



**Imagination大学计划**

# **RVfpga实验8**

## **定时器**

## 1. 简介

硬件定时器是单片机和SoC中常见的外设，通常用于生成精确时序。定时器以固定频率（该频率通常是可配置的）递增或递减计数器，然后在计数器达到零或预定义值时中断处理器。更复杂的定时器还可以执行其他功能，例如生成脉宽调制（Pulse-Width Modulated, PWM）波形以控制电机转速或灯光亮度。

本实验的结构编排与之前的实验类似，我们首先介绍RVfpga系统中所含定时器的高级规范，然后解释说明其底层实现。此外，还提供了基础练习和高级练习，以展示如何使用和修改定时器。

## 2. RVfpga系统中所含定时器的高级规范

在本部分中，我们首先分析RVfpga系统中使用的定时器的高级规范，然后提供一个使用该外设的练习。

### A. 定时器高级规范

已从OpenCores（<https://opencores.org/projects/ptc>）获取RVfpga系统中使用的定时器模块。如果下载教学包，其中会随附一个文档，用于描述该模块的高级规范（此文档位于以下位置：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/ptc/docs/ptc\_spec.pdf*）。我们在此总结了定时器模块的主要操作和特性；有关完整的信息，请参见上述文档。

定时器模块的主要特性如下：

- 使用Wishbone互连
- 32位计数器/定时器模式
- 单次运行或连续运行PWM/定时器/计数器（PWM/Timer/Counter, PTC）
- 可编程PWM（脉宽调制）模式
- 系统时钟和外部时钟源，用于定时器功能
- 高电平/低电平参考和捕捉寄存器
- PWM输出驱动器的三态控制
- PTC功能可向CPU发出中断

定时器模块规范文档的第4部分介绍了定时器模块内部可用的控制和状态寄存器，每个寄存器都分配到不同的地址（参见表1）。RVfpga系统中定时器的基址为0x80001200。

**表1. 定时器寄存器**

名称	地址	宽度	访问	说明
RPTC_CNTR	0x80001200	1-32	R/W	主PTC计数器
RPTC_HRC	0x80001204	1-32	R/W	PTC高电平参考/捕捉寄存器
RPTC_LRC	0x80001208	1-32	R/W	PTC低电平参考/捕捉寄存器
RPTC_CTRL	0x8000120C	9	R/W	控制寄存器

RPTC\_CNTR寄存器是实际的计数器寄存器，计数器/定时器每个时钟周期递增一次。RPTC\_CTRL寄存器用于控制定时器模块；表2显示了其中每个位的功能。RPTC\_HRC和RPTC\_LRC用作参考/捕捉寄存器。

**表2. RPTC\_CTRL位**

位	访问	复位	名称和说明
0	R/W	0	<b>EN</b> 置1时，RPTC_CNTR递增。
1	R/W	0	<b>ECLK</b> 选择时钟信号：外部时钟（通过 <code>ptc_ecgt</code> ）（1）或系统时钟（0）。
2	R/W	0	<b>NEC</b> 用于选择外部时钟（ <code>ptc_ecgt</code> ）的下降沿/上升沿和低电平/高电平周期。
3	R/W	0	<b>OE</b> 使能PWM输出驱动器。
4	R/W	0	<b>SINGLE</b> 置1时，RPTC_CNTR在达到RPTC_LRC值后不递增。清零时，RPTC_CNTR在达到RPTC_LCR寄存器中的值后重新启动。
5	R/W	0	<b>INTE</b> 置1时，当RPTC_CNTR值等于RPTC_LRC或RPTC_HRC的值时，PTC会将中断置为有效。信号清零时，中断将被屏蔽。
6	R/W	0	<b>INT</b> 读取时，该位表示待处理的中断。置1时，表示有一个中断待处理。当该位写入1时，中断请求将被清除。
7	R/W	0	<b>CNTRRST</b> 置1时，将复位RPTC_CNTR。清零时，计数器将正常工作。
8	R/W	0	<b>CAPTE</b> 置1时，RPTC_CNTR将被捕捉到RPTC_LRC或RPTC_HRC寄存器中。清零时，捕捉功能将被屏蔽。

**任务：**在定时器模块中查找寄存器RPTC\_CNTR、RPTC\_HRC、RPTC\_LRC和RPTC\_CTRL的声明，以及这些寄存器的地址定义（分别为0x80001200、0x80001204、0x80001208和0x8000120C）。定时器模块位于文件夹  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/ptc内。

