Lab Exercise 8­: Interrupt Mechanisms

# Overview

In this lab, we design and implement the interrupt mechanism for the timer and the UART peripheral. The work includes:

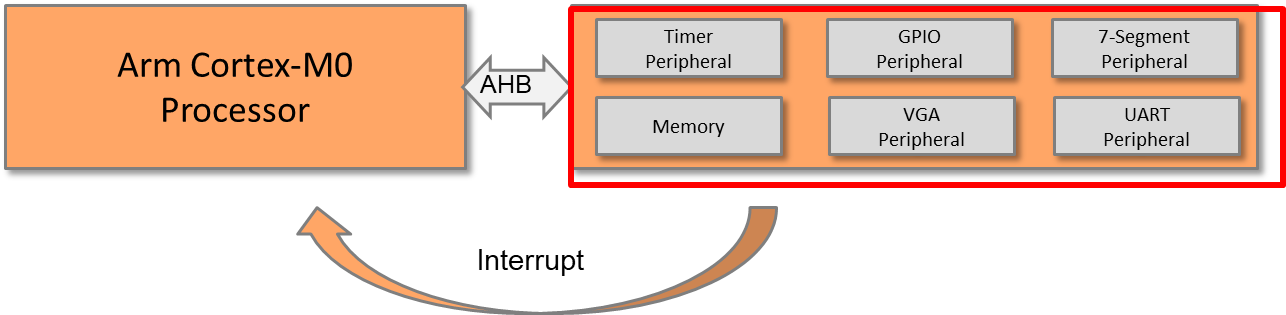
* Hardware design and implementation:
* Design and implement the interrupt mechanism for the AHB timer.
* Design and implement the interrupt mechanism for the AHB UART.
* Prototype the hardware onto a FPGA.
* Software programming:
* Program the Cortex-M0 processor; write the interrupt service routines in assembly language.
* Demonstrate the SoC:
* Use the timer interrupt to implement a counter (counting from 0 to 9), and display the value to the VGA display.
* Use the UART interrupt to send characters to a PC or laptop.

# Details

## Hardware

The hardware components of the SoC include all the peripherals that have been developed in the previous modules.

The AHB timer and the AHB UART will need to be reimplemented with interrupt signals.

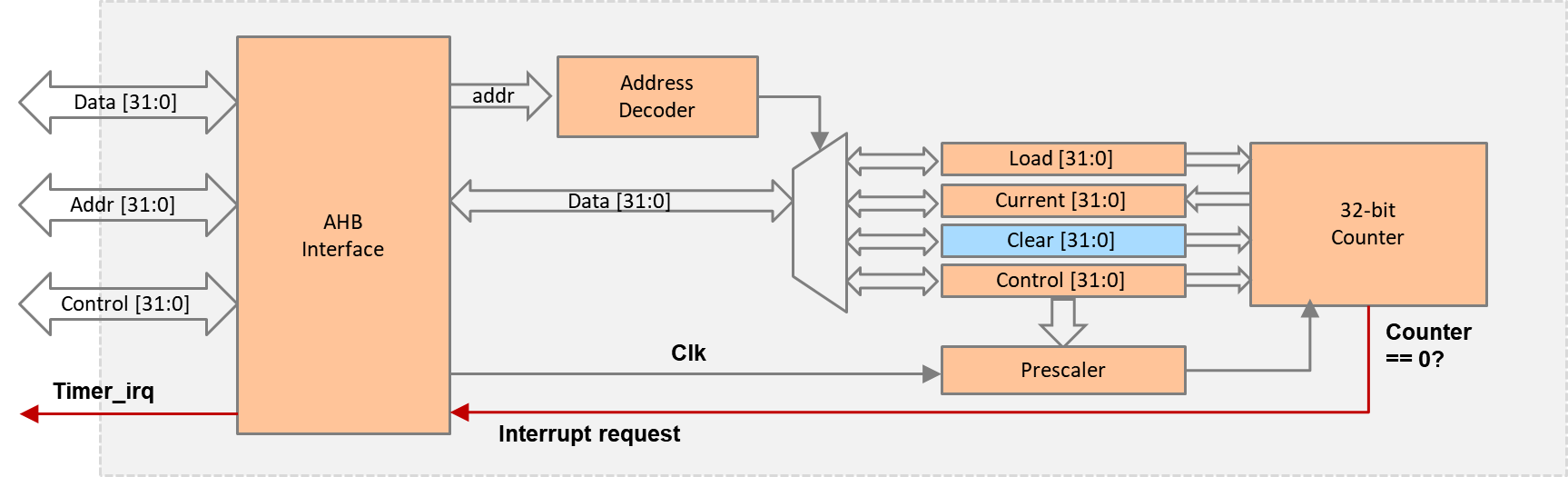


**SoC Peripheral Interrupts**

### Implementation of the timer interruption mechanism

An interrupt is generated every time the counter reaches zero.

A clear register needs to be added; this is used to clear the interrupt request once the processor finishes its ISR.



