Temporizadores

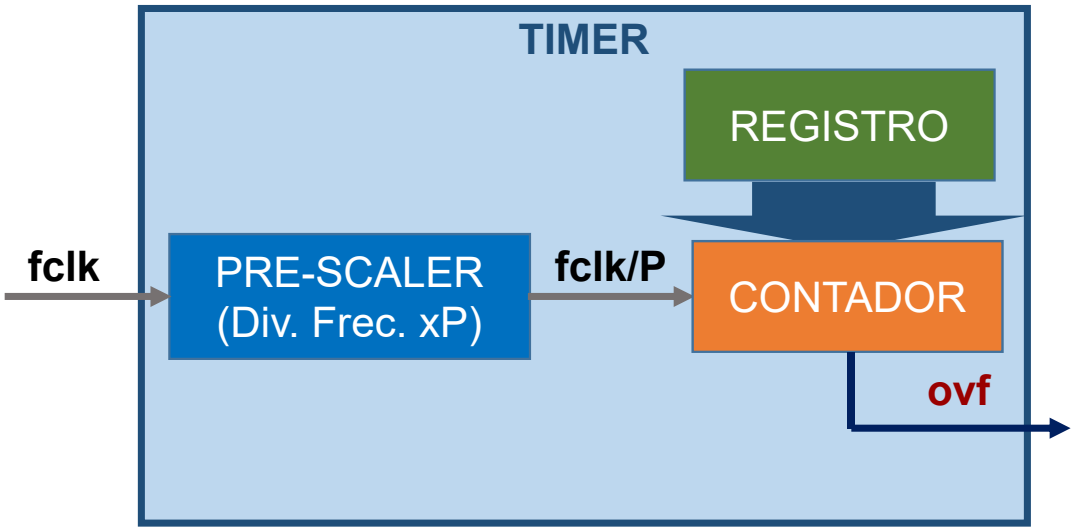
Índice

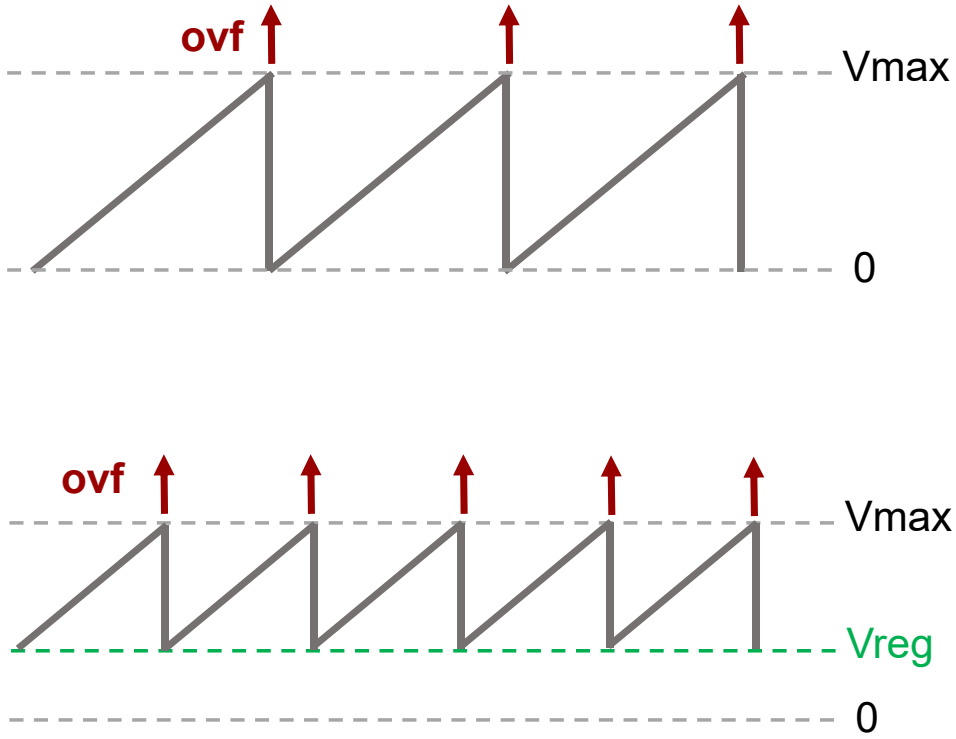
- Temporizadores y watchdog
- Temporizadores en ESP32
- Librería “**esp-hal-timer**” de Arduino
- Funciones del entorno ESP-IDF
- Ejercicios