定时器可以在不同的模式下运行（接下来，我们简要介绍本实验中将使用的模式；有关更多详细信息，请参见定时器模块规范文档的第3部分）：

- **定时器/计数器模式：**在此模式下，如果使能了计数器（RPTC\_CTRL[EN] = 1），则系统时钟或外部时钟参考会递增寄存器RPTC\_CNTR。当RPTC\_CNTR等于RPTC\_LRC时，如果RPTC\_CTRL[INTE]置1，则RPTC\_CTRL[INT]变为高电平。
- **PWM模式：**脉宽调制（PWM）信号是一种使用数字源生成模拟信号的方法。PWM信号由两个定义其行为值组成：**占空比**和**频率**。占空比描述信号为高电平的时间占完成一个周期所用总时间的百分比。频率是周期重复的频率。当器件上电后，如果以足够快的速率和一定的占空比循环开关数字信号，输出会表现为恒压模拟信号。例如，占空比为50%（一半的周期时间为高电平）的3.3 V高压信号相当于1.67 V（整个周期的平均电压）的模拟负载。相同的信号，占空比为33%时则相当于1.1V。要在PWM模式下运行，RPTC\_CTRL[OE]应置1。寄存器RPTC\_HRC和RPTC\_LRC应分别设为PWM输出信号的高电平周期和低电平周期的值：（RPTC\_CNTR）复位后，PWM信号应变为高电平RPTC\_HRC时钟周期；（RPTC\_CNTR）复位后，PWM信号应变为低电平

RPTC\_LRC时钟周期。

### 3. 基本练习

**练习1.** 编写一个程序，在8位7段显示屏上显示升序计数。该值应大约每秒钟改变一次，一秒延时需用定时器模块生成。

- a. 首先，用RISC-V汇编语言编写程序并在Nexys A7开发板上运行该程序。
- b. 然后，使用相同程序在Verilator中进行仿真。可添加以下信号：系统时钟、用于存储在8位7段显示屏中显示的值的处理器寄存器，以及定时器寄存器RPTC\_CNTR、RPTC\_LRC、RPTC\_HRC和RPTC\_CRTL。
- c. 现在，用C语言编写程序并在Nexys A7开发板上运行该程序。
- d. 与RISC-V汇编程序的（b）部分一样，在Verilator中仿真您的C程序。

### 4. 定时器底层实现

在本部分中，我们首先介绍RVfpga系统中定时器模块的底层实现，然后提供一些练习。在练习中，我们将首先修改该模块，然后在程序中使用该模块来控制Nexys A7开发板上的三色LED。

