Programmable timing functions
Part 1: Timer-generated interrupts

Textbook: Chapter 15, General-Purpose Timers and Timer Interrupts
Chapter 12.4, Cortex SysTick Timer and Interrupts

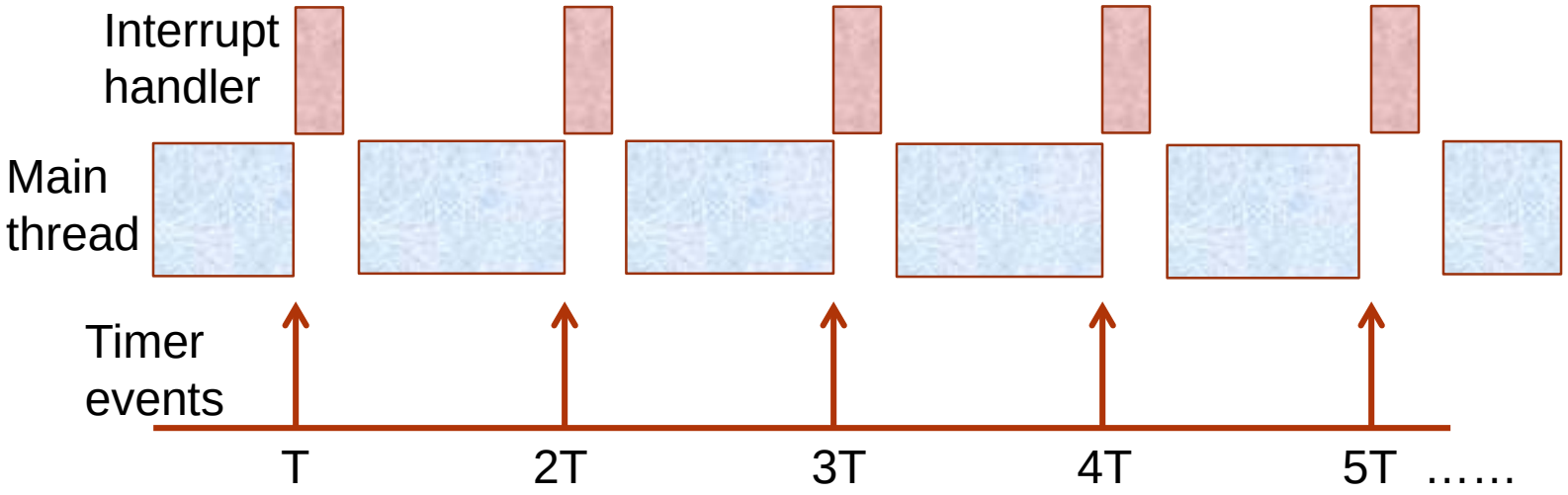
STM32F4xx Technical Reference Manual:
Chapter 17 – Basic timers (TIM6)
Chapter 15 – General-purpose timers (TIM4)
Chapter 10 - Interrupt vectors (for TIM4/TIM6 interrupts)

Timing functions in computer systems

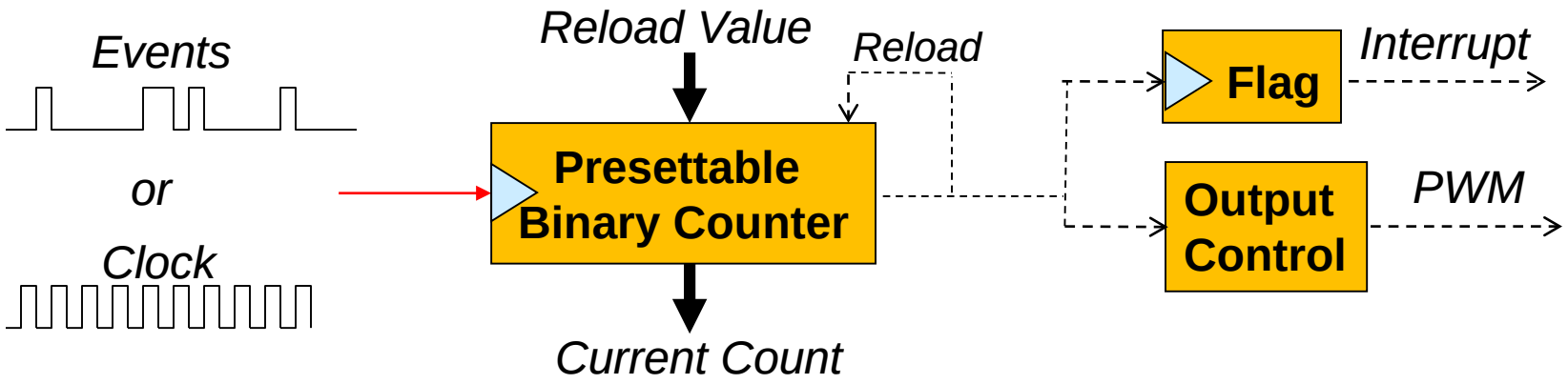
- Periodically interrupt CPU to perform tasks
  - Sample sensor readings (temperature, pressure, etc.)
  - Generate music samples
- Provide accurate time delays
  - Instead of software loops
- Generate pulses or periodic waveforms
  - PWM signal for motor control
  - Strobe pulse for an external device
- Measure duration of an external event
  - Tachometer signal period to measure motor speed