Temporizadores

- Contador que avisa cuándo ha llegado a su valor máximo/mínimo o a un valor prefijado
- Se utiliza para
  - generar retardos
  - medir tiempo
  - generar interrupciones o eventos periódicos

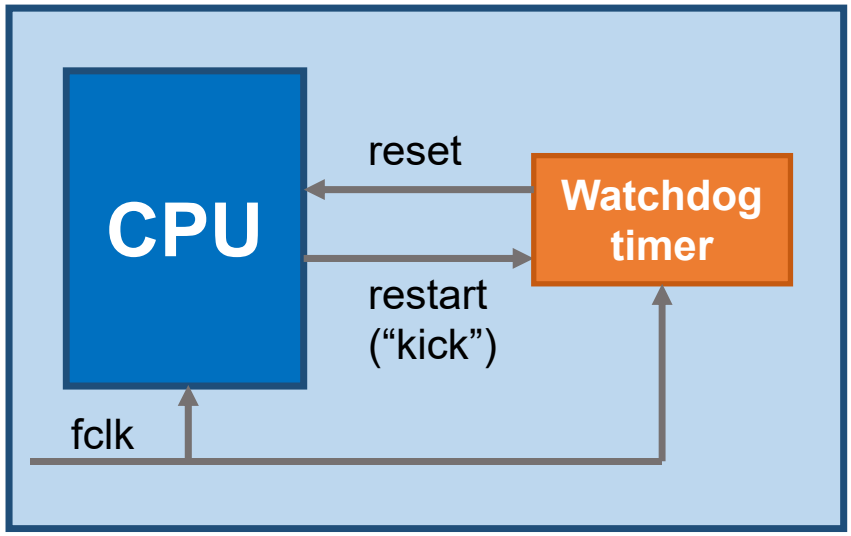
Temporizadores





Temporizador “watchdog”

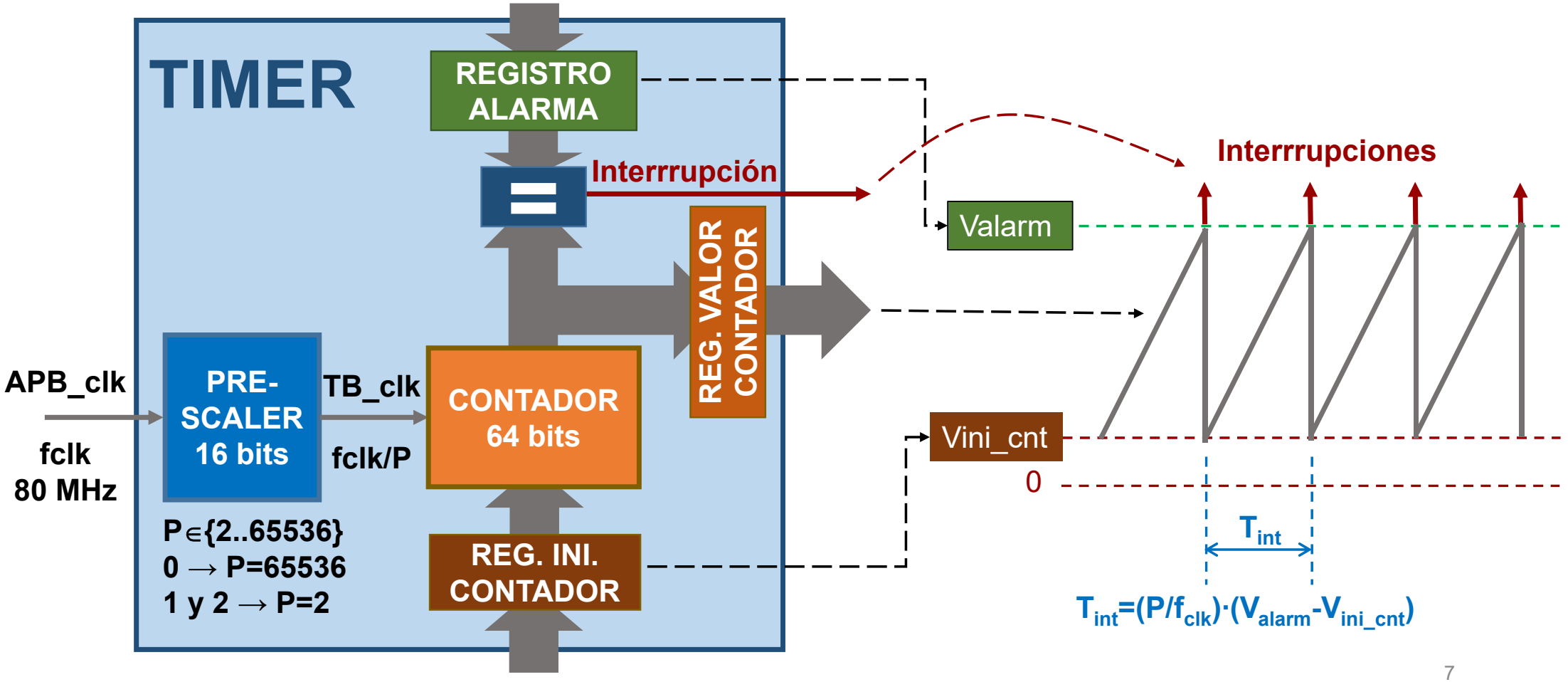
- Muchos sistemas embebidos funcionan sin la presencia de un operador
  - ¿Quién hace un reset si el sistema no está funcionando correctamente?
- **Watchdog**: temporizador utilizado para evitar el mal funcionamiento de la CPU
  - Provoca un reset del sistema cuando vence la cuenta
  - El programador se encargará de inicializar la cuenta en diferentes puntos del programa



