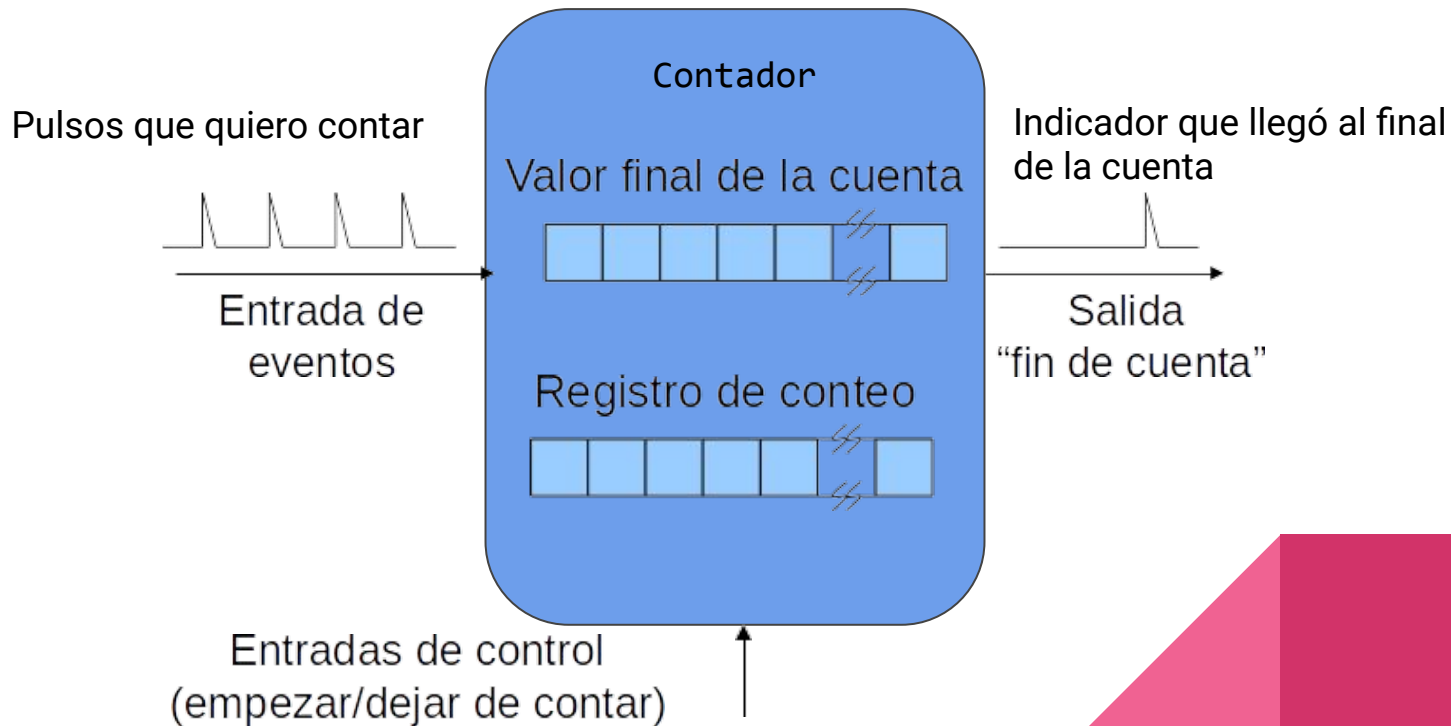


# Contadores y Timers - Systick

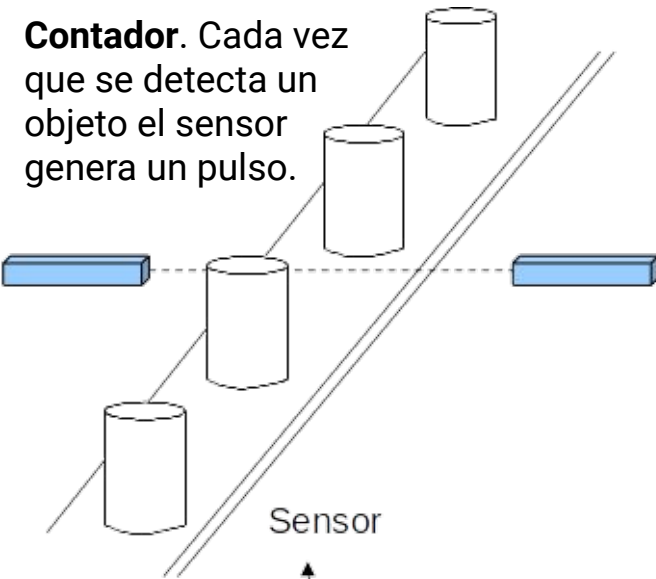
Informática II  
R2004 - 2021

# ¿Cómo funciona un contador?

Por medio de registros disponibles en los periféricos que cumplen el fin de Temporizar/Contar podemos establecer qué valor final de cuenta queremos y llevar la cuenta.



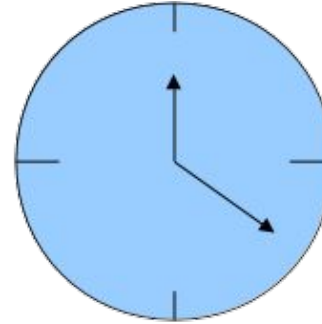
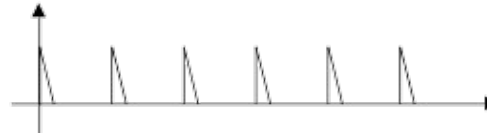
# ¿Cuál es la diferencia entre un contador y un timer?



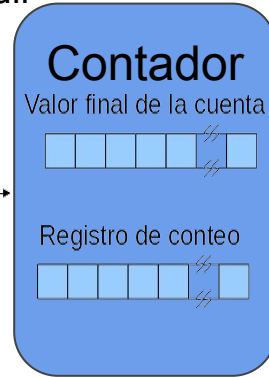
Sensor



Reloj



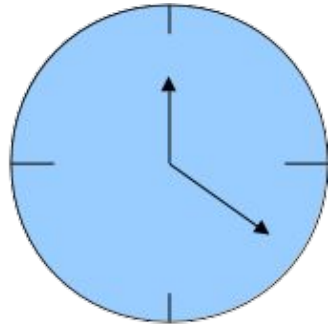
**Temporizador.** Se coloca como entrada eventos periódicos para poder medir tiempo.



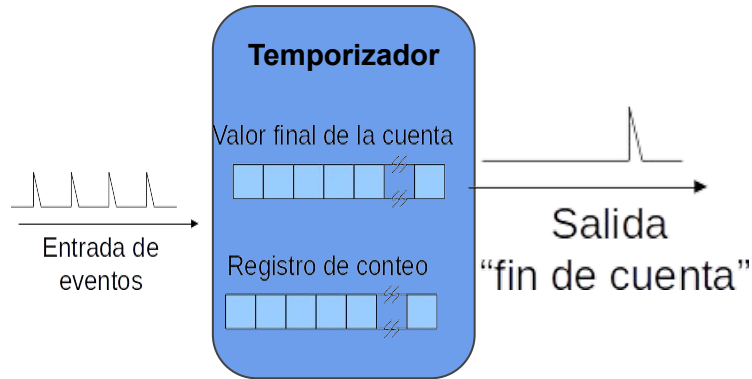
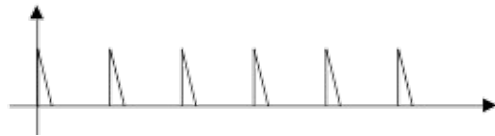
**Un timer es un contador que cuenta eventos PERIÓDICOS  
(Generados típicamente por un oscilador y XTAL)**

# Temporizadores

Nos vamos a centrar en la temporización. Para ello contaremos pulsos que generan con osciladores



Reloj



- Sabemos cada cuanto se produce cada evento de entrada
- Establecemos que cantidad de pulsos queremos contar
- Obtenemos un pulso al final de esa cuenta.

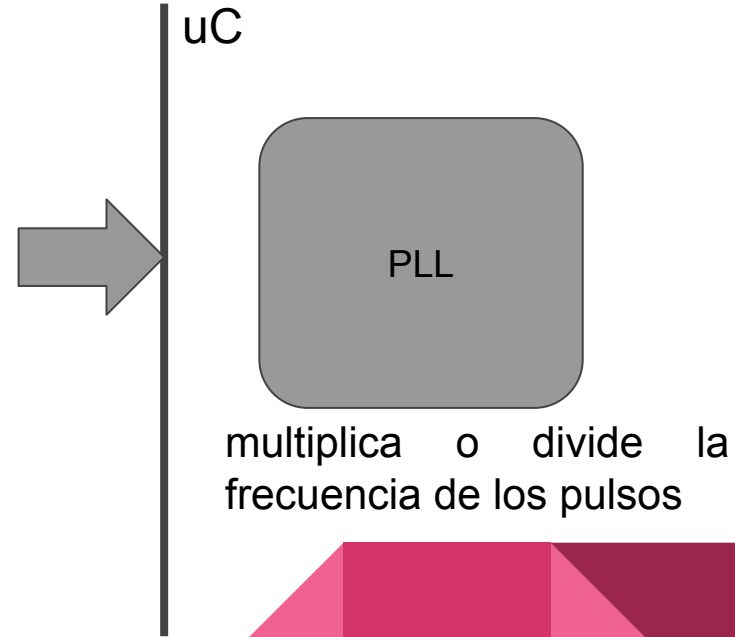
# ¿Qué dispositivos generan pulsos periódicos?



Un Cristal es un circuito que genera una oscilación periódica basada en un material piezoeléctrico (cristal de cuarzo)



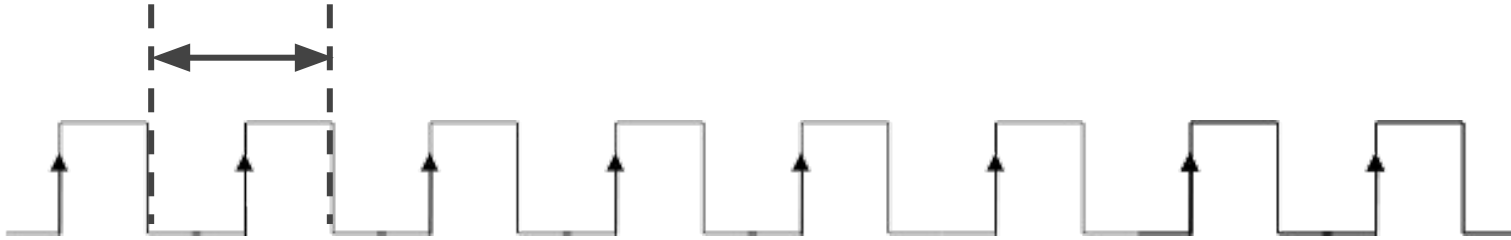
Un oscilador es un circuito que aprovecha las propiedades capacitivas o inductivas de ciertos componentes para generar una oscilación (RC, RLC o RL)



# Frecuencia y Período

El **PERÍODO** de una señal (periódica) es el tiempo que tarda en repetirse a si misma.

$$T = 1\mu\text{seg} = 0,001\text{mseg} = 0.000001\text{seg}$$



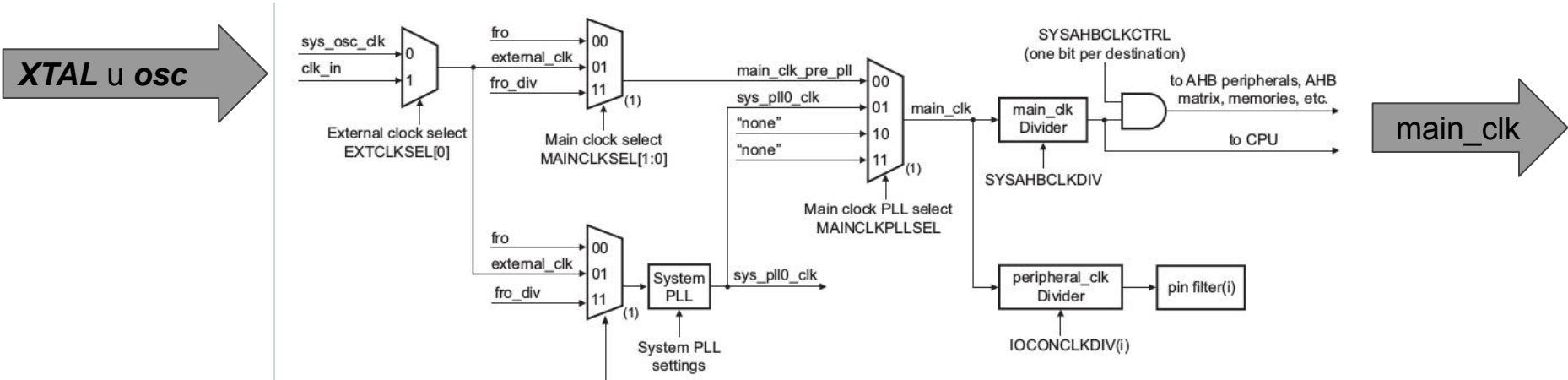
La **FRECUENCIA** de una señal periódica es la inversa del período