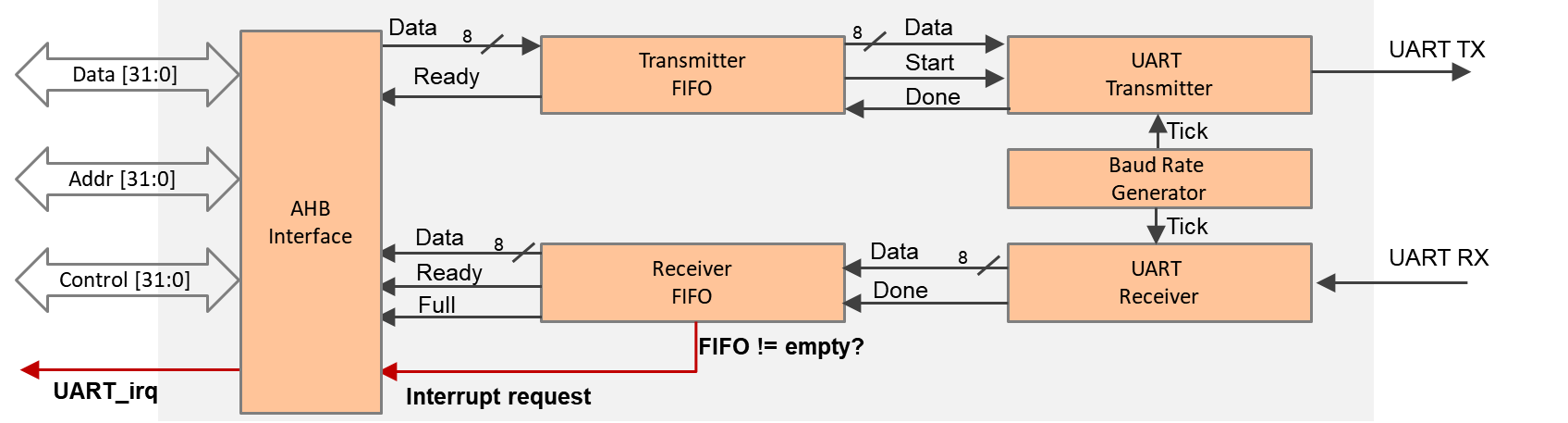
**Timer Interrupt Signal**

TIMER PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Load value | 0x5300\_0000 | 4 bytes |
| Current value | 0x5300\_0004 | 4 bytes |
| Control value | 0x5300\_0008 | 4 bytes |
| Clear register | 0x5300\_000C | 4 bytes |

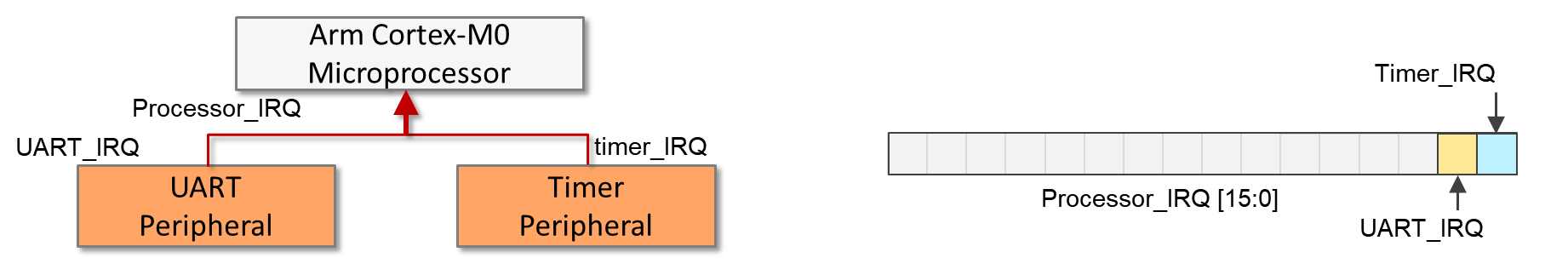
### Implementation of the UART interruption mechanism

For example, the interrupt can be generated if the receiver FIFO is not empty.

** UART Interrupt Signal**

Unlike the timer interruption, we can omit the interrupt cleaning step, since the interrupt request will be automatically cleared after the data is read out from the FIFO.

### Connect the interrupt to the processor



**Connect Interrupt Signals to the Processor**

## Software

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

* Initialize the interrupt vector, adding the timer and the UART interrupt vectors
* Reset handler
  + Set the timer interrupt priority to 0x00 (higher).
  + Set the UART interrupt priority to 0x40 (lower).
  + Enable interrupts for the timer and UART.
  + Initialize the timer to generate an interrupt every second.
    - Write the load value register, e.g., 50,000,000
    - Set prescaler, e.g., 1x or 16x.
    - Change the operation mode to load mode.
  + Start the timer.
  + Set up a counting up counter; start from “0” (ASCii=0x30).
* Timer interrupt handler
  + Push registers to the stack (e.g., R1-R4).
  + Clear the timer interrupt request.
  + Increment the counter.
  + Display the counter to the VGA text region.
  + Disable the timer interrupt if the counter reaches 9.
  + Pop the registers from the stack.
* UART interrupt handler
  + Push registers to the stack.
  + Read from the UART (from the keyboard).
  + Write to the UART (to the terminal window).
  + Pop the registers from the stack.

# 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:

* Processor\_IRQ
* Timer\_IRQ
* UART\_IRQ

# extension work

Here are some extra things that you can do:

* Implement the interrupt mechanism for the GPIO peripheral; e.g., use switch as external interrupts.
* Implement a watchdog and connect it to the non-maskable interrupt (NMI).