Temporizadores en ESP32

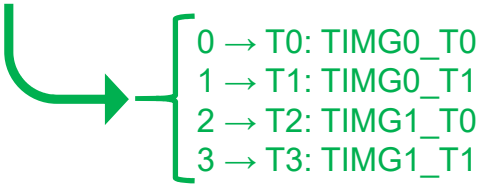
- ESP32 dispone de 4 temporizadores de 64 bits
  - Divididos en 2 módulos de 2 timers
  - Nombres: TIMGn_Tx (n=0,1 y x=0,1)
- Características de los temporizadoress:
  - Prescaler de reloj de 16 bits (de 2 a 65536)
  - Contador de 64 bits
  - Contador configurable up/down
  - Parada y continuación controlable
  - Recarga con la alarma
  - Recarga controlada por SW
  - Generación de interrupciones

Temporizadores en ESP32



Uso de los temporizadores con Arduino

- Disponible con funciones de la librería "**esp-hal-timer**" (declarada por defecto)
- Hay que utilizar la estructura "*hw_timer_t*" para declarar el objeto que identifica el temporizador a utilizar:
  - hw_timer_t * timer = NULL;
- Funciones básicas:
  - hw_timer_t * timerBegin(uint8_t timer_num, uint16_t divider, bool countUp);
  - void timerEnd(hw_timer_t *timer);
  - void timerStart(hw_timer_t *timer);
  - void timerStop(hw_timer_t *timer);
  - void timerWrite(hw_timer_t *timer, uint64_t val);
  - void timerSetAutoReload(hw_timer_t *timer, bool autoreload);
  - void timerSetDivider(hw_timer_t *timer, uint16_t divider);



