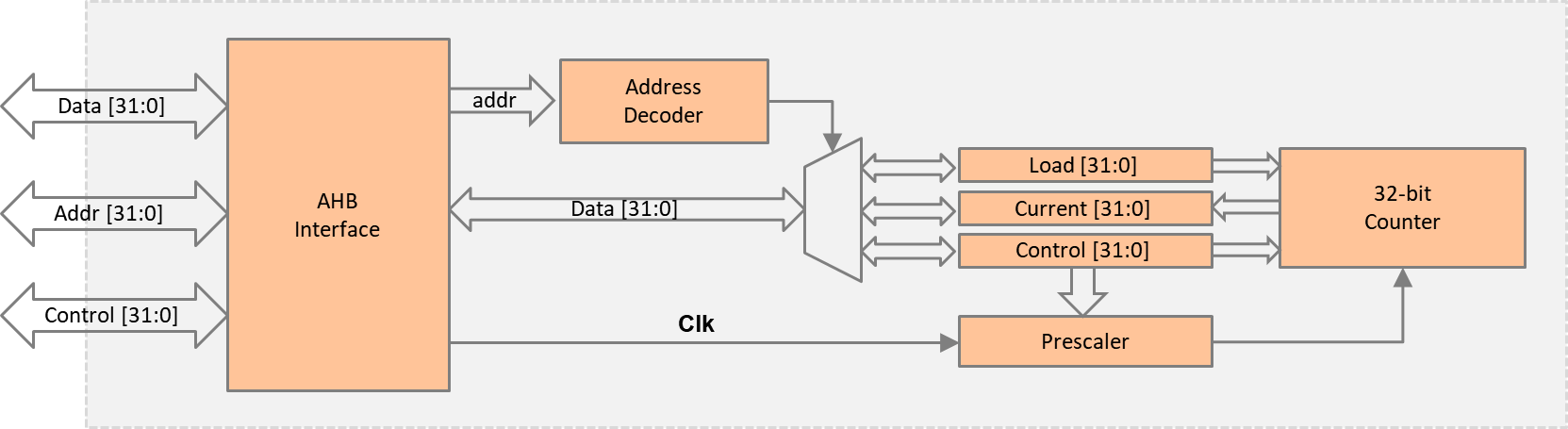
AHB7SEGDEC.v

AHBGPIO.v

AHBTIMER.v

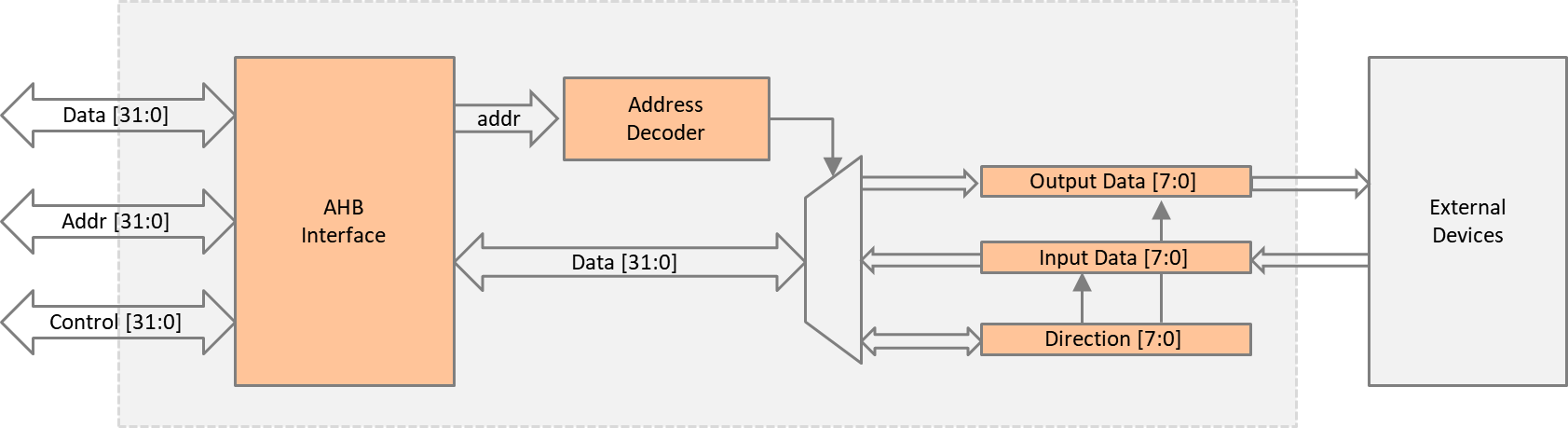
prescaler.v

### block diagram

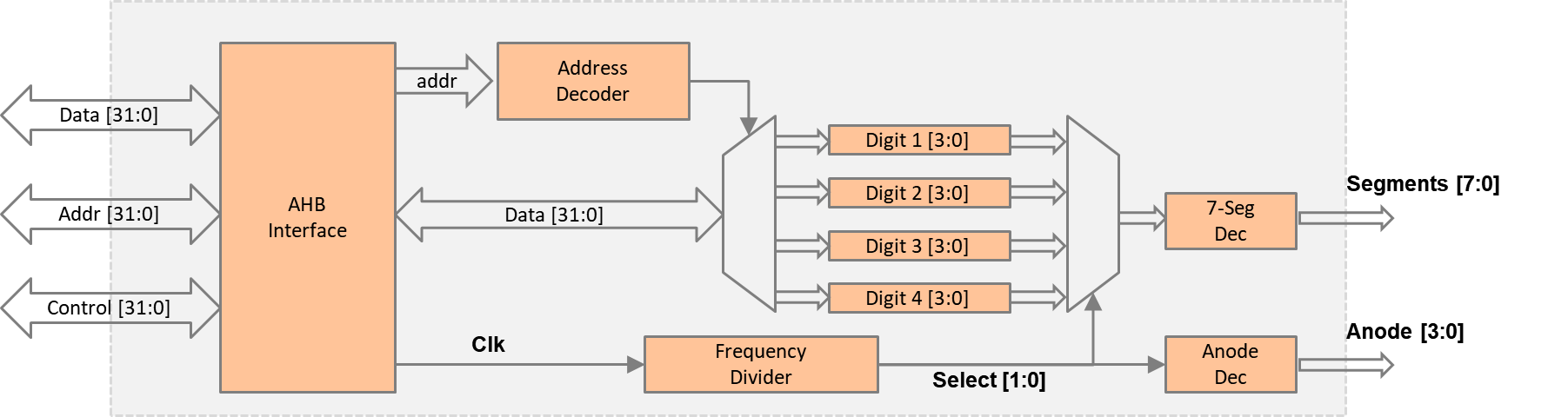


­

**Timer Peripheral Block Diagram**



**GPIO Peripheral Block Diagram**



**7-Segment Display Peripheral Block Diagram**

### memory map

The default memory map of the peripherals is listed below:

MEMORY MAP OF PERIPHERALS

|  |  |  |  |
| --- | --- | --- | --- |
| **Peripheral** | **Base address** | **End address** | **Size** |
| SRAM | 0x0000\_0000 | 0x00FF\_FFFF | 16MB |
| VGA | 0x5000\_0000 | 0x50FF\_FFFF | 16MB |
| UART | 0x5100\_0000 | 0x51FF\_FFFF | 16MB |
| Timer | 0x5200\_0000 | 0x52FF\_FFFF | 16MB |
| GPIO | 0x5300\_0000 | 0x53FF\_FFFF | 16MB |
| 7-segment display | 0x5400\_0000 | 0x54FF\_FFFF | 16MB |

TIMER PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Load value | 0x5200\_0000 | 4 bytes |
| Current value | 0x5200\_0004 | 4 bytes |
| Control value | 0x5200\_0008 | 4 bytes |

* Load value register

The reset value when the timer reaches zero

* Current value register

The current value of the 32-bit counter

* Control register

Used to start/stop a counter and set the prescaler

GPIO PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Data | 0x5300\_0000 | 4 bytes |
| Direction | 0x5300\_0004 | 4 bytes |

* Data registers
  + Input data: the data read from external devices
  + Output data: the data sent to external devices
* Direction register
  + Controls whether it is a read or write operation

7-SEGMENT DISPLAY PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Digit1 | 0x5400\_0000 | 4 bytes |
| Digit2 | 0x5400\_0004 | 4 bytes |
| Digit3 | 0x5400\_0008 | 4 bytes |
| Digit4 | 0x5400\_000C | 4 bytes |

* Digit 1: the first digit on the 7-segment display
* Digit 2: the second digit on the 7-segment display
* Digit 3: the third digit on the 7-segment display
* Digit 4: the forth digit on the 7-segment display

## Software

The main code should be written in assembly and should perform the following:

* Initialize the interrupt vector.
* Display a string (e.g., “TEST”) at the console region (same as previous lab).
* Initialize the timer.
  + Write the load value register.
  + Set prescaler, e.g., 1x or 16x.
  + Change the operation mode.
    - Free-run mode: the timer resets on overflow.
    - Load mode: the timer resets when reaching the value in the load register.
  + Start the timer

Then repeat the following:

* Display the timer
  + Read the current timer value.
  + Choose the higher bytes, e.g., 16-bit MSB.
  + Display the value on the 7-segment display using the four digits.
* GPIO test
  + Change the direction of the GPIO to read mode.
  + Read the switch value from the GPIO.
  + Change the direction of the GPIO to write mode.
  + Write the switch value to the LEDs.

# HARDWARE debugging

## on-chip debugging

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

Towards AHB bus:

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

Towards the peripherals:

* Timer\_Select
* Timer\_Write
* Timer\_Ready
* Timer\_Data
* GPIO\_Select
* GPIO \_Write
* GPIO \_Ready
* GPIO \_Data
* 7-SEG\_Select
* 7-SEG \_Write
* 7-SEG \_Ready
* 7-SEG \_Data

# extension work

Here are some extra things that you can do:

* In the timer peripheral, add more operation modes to the timer, such as compare mode, capture mode, or PWM mode.
* In the GPIO peripheral, add a mask register that can mask out certain bits while writing to the GPIO.
* In the 7-segment display peripheral, add a register to change the display mode, such as display in hex or decimal.