Performing periodic operations

- Certain operations are to be performed every T seconds
  - Timer module interrupts the main thread every T seconds
    - Timer period is usually programmable
  - Interrupt handler performs required operations
    - Operations usually include clearing a flag in the timer



Timer Peripheral Modules



- Based on pre-settable binary counter
  - Count value can be read and written by MCU
  - Count **direction** might be fixed or selectable (up or down)
  - Counter's **clock source** might be fixed or selectable
    - **Counter mode:** count **pulses** which indicate **events** (e.g. odometer pulses)
    - **Timer mode**: periodic clock source, so count value proportional to **elapsed time** (e.g. stopwatch)
  - Counter's **overflow/underflow action** can be configured
    - Set a flag (testable by software)
    - Generate an interrupt (if enabled)
    - Reload counter with a designated value and continue counting
    - Activate/toggle a hardware output signal

STM32F4 Timer Peripherals

- Basic Timer (Simple timer)
  - **TIM6** and TIM7
  - Can be generic counter and internally connected to DAC
  - 16-bit counter
- General Purpose Timer
  - TIM9 to TIM14
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit counter
- General Purpose Timer
  - TIM2, TIM3, **TIM4**, TIM5
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit (TIM3/4) or 32-bit (TIM2/5) counter
- Advanced Control Timer
  - TIM1 and TIM8
  - Input capture, output compare, PWM, one pulse mode
  - 16-bit counter
  - Additional control for driving motor or other devices