#### A. 定时器的底层实现

与先前实验中遵循的方案类似，我们分阶段来进行定时器模块分析。

1. 在SweRVofX SoC中集成新模块（图1中左侧阴影区域）
2. 新模块与SweRV EH1内核之间的连接（图1中右侧阴影区域）。

请注意，此外设（定时器）未物理连接到Nexys A7开发板，这一点与先前的实验有所不同。定时器在SweRVofX内部。

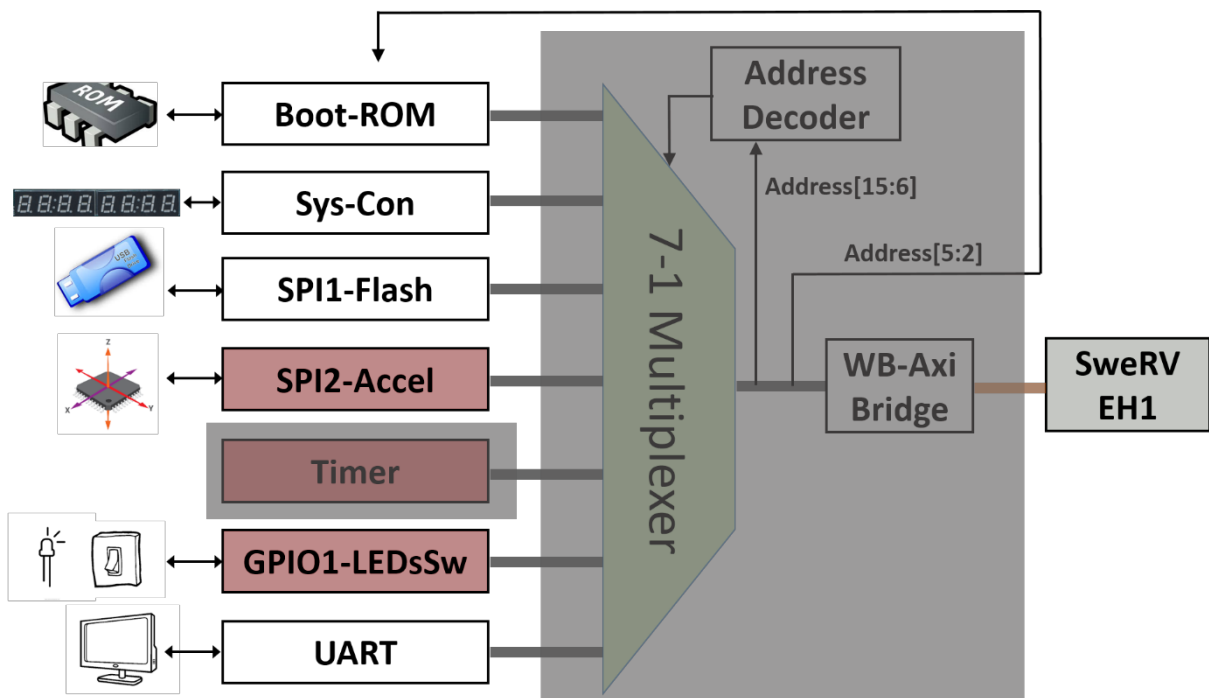


图1. 定时器模块分析分2个阶段

#### i. 定时器模块在SoC中的集成

在模块 **swervolf\_core** (`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`) 的第361-379行，定时器模块实例化（参见图2）。

```

358 // PTC
359 wire      ptc_irq;
360
361 ptc_top timer_ptc(
362     .wb_clk_i      (clk),
363     .wb_rst_i      (wb_rst),
364     .wb_cyc_i      (wb_m2s_ptc_cyc),
365     .wb_adr_i      ({2'b0,wb_m2s_ptc_adr[5:2],2'b0}),
366     .wb_dat_i      (wb_m2s_ptc_dat),
367     .wb_sel_i      (4'b1111),
368     .wb_we_i      (wb_m2s_ptc_we),
369     .wb_stb_i      (wb_m2s_ptc_stb),
370     .wb_dat_o      (wb_s2m_ptc_dat),
371     .wb_ack_o      (wb_s2m_ptc_ack),
372     .wb_err_o      (wb_s2m_ptc_err),
373     .wb_inta_o     (ptc_irq),
374     // External PTC Interface
375     .gate_clk_pad_i (),
376     .capt_pad_i    (),
377     .pwm_pad_o     (),
378     .oen_padoen_o  ()
379 );

```

图2. 定时器模块的集成（文件 **swervolf\_core.v**）。

模块的接口依旧可以分为两个模块：Wishbone信号（表3）和外部I/O信号（表4）。Wishbone信号允许SweRV EH1内核使用控制器/外设设备与定时器进行通信。外部I/O信号将定时器模块与外部设备连接；例如，在上述PWM模式下运行时，`pwm_pad_o`会提供PWM输出信号（在练习2中需使用此信号将定时器模块与三色LED连接）。

**表3. Wishbone信号**

端口	宽度	方向	说明
wb_cyc_i	1	输入	指示有效的总线周期（内核选择）
wb_adr_i	15	输入	地址输入
wb_dat_i	32	输入	数据输入
wb_dat_o	32	输出	数据输出
wb_sel_i	4	输入	指示数据总线上的有效字节（在有效周期内，此信号必须为0xf）
wb_ack_o	1	输出	应答输出（指示正常事务终止）
wb_err_o	1	输出	错误应答输出（指示异常事务终止）
wb_rty_o	1	输出	未使用
wb_we_i	1	输入	置为高电平时写事务
wb_stb_i	1	输入	指示有效的数据传输周期
wb_inta_o	1	输出	中断输出

**表4. 外部I/O信号**

端口	宽度	方向	说明
gate_clk_pad_i	1	输入	外部时钟/门控输入
capt_pad_i	1	输入	捕捉输入
pwm_pad_o	1	输出	PWM输出
oen_padoen_o	1	输出	PWM输出驱动器使能 (用于三态或漏极开路驱动器)

如图2的第365行所示，Wishbone总线信号中由内核提供的地址的位[5:2]

（`wb_m2s_ptc_adr[5:2]`）用于从4个可用的寄存器中选择1个（存储器映射I/O）。因此，我们可以访问地址0x80001200处的寄存器RPTC\_CNTR、地址0x80001204处的寄存器RPTC\_HRC、地址0x80001208处的寄存器RPTC\_LRC和地址0x8000120C处的寄存器RPTC\_CTRL。

