



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 8

Timers

1. Introduction

Hardware timers are common peripherals found in microcontrollers and SoCs. They are typically used to generate precise timing. Timers increment or decrement a counter at a fixed frequency, which is often configurable, and then interrupt the processor when the counter reaches zero or a predefined value. More sophisticated timers can also perform other functions, such as generating pulse-width modulated (PWM) waveforms to control the speed of a motor or the brightness of a light.

In this lab, using a similar structure to that of previous labs, we first describe the high-level specification of the timer included in the RVfpgaEL2 System and then explain its low-level implementation. Finally, some exercises are proposed that show how to use the timer.

2. High-Level Specification of the Timer Included in RVfpgaEL2

In this section, we first analyse the high-level specification of the timer used in the RVfpgaEL2 System and then we propose one exercise that uses this peripheral.

A. Timer high-level specification

The timer module used in the RVfpgaEL2 System has been obtained from OpenCores (<https://opencores.org/projects/ptc>). If you download the package, a document is provided that describes the high-level specification of the module (and which we provide at: *[RVfpgaBasysPath]/src/VeeRwolf/Peripherals/ptc/docs/ptc_spec.pdf*). We summarize the main operation and features of the timer module here; however, the complete information can be found in the above document.

The timer module has the following main features:

- Uses a Wishbone Interconnection
- 32-bit counter/timer facility
- Single-run or continuous run of PWM/Timer/Counter (PTC)
- Programmable PWM (Pulse-width modulation) mode
- System clock and external clock sources for timer functionality
- HI/LO Reference and Capture registers
- Three-state control for PWM output driver
- PTC functionalities can cause an interrupt to the CPU

Section 4 of the timer module specification document describes the control and status registers available inside the timer module, each of which is assigned to a different address (see Table 1). The base address for the timer in the RVfpgaEL2 System is **0x80001200**.

Table 1. Timer registers

Name	Address	Width	Access	Description
RPTC_CNTR	0x80001200	1-32	R/W	Main PTC counter
RPTC_HRC	0x80001204	1-32	R/W	PTC HI Reference/Capture register
RPTC_LRC	0x80001208	1-32	R/W	PTC LO Reference/Capture register
RPTC_CTRL	0x8000120C	9	R/W	Control register

The RPTC_CNTR register is the actual counter register, and it is incremented at every counter/timer clock cycle. The RPTC_CTRL register is used for controlling the timer module;

Table 2 shows the function of each of its bits. `RPTC_HRC` and `RPTC_LRC` are used as reference/capture registers.

Table 2. RPTC_CTRL bits

Bit	Access	Reset	Name & Description
0	R/W	0	EN When set, <code>RPTC_CNTR</code> increments.
1	R/W	0	ECLK Selects the clock signal: external clock, through <code>ptc_ecgt</code> (1), or system clock (0).
2	R/W	0	NEC Used for selecting the negative/positive edge and low/high period of the external clock (<code>ptc_ecgt</code>).
3	R/W	0	OE Enables PWM output driver.
4	R/W	0	SINGLE When set, <code>RPTC_CNTR</code> is not incremented after it reaches value equal to the <code>RPTC_LRC</code> value. When cleared, <code>RPTC_CNTR</code> is restarted after it reaches value in the <code>RPTC_LCR</code> register.
5	R/W	0	INTE When set, PTC asserts an interrupt when <code>RPTC_CNTR</code> value is equal to the value of <code>RPTC_LRC</code> or <code>RPTC_HRC</code> . When the signal is cleared, interrupts are masked.
6	R/W	0	INT When read, this bit represents pending interrupt. When it is set, an interrupt is pending. When this bit is written with '1', interrupt request is cleared.
7	R/W	0	CNTRRST When set, <code>RPTC_CNTR</code> is reset. When cleared, normal operation of the counter occurs.
8	R/W	0	CAPTE When set, <code>RPTC_CNTR</code> is captured into <code>RPTC_LRC</code> or <code>RPTC_HRC</code> registers. When cleared, capture function is masked.

TASK: Locate the declaration of registers `RPTC_CNTR`, `RPTC_HRC`, `RPTC_LRC` and `RPTC_CTRL` in the timer module, as well as the definition of their addresses (0x80001200, 0x80001204, 0x80001208 and 0x8000120C respectively). The timer module is available inside folder `[RVfpgaBasysPath]/src/VeeRwolf/Peripherals/ptc`.

The timer can operate in different modes (we next briefly describe the modes that you will use in this lab; see Section 3 of the timer module specification document for more details):

- **Timer/Counter mode:** In this mode, the system clock or external clock reference increments register `RPTC_CNTR` if the counter is enabled (`RPTC_CTRL[EN] = 1`). When `RPTC_CNTR` equals the `RPTC_LRC`, if `RPTC_CTRL[INTE]` is set, `RPTC_CTRL[INT]` goes high.
- **PWM mode:** A pulse width modulated (PWM) signal is a method for generating an analog signal using a digital source. A PWM signal consists of two values that define its behaviour: the *duty cycle* and *frequency*. The duty cycle describes the amount of time the signal is high as a percentage of the total time of it takes to complete one cycle. The frequency is how often that cycle repeats. By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices. For example, a

signal with a 50% duty cycle (half the cycle time it is high) and a high voltage of 3.3 V would appear to an analog load as 1.67 V (the average voltage across the cycle). The same signal with a 33% duty cycle would appear to be 1.1V. To operate in PWM mode, `RPTC_CTRL[OE]` should be set. Registers `RPTC_HRC` and `RPTC_LRC` should be set with the value of high and low periods of the PWM output signal: the PWM signal should go high `RPTC_HRC` clock cycles after reset (of `RPTC_CNTR`); and the PWM signal should go low `RPTC_LRC` clock cycles after reset (of the `RPTC_CNTR`).

3. Exercise

Exercise 1. Write a program that displays an ascending count on the 8-digit 7-segment displays. The value should change about once per second and, for creating this delay, you must use the timer module.

- a. First, write the program in RISC-V assembly language and run it on the Basys 3 Board.
- b. Then, perform a simulation in the different RVfpgaEL2 tools.
- c. Now write the program in C and run it on the Basys 3 Board.
- d. Simulate your C program as in part (b) for the RISC-V assembly program.

4. Timer Low-Level Implementation

In this section, we first describe the low-level implementation of the timer module in the RVfpga System and then propose some exercises where you will first modify the module and then use it in a program for controlling the tri-colour LEDs available on the Basys 3 Board.

A. Low-level implementation of the timer

Similarly to the scheme that we followed in previous labs, we divide the analysis of the timer module into phases.

1. Integration of the new module in the VeeRwolfX SoC (left shadowed region in Figure 1)
2. Connection between the new module and the VeeR EL2 Core (right shadowed region in Figure 1).

Note that, as opposed to previous labs, this peripheral (the timer) is not connected physically to the Basys 3 Board. The timer is internal to VeeRwolfX.

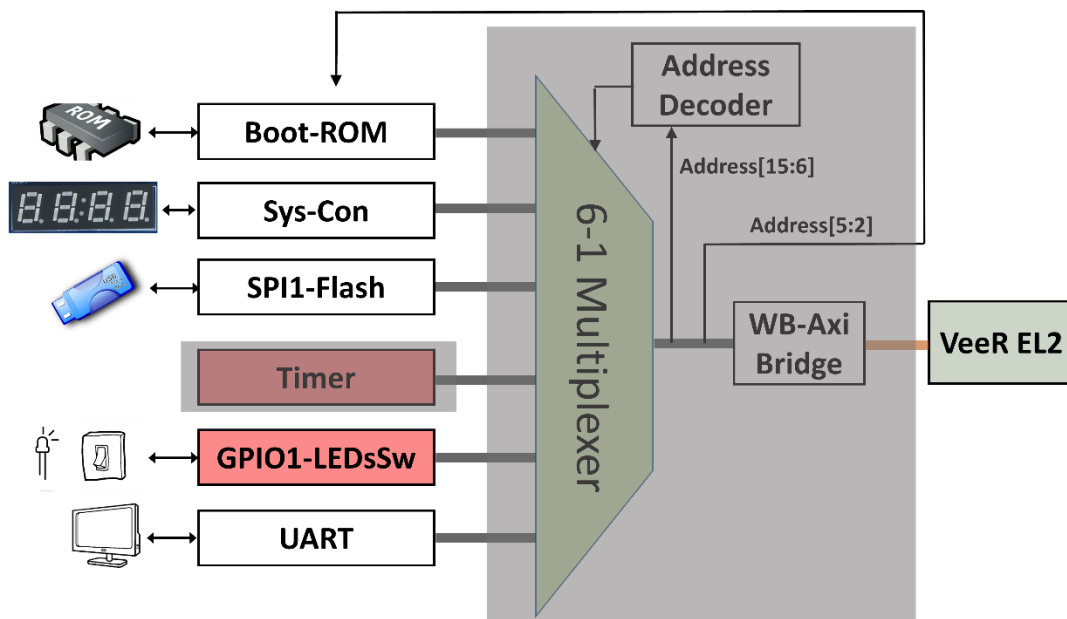


Figure 1. Timer module analysis in 2 phases

i. Integration of the timer module in the SoC

In module **veerwolf_core** ([RVfpgaBasysPath]/src/VeeRwolf/veerwolf_core.v) the timer module is instantiated (see Figure 2).

```
// PTC
wire      ptc_irq;

ptc_top timer_ptc(
    .wb_clk_i      (clk),
    .wb_rst_i      (wb_rst),
    .wb_cyc_i      (wb_m2s_ptc_cyc),
    .wb_adr_i      ({2'b0,wb_m2s_ptc_adr[5:2],2'b0}),
    .wb_dat_i      (wb_m2s_ptc_dat),
    .wb_sel_i      (4'b1111),
    .wb_we_i      (wb_m2s_ptc_we),
    .wb_stb_i      (wb_m2s_ptc_stb),
    .wb_dat_o      (wb_s2m_ptc_dat),
    .wb_ack_o      (wb_s2m_ptc_ack),
    .wb_err_o      (wb_s2m_ptc_err),
    .wb_inta_o     (ptc_irq),
    // External PTC Interface
    .gate_clk_pad_i (),
    .capt_pad_i    (),
    .pwm_pad_o     (),
    .oen_padoen_o  ()
);
```

Figure 2. Integration of the timer module (file veerwolf_core.v).