Projects will use TIM4, TIM6

- 24 bit system timer(SysTick) – standard in all Cortex-M CPUs

STM32F407 programmable timers

14 timer modules – vary in counter width, max clock, and functionality

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84

Alternate functions for pins PD12-13-14-15

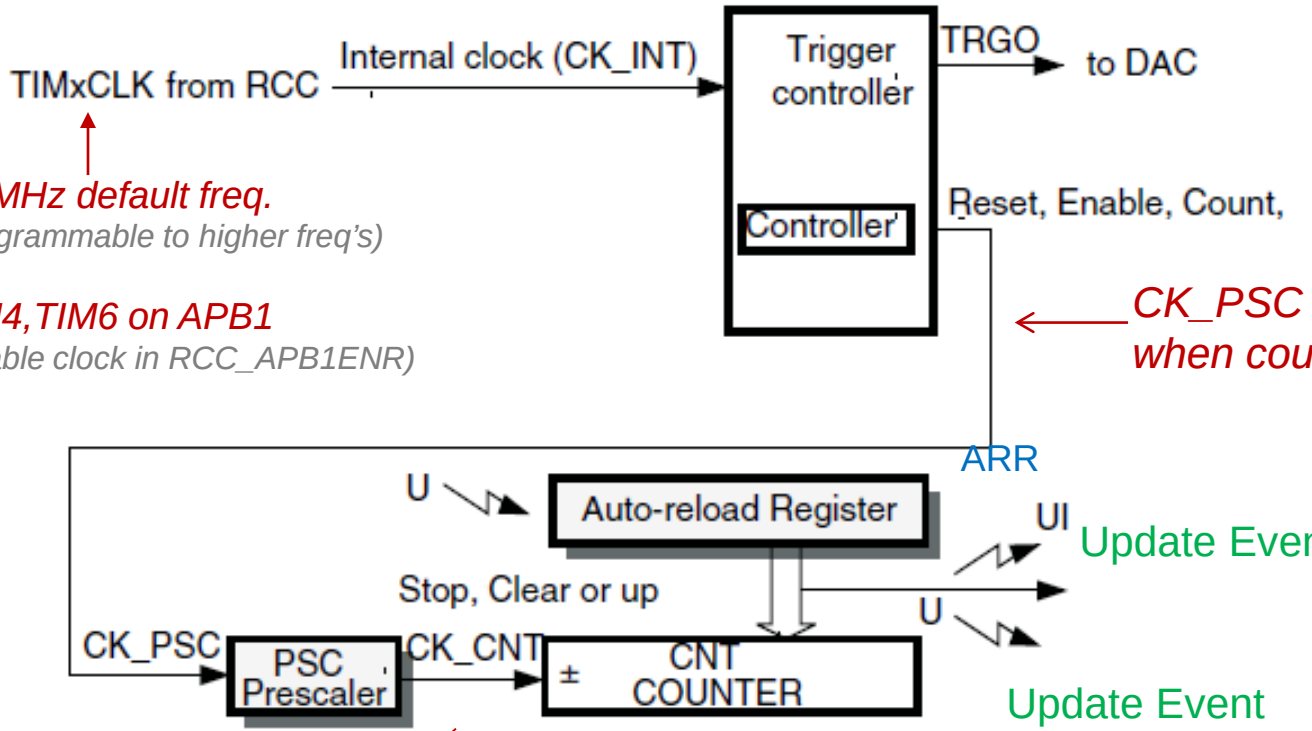
From STM32F407 Data Sheet – Table 6

Pin number						Pin name (function after reset)(1)	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP64	WLCSP90	LQFP100	LQFP144	UFBGA176	LQFP176						
-	-	60	82	M15	101	PD13	I/O	FT		FSMC_A18/TIM4_CH2/ EVENTOUT	
-	-	-	83	-	102	V_{SS}	S				
-	-	-	84	J13	103	V_{DD}	S				
-	F2	61	85	M14	104	PD14	I/O	FT		FSMC_D0/TIM4_CH3/ EVENTOUT/ EVENTOUT	
-	F1	62	86	L14	105	PD15	I/O	FT		FSMC_D1/TIM4_CH4/ EVENTOUT	
-	G2	59	81	N13	100	PD12	I/O	FT		FSMC_A17/TIM4_CH1 / USART3_RTS/ EVENTOUT	

TIM4 can drive LEDs connected to
- PD12 with TIM4_CH1
- PD13 with TIM4_CH2
- PD14 with TIM4_CH3
- PD15 with TIM4_CH4

(So, we will examine TIM4)

Basic timing function



16 MHz default freq.
(programmable to higher freq's)

TIM4,TIM6 on APB1
(enable clock in RCC_APB1ENR)

CK_PSC = CK_INT when count enabled

Update Event Interrupt

Signaled when UIF sets, if enabled (UIE=1)

Update Event

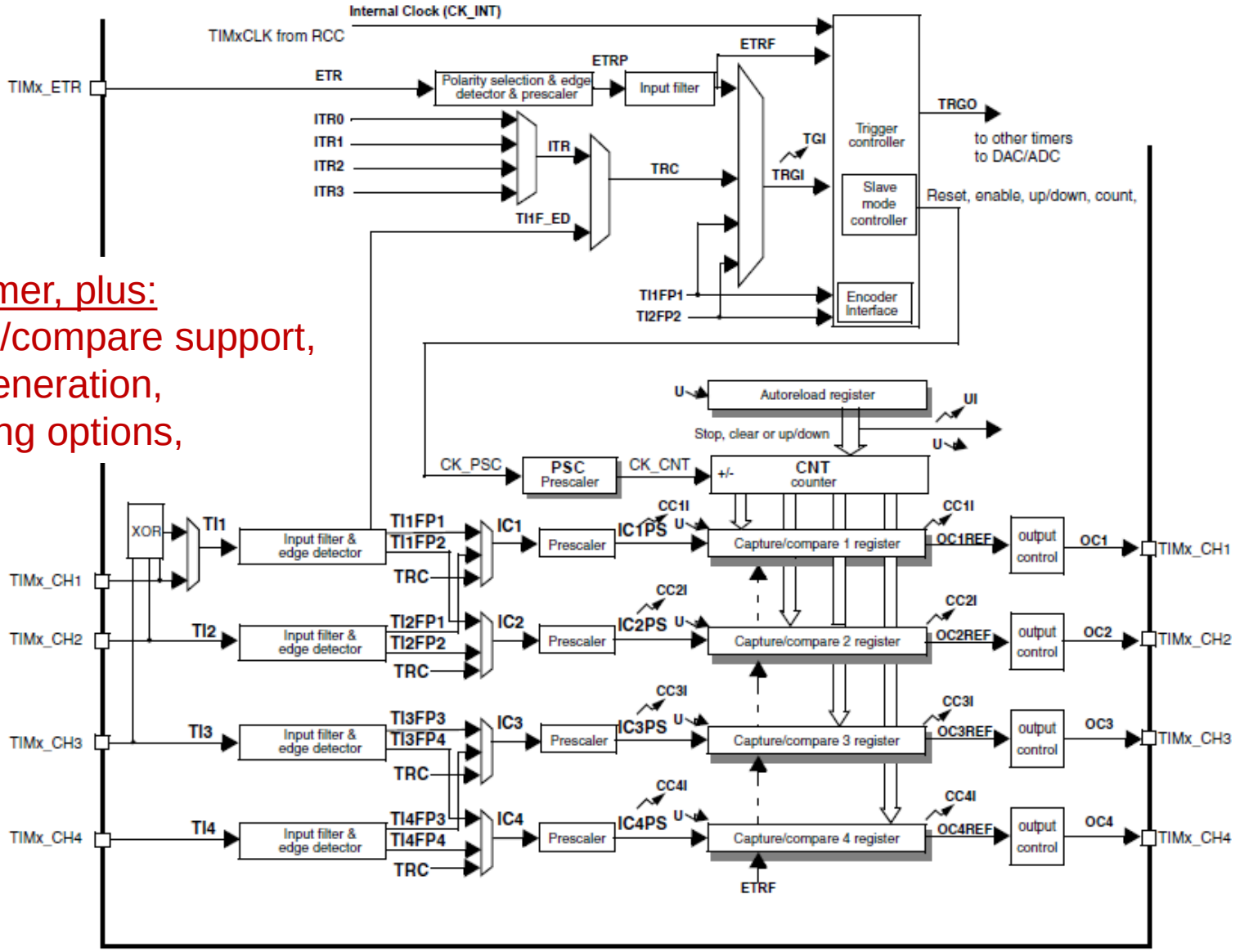
Scaled clock triggers up-counter/down-counter