https://lang-ship.com/reference/unofficial/M5StickC/Functions/esp32-hal-timer/

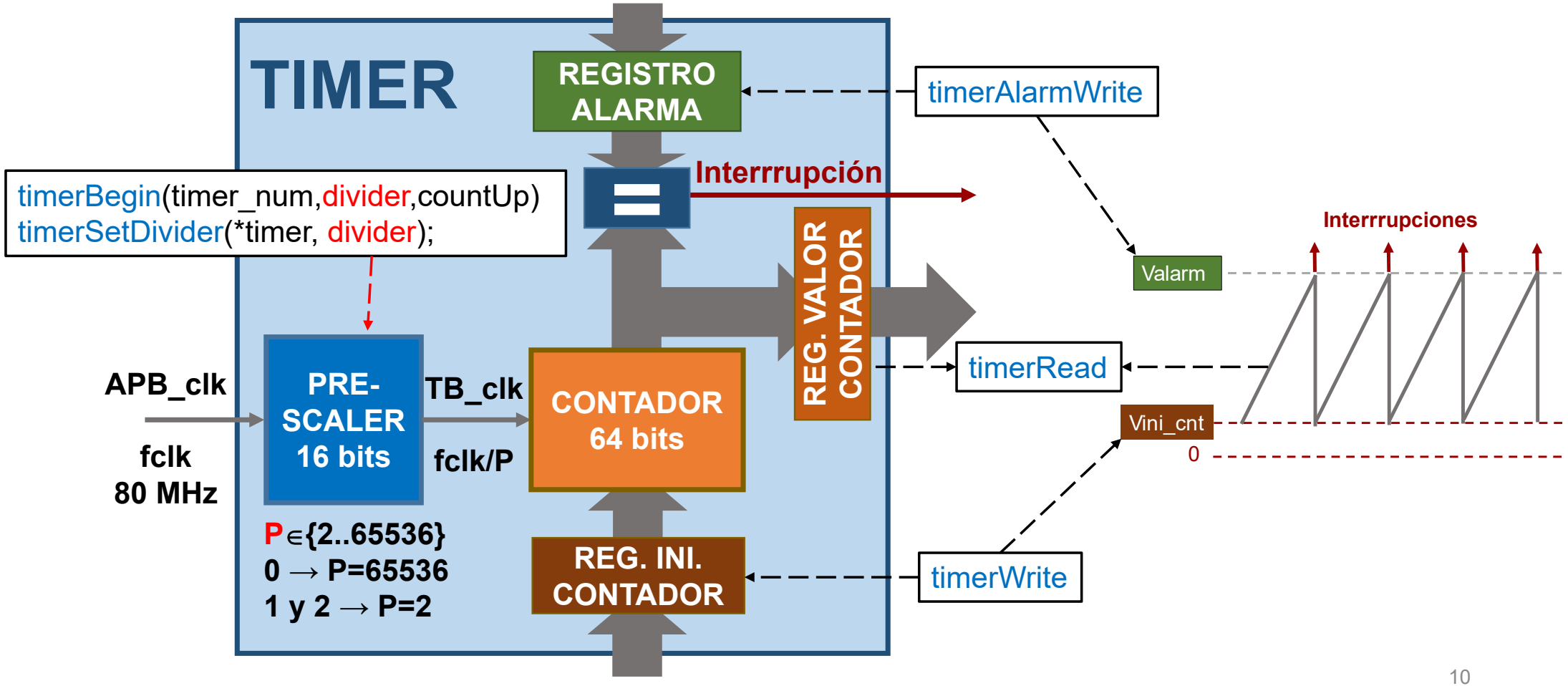
Uso de los temporizadores con Arduino

- Funciones básicas:
  - uint64_t timerRead(hw_timer_t *timer);

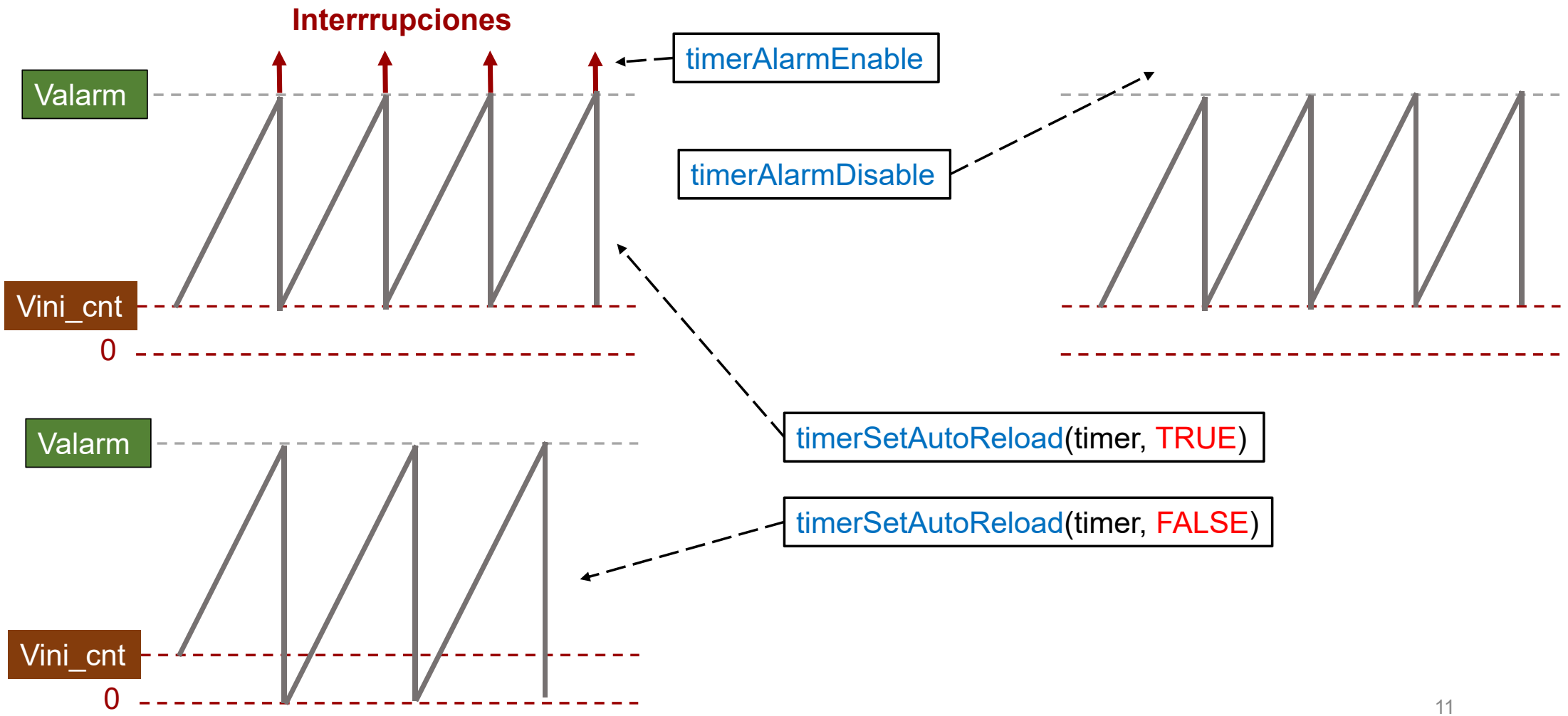
  - void timerAlarmEnable(hw_timer_t *timer);
  - void timerAlarmDisable(hw_timer_t *timer);
  - void timerAlarmWrite(hw_timer_t *timer, uint64_t interruptAt, bool autoreload);

  - void timerAttachInterrupt(hw_timer_t *timer, void (*fn)(void), bool edge);
  - void timerDetachInterrupt(hw_timer_t *timer);

Uso de los temporizadores con Arduino



Uso de los temporizadores con Arduino



Uso de los temporizadores con ESP-IDF

- Es necesario declarar la **librería** driver\“**timer.h**”
- Inicialización del temporizador
  - Identificar el grupo con el tipo *timer_group_t* (nombres: TIMER_GROUP_0, TIMER_GROUP_1)
  - Identificar el timer de un grupo con el tipo *timer_idx_t* (nombres: TIMER_0, TIMER_1)
  - Estructura de configuración *timer_config_t*

```
timer_config_t conf = {
        .alarm_en = TIMER_ALARM_EN;
        .auto_reload = TIMER_AUTOLOAD_EN;
        .counter_dir = TIMER_COUNT_UP;
        .divider = (uint16_t)80;
        .intr_type = TIMER_INTR_LEVEL;
        .counter_en = TIMER_PAUSE};
```

  - Función de inicialización: timer_init()

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/timer.html

Uso de los temporizadores con ESP-IDF

- Funciones para el control de los temporizadores
  - Leer el valor actual del contador: timer_get_counter_value()
  - Escribir el valor inicial de la cuenta: timer_set_counter_value()
  - Detener el contador: timer_pause()
  - Activar el contador: timer_start()
- Control de las alarmas
  - Escribir el valor de la alarma: timer_set_alarm_value()
  - Activar la alarma: timer_set_alarm() (también con timer_init())