## ii. 定时器与SweRV EH1内核的连接

如先前的实验中所述，设备控制器通过多路开关与SweRV EH1内核连接（图1）。请记住，7:1多路开关（图3）在文件

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v`中实现，该文件在文件

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`的第104-205行实例化。后一个文件包含在`swervolf_core`模块的第168行，该模块位于：

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`。

```

108 wb_mux
109 #(.num_slaves (7),
110 .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
111 .MATCH_MASK ({32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0}))
112 wb_mux_io
113 (.wb_clk_i (wb_clk_i),
114 .wb_rst_i (wb_rst_i),
115 .wbm_adr_i (wb_io_adr_i),
116 .wbm_dat_i (wb_io_dat_i),
117 .wbm_sel_i (wb_io_sel_i),
118 .wbm_we_i (wb_io_we_i),
119 .wbm_cyc_i (wb_io_cyc_i),
120 .wbm_stb_i (wb_io_stb_i),
121 .wbm_cti_i (wb_io_cti_i),
122 .wbm_bte_i (wb_io_bte_i),
123 .wbm_dat_o (wb_io_dat_o),
124 .wbm_ack_o (wb_io_ack_o),
125 .wbm_err_o (wb_io_err_o),
126 .wbm_rty_o (wb_io_rty_o),
127 .wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o}),
128 .wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o}),
129 .wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o}),
130 .wbs_we_o ({wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o}),
131 .wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o}),
132 .wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o}),
133 .wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o}),
134 .wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o}),
135 .wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i}),
136 .wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i}),
137 .wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i}),
138 .wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i}));
139
140 endmodule

```

CPU/Controller Signals

Peripheral Signals

图3. 选择与CPU连接的外设的7-1多路开关（文件**wb\_intercon.v**）

多路开关选择要读取或写入哪个外设，根据地址（第110-111行）将CPU（**wb\_io\_\***信号 – 图3的第115-126行）与一个外设的Wishbone总线（图3的第127-138行）连接。例如，如果CPU生成的地址在0x80001200-0x8000123F范围内，则选择定时器模块，从而将信号**wb\_io\_\***与信号**wb\_ptc\_\***连接。

## 5. 高级练习

**练习2.** 修改RVfpgaNexys以将定时器的PWM输出信号（**pwm\_pad\_o**）连接到Nexys A7开发板上的两个三色LED之一。建议将此新功能添加到在实验6和7中修改并更新后的RVfpgaNexys系统中。

- Digilent提供了以下有关Nexys A7开发板上的三色LED的信息：  
<https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual>
- 综上所述，开发板包含两个三色LED。每个三色LED具有三个输入信号，这些信号驱动三个较小的内部LED的阴极：一个红色、一个蓝色和一个绿色。将其中一个阴极驱动为高电平即可使相应的内部LED点亮。三色LED发出何种颜色取决于当前点亮的内部LED的组合。例如，将红色和蓝色驱动为高电平会发出紫色。Digilent强烈建议在驱动三色LED时使用脉宽调制（PWM）。将任何输入驱动为稳定的逻辑“1”会导致LED的亮度异常。通过确保以不超过50%的占空比来驱动所有三色信号，可以避免这种情况。此外，使用PWM还可以极大地扩展三色LED的潜在调色板。分别将每种颜色的占空比调整到50%和0%之间，会使不同的颜色以不同的强度点亮，从而几乎可以显示任何颜色。
- 基于SweRVofX中已有的定时器模块创建三个新的定时器模块。每种颜色（红色、蓝色和绿色）应由不同的定时器模块驱动，以便可以接收不同的电压。



- 使用以下地址范围将每个新定时器的寄存器映射到存储器：

- i. 定时器2: 0x80001240-0x8000127F
- ii. 定时器3: 0x80001280-0x800012BF
- iii. 定时器4: 0x800012C0-0x800012FF

请注意，在这种情况下，必须将3个新条目添加到选择外设的多路开关（图1）。

- 在修改约束文件时，必须考虑到3种颜色连接到以下开发板引脚：
  - iv. LED16\_B  $\leftrightarrow$  PIN R12
  - v. LED16\_G  $\leftrightarrow$  PIN M16
  - vi. LED16\_R  $\leftrightarrow$  PIN N15

**练习3.** 使用16个开关提供的值实现一个程序，该程序使用新的外设来控制三色LED。按从最右到最左的顺序，依次使用5个开关调整蓝色的占空比，5个开关调整绿色的占空比，5个开关调整红色的占空比。（最左侧的开关将不使用。）

- a. 首先，编写RISC-V汇编程序。
- b. 然后，编写C程序。