$$f = 1 / 1\mu\text{seg} = 1\text{MHz}$$

En general es más cómodo trabajar con frecuencias que con períodos

# Osiladores y PLL (Phase Locked Loop)

En conjunción con los osciladores se suelen utilizar circuitos de PLL. Estos son dispositivos integrados en el microcontrolador que permiten tomar la frecuencia de la señal de entrada y multiplicarla o dividirla para generar una señal de la frecuencia deseada



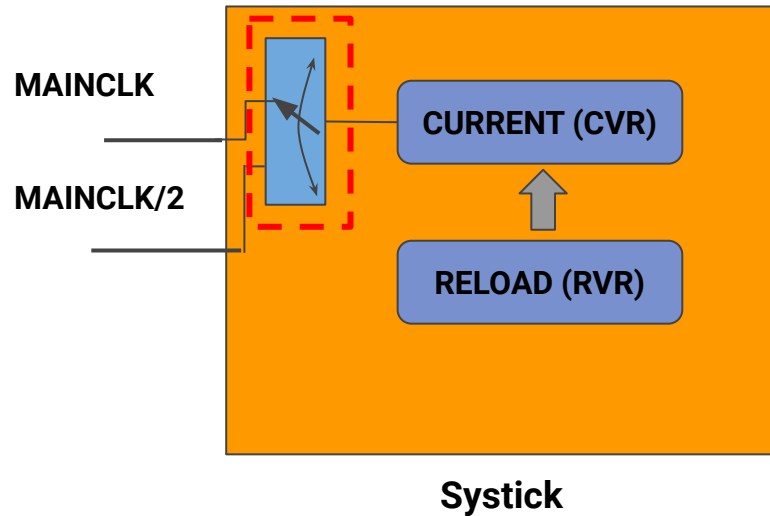
El PLL toma la frecuencia del XTAL (o el oscilador que se utilice) y genera una base de tiempos estable para todos los periféricos. Cada periférico luego puede utilizar esta frecuencia o un divisor de la misma (Según los registros de configuración disponibles para cada periférico)

**En nuestro LPC845 ejecutando la función de la librería InitHw() tendremos un main\_clk de 30Mhz.**

# Diagrama en bloques y registros - SysTick

Selección de entrada, habilitación de interrupciones y comienzo/parada del timer:

## CONTROL (CSR)



- El systick es un dispositivo integrado en el core.
- Está diseñado como un **timer** sencillo, de **24 bits**, que cuenta en forma descendente desde **STRELOAD (SYSTICK->RVR)**.
- No necesita habilitarse su interrupción en el NVIC, sino que se hace directamente en el registro **CONTROL (SYSTICK->CSR)**



# Descripción de los registros - CSR

**Table 435. SysTick Timer Control and status register (SYST\_CSR, 0xE000 E010) bit description**

Bit	Symbol	Description	Reset value
0	ENABLE	System Tick counter enable. When 1, the counter is enabled. When 0, the counter is disabled.	0
1	TICKINT	System Tick interrupt enable. When 1, the System Tick interrupt is enabled. When 0, the System Tick interrupt is disabled. When enabled, the interrupt is generated when the System Tick counter counts down to 0.	0
2	CLKSOURCE	System Tick clock source selection. When 1, the system clock (CPU) clock is selected. When 0, the system clock/2 is selected as the reference clock.	0
15:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
16	COUNTFLAG	Returns 1 if the SysTick timer counted to 0 since the last read of this register.	0
31:17	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

# Descripción de los registros - CVR / RVR

Table 437. System Timer Current value register (SYST\_CVR, 0xE000 E018) bit description

Bit	Symbol	Description	Reset value
23:0	CURRENT	Reading this register returns the current value of the System Tick counter. Writing any value clears the System Tick counter and the COUNTFLAG bit in STCTRL.	0
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 436. System Timer Reload value register (SYST\_RVR, 0xE000 E014) bit description

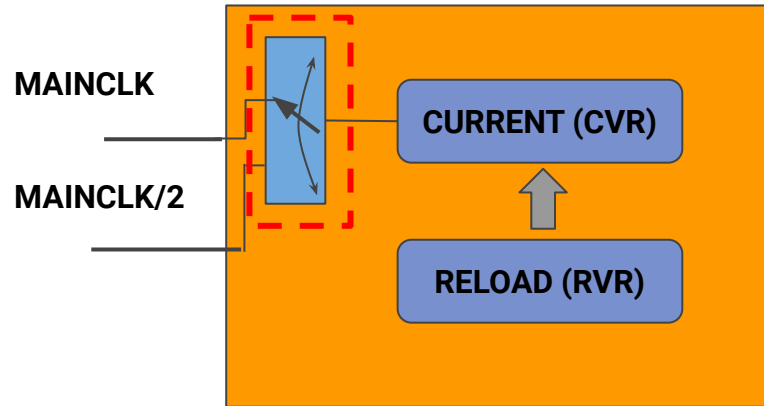
Bit	Symbol	Description	Reset value
23:0	RELOAD	This is the value that is loaded into the System Tick counter when it counts down to 0.	0
31:24	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

El registro **CVR** es el que lleva la cuenta. Un tick después de llegar a 0 (underflow) genera un evento (que puede o no generar una interrupción, y **CVR** se carga nuevamente con el valor de **RVR**

# Configurando el SysTick

Selección de entrada, habilitación de interrupciones y comienzo/parada del timer:

**CSR**



**Systick**

# Para STRELOAD

## 34.4.4.2.1 Calculating the RELOAD value

The RELOAD value can be any value in the range `0x00000001-0x00FFFFFF`. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use:

- To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.
- To deliver a single SysTick interrupt after a delay of N processor clock cycles, use a RELOAD of value N. For example, if a SysTick interrupt is required after 400 clock pulses, set RELOAD to 400.

# Hagamos la inicialización

La función de inicialización del systick que está en la librería se llama:

```
void InicializarSysTick ( void );
```

```
//2.5ms = 400Hz
```

RVR = 1 pulso ----- T = 1/30MHz

RVR = N pulsos ----- T = 1/400Hz -----> N = 30000000/400 = 75000

```
Systick->RVR = 75000;
```


```
Systick->CVR = Systick->RVR;
```

```
Systick->SCR |= 0x07;
```

```
STCTRL |= 0x01
```

```
STCTRL |= 0x01 << 1
```

```
STCTRL |= 0x01 << 2
```



# Recordando las interrupciones...

```
148//*****
149// The vector table.
150// This relies on the linker script to place at correct location in memory.
151//*****
152extern void (* const g_pfnVectors[]) (void);
153__attribute__((section(".isr_vector")))
154void (* const g_pfnVectors[]) (void) = {
155    // Core Level - CM3
156    &vStackTop, // The initial stack pointer
157    ResetISR,    // The reset handler
158    NMI_Handler, // The NMI handler
159    HardFault_Handler, // The hard fault handler
160    MemManage_Handler, // The MPU fault handler
161    BusFault_Handler, // The bus fault handler
162    UsageFault_Handler, // The usage fault handler
163    0, // Reserved
164    0, // Reserved
165    0, // Reserved
166    0, // Reserved
167    SVCall_Handler, // SVCall handler
168    DebugMon_Handler, // Debug monitor handler
169    0, // Reserved
170    PendSV_Handler, // The PendSV handler
171    SysTick_Handler, // The SysTick handler
172    // Chip Level - LPC17
173    WDT_IRQHandler, // 16, 0x40 - WDT
174    TIMER0_IRQHandler, // 17, 0x44 - TIMER0
175    TIMER1_IRQHandler, // 18, 0x48 - TIMER1
176    TIMER2_IRQHandler, // 19, 0x4c - TIMER2
177    TIMER3_IRQHandler, // 20, 0x50 - TIMER3
178    UART0_IRQHandler, // 21, 0x54 - UART0
179    UART1_IRQHandler, // 22, 0x58 - UART1
```

//En un archivo de mi proyecto...

```
void SysTickHandler (void)
{
```

//Este código se ejecutará cada N mseg

...

```
}
```

# Primer ejercicio de timers

Generar una función de inicialización y la ISR del SysTick de manera de poder hacer titilar un led cada 1 segundo, de la siguiente manera:

```
uint8_t estado = 0;

while ( 1 ) {

    if ( segundo ){

        segundo = 0;
        SetLed ( ROJO , estado );
        estado ^=1;

    }

}
```