As usual, the interface of the module can be divided in two blocks: Wishbone signals (Table 3) and External I/O signals (Table 4). The Wishbone signals allow the VeeR EL2 Core to communicate with the timer using a controller/peripheral model. The External I/O signals, connect the timer module with external devices; for example, `pwm_pad_o` provides the PWM output signal when operating in the PWM mode described above (you will have to use this signal in Exercise 2 to connect the timer modules with the tri-colour LEDs).

Table 3. Wishbone Signals

Port	Width	Direction	Description
wb_cyc_i	1	Inputs	Indicates valid bus cycle (core select)
wb_adr_i	15	Inputs	Address inputs
wb_dat_i	32	Inputs	Data inputs
wb_dat_o	32	Outputs	Data outputs
wb_sel_i	4	Inputs	Indicates valid bytes on data bus (during valid cycle, this signal must be 0xf)
wb_ack_o	1	Output	Acknowledgment output (indicates normal transaction termination)
wb_err_o	1	Output	Error acknowledgment output (indicates an abnormal transaction termination)
wb_rty_o	1	Output	Not used
wb_we_i	1	Input	Write transaction when asserted high
wb_stb_i	1	Input	Indicates valid data transfer cycle
wb_inta_o	1	Output	Interrupt output

Table 4. External I/O signals

Port	Width	Direction	Description
gate_clk_pad_i	1	Input	External clock / Gate input
capt_pad_i	1	Input	Capture input
pwm_pad_o	1	Output	PWM output
oen_padoen_o	1	Output	PWM output driver enable (for three-state or open-drain driver)

As shown in Figure 2, bits [5:2] of the address provided by the core in the Wishbone bus signal (`wb_m2s_ptc_adr[5:2]`) are used for selecting one among the 4 available registers (Memory Mapped I/O). Thus, we can access register `RPTC_CNTR` at address `0x80001200`, register `RPTC_HRC` at address `0x80001204`, register `RPTC_LRC` at address `0x80001208`, and register `RPTC_CTRL` at address `0x8000120C`.

ii. Connection between the timer and the VeeR EL2 Core

As described in Lab 6, the device controllers are connected to the VeeR EL2 Core using a multiplexer (see Figure 3). Remember that the 6:1 multiplexer is instantiated in file `[RVfpgaBasysPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.v`. Then, the `wb_intercon` module is instantiated in file `[RVfpgaBasysPath]/src/VeeRwolf/Interconnect/WishboneInterconnect/wb_intercon.vh`. This latter file is included in the `veerwolf_core` module located here: `[RVfpgaBasysPath]/src/VeeRwolf/veerwolf_core.v`.

The multiplexer selects which peripheral to read or write, connecting the CPU (`wb_io_*` signals) with the Wishbone Bus of one peripheral, depending on the address. For example, if the address generated by the CPU is in the range `0x80001200-0x8000123F`, the timer module is selected, and thus signals `wb_io_*` are connected with signals `wb_ptc_*`.

```

wb_mux
#(.num_slaves (6),
.MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001200, 32'h00001400, 32'h00002000}),
.MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00}))
wb_mux_io
(.wb_clk_i (wb_clk_i),
.wb_rst_i (wb_rst_i),
.wbm_adr_i (wb_io_adr_i),
.wbm_dat_i (wb_io_dat_i),
.wbm_sel_i (wb_io_sel_i),
.wbm_we_i (wb_io_we_i),
.wbm_cyc_i (wb_io_cyc_i),
.wbm_stb_i (wb_io_stb_i),
.wbm_cti_i (wb_io_cti_i),
.wbm_bte_i (wb_io_bte_i),
.wbm_dat_o (wb_io_dat_o),
.wbm_ack_o (wb_io_ack_o),
.wbm_err_o (wb_io_err_o),
.wbm_rty_o (wb_io_rty_o),
.wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o}),
.wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o}),
.wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o}),
.wbs_we_o ({wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o}),
.wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o}),
.wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o}),
.wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o}),
.wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o}),
.wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i}),
.wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i}),
.wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i}),
.wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i}));

```

Figure 3. 6:1 multiplexer that selects the peripheral connected with the CPU (file *wb_intercon.v*)