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/timer.html

Uso de los temporizadores con ESP-IDF

- Funciones para la gestión de interrupciones
  - Registrar el manejador de la interrupción: timer_isr_register()
  - Activar/desactivar la interrupción de un grupo de timers: timer_group_intr_enable(), timer_group_intr_disable()
  - Activar/desactivar la interrupción de un timer: timer_enable_intr(), timer_disable_intr()
  - Borrar el flag de la interrupción dentro de la ISR
    - Hay que poner a TIMERGN.int_clr_timers.tM = 1 (N es el número de grupo y M el de timer)
  - Leer el status de los flags de interrupción de los timers de un grupo
    - uint32_t intr_status = TIMERG0.int_st_timers.val;
    - Devuelve un 1 si la interrupción la solicita el timer 0, un 2 si la solicita el timer 1 y un 3 si la solicitan ambos.

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/timer.html

Ejercicio 1

Completar el código del programa Ejerc_1.ino para configurar el Timer 0 del ESP32 utilizando funciones de Arduino para que genere interrupciones periódicas de 1 seg y que ejecute la rutina de atención a la interrupción ISR_Timer0. El timer se activará cuando se pulse el botón A y se deshabilitará cuando se pulse el botón B.

Tenga en cuenta que el reloj de entrada al prescaler es de 80 MHz.

Utilice las funciones timerBegin, timerAttachInterrupt, timerAlarmWrite y timerAlarmEnable para configurar el timer y timerStart y timerStop para activarlo y detenerlo.

Ejercicio 2

Completar el código del programa Ejerc_2.ino utilizando funciones de Arduino para configurar los 4 timers del ESP32 para que generen interrupciones periódicas de 1, 2, 0.5 y 1.5 seg y que ejecuten las rutinas de atención a la interrupciones ISR_Timer0, ISR_Timer1, ISR_Timer2 y ISR_Timer3.

Los timers se activarán cuando se pulse el botón A y se deshabilitarán cuando se pulse el botón B.

Ejercicio 3

Analizar el código del programa Ejerc_3.ino que realiza el ejercicio 1 utilizando las funciones del entorno IDF.

Configurar el Timer 0 del ESP32 para que genere interrupciones periódicas de 1 seg y que ejecute la rutina de atención a la interrupción ISR_Timer0.

El timer se activará cuando se pulse el botón A y se deshabilitará cuando se pulse el botón B.

Referencias

- **Tutorial ESP32 Arduino: Timer interrupts:** https://techtutorialsx.com/2017/10/07/esp32-arduino-timer-interrupts/
- **Tutorial ESP32 Timer & Multiple Timer & Changing Timer:** https://www.youtube.com/watch?v=LONGI_JcwEQ
- **ESP-IDF : Using timers in ESP32 :** http://icircuit.net/esp-idf-using-timers-esp32/2050
- **ESP-IDF Programming Guide, API Reference\Peripherals\Timer:** https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/timer.html

3.2.2 Watchdog Timers

The ESP32 has three watchdog timers: one in each of the two timer modules (called the Main Watchdog Timer, or MWDT) and one in the RTC module (called the RTC Watchdog Timer, or RWDT). These watchdog timers are intended to recover from an unforeseen fault, causing the application program to abandon its normal sequence. A watchdog timer has four stages. Each stage may take one of three or four actions upon the expiry of its programmed time period, unless the watchdog is fed or disabled. The actions are: interrupt, CPU reset, core reset, and system reset. Only the RWDT can trigger the system reset, and is able to reset the entire chip, including the RTC itself. A timeout value can be set for each stage individually.
During flash boot the RWDT and the first MWDT start automatically in order to detect, and recover from, booting problems.

The ESP32 watchdogs have the following features:
• Four stages, each of which can be configured or disabled separately
Espressif Systems 15 ESP32 Datasheet V2.1
3. FUNCTIONAL DESCRIPTION
• Programmable time period for each stage
• One of three or four possible actions (interrupt, CPU reset, core reset, and system reset) upon the expiry of each stage
• 32-bit expiry counter
• Write protection to prevent the RWDT and MWDT configuration from being inadvertently altered
• SPI flash boot protection
If the boot process from an SPI flash does not complete within a predetermined time period, the watchdog will reboot the entire system.