$F_{CK\_CNT} = F_{CK\_PSC} \div$ Prescale

Event: CNT=ARR (up-count) or CNT=0 (down-count)

- **CNT** resets to 0 (if count up) or reloads ARR (if count down)
- **UIF flag** is set in the status register

General-purpose timers TIM2 – TIM5



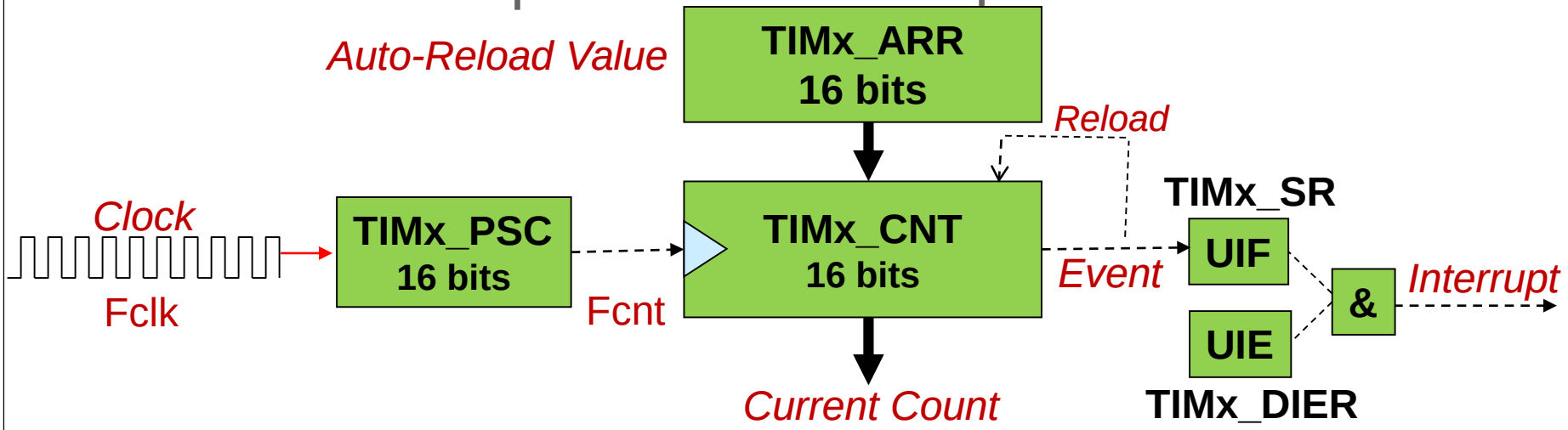
Basic timer, plus:

Capture/compare support,

PWM generation,

Triggering options,

Timer as a periodic interrupt source



- Count-up "overