Timer: A Peripheral Interface

Learning Goal: Implement a peripheral interface.

Requirements: ModelSim, Quartus.

1 Introduction

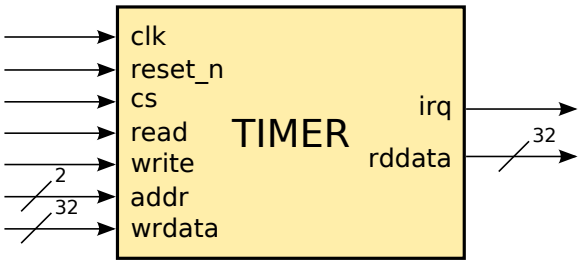
The goal of the first group of labs is to implement a stopwatch on FPGA which can accurately measure the time while reacting to the user's input. To do so, you will first develop the following components:

- A hardware timer (**this lab**), which generates a periodic interrupt.
- Interrupt service routines (ISRs) to handle the interrupts from the timer and from the buttons (**lab 2**).
- A Nios II processor with interrupt support (**lab 3**).

You will eventually use these components in **lab 4** to implement the stopwatch.

In this first lab, you will implement a **timer** with an interface that is compatible with the memory bus you implemented last semester in ArchOrd. For more details about the bus, please refer to Section 3 of the **Register files and memories** lab available under the **Additional material** section of the course Moodle.

2 Timer Description



The **timer** has an internal counter that counts down to zero, and whenever it reaches zero, it is immediately reloaded from the `period` register (see following table). An interrupt request (IRQ) is generated whenever the counter reaches zero AND interrupt requests are enabled.

The processor controls the **timer** through the **timer**'s registers. The registers enable the following operations:

- Start and stop the **timer**.

- Enable/disable IRQs.
- Specify the counting mode: count-down once or continuous count-down.
- Specify the **timer** period by writing a value to the `period` register.
- Get the current **timer** state by reading the `status` register.
- Get the current counter value by reading the the `counter` register.

2.1 Register Map

Register	Address	Name	31 . . . 4	3	2	1	0
0	`0x0`	`counter`	Internal counter value				
1	`0x4`	`period`	Timeout period				
2	`0x8`	`control`	*Reserved*	START	STOP	ITO	CONT
3	`0xC`	`status`	*Reserved*			TO	RUN

Reserved bits are unused. Writing them has no effect and reading them returns 0.

2.1.1 Status Register

- The **TO** (*timeout*) bit is set to 1 when the counter reaches zero. The **TO** bit stays 1 until it is explicitly cleared by *software*. Write zero to the status register to clear the **TO** bit. Writing 1 to the **TO** bit has no effect.
- The **RUN** bit is 1 when the counter is running; otherwise this bit is 0. It is a *read only* bit.

2.1.2 Control Register

- If the **ITO** bit is set to 1, the **timer** generates an IRQ when the counter reaches zero (i.e., when **TO** is set to 1); when the **ITO** bit is 0, the **timer** does not generate IRQs.
- The **CONT** (*continuous*) bit determines how the internal counter behaves when it reaches zero. If the **CONT** bit is set to 1, the counter runs continuously until it is stopped by the STOP bit. If **CONT** is 0, the counter stops after it reaches zero. When the counter reaches zero, it reloads with the value stored in the `period` register, regardless of the **CONT** bit.
- By setting the **START** bit, the counter starts counting. The **START** bit is a *write only* bit (i.e., when reading the control register, this bit will always be read as 0). If the **timer** is stopped, writing a 1 to the **START** bit causes the timer to restart counting from the number currently held in its counter. If the **timer** is already running, writing 1 in **START** has no effect. Writing 0 in the **START** bit has no effect.
- By setting the **STOP** bit, the counter stops counting. The **STOP** bit is a *write only* bit. If the timer is already stopped, writing 1 to STOP has no effect. Writing 0 to the stop bit has no effect.

2.1.3 Period Register

The `period` register maintains the timeout period value. The internal counter is loaded with the `period` register value whenever one of the following event occurs:

- A write operation to the `period` register.
- The internal counter reaches 0.

The timer's actual period is one cycle greater than the value stored in the `period` register, because the counter stays zero for one clock cycle before reloading the period value. Writing to the `period` register stops the internal counter.

2.1.4 Counter Register

The current value of the counter is returned by reading the `counter` register. This register is a *read only* register.

2.2 Interrupt Behaviour

The **timer** sets the **irq** output to 1 whenever the **TO** bit of the status register is set to 1 and the **ITO** bit of the control register is 1. Clearing the **TO** bit of the status register will acknowledge the IRQ and set the **irq** output to 0.

3 Exercise

- Download the template of the **timer** lab. This template contains two main files: `timer.vhd` and `timer_tb.vhd`.
  - In the file `timer.vhd` you will write the VHDL source code of the **timer** described in Section 2. The entity is already given to match with the declaration used in the *testbench*.
  - The file `timer_tb.vhd` is a *testbench* that verifies the basic requirements of the **timer**. You should use this file to simulate your design and verify its functionality.
- Implement the **timer** according to its description.
  - Refer to section 3 of the **Register files and memories** lab available under the **Additional material** section of the course Moodle for the expected read and write timing diagrams. *Remember that the read process has one cycle latency: the **addr**, **read** and **cs** signals should go through flip-flops before selecting the output data.*
  - When set to 0, the **reset_n** input signal should initialize every register to 0 *asynchronously*.
  - The **irq** output signal can be generated from a single logic gate, how?
  - The counting process typically requires a small state machine. To know the current state of the timer, do you need to store the current state in additional registers or can you directly use the information stored in the **timer** registers?
- Run the *testbench* with ModelSim to check your design.
  - Start a simulation of `timer_tb`.
  - Add the timer signals to the wave.
  - Type `run -all` in the command line.
  - The messages will give you additional information about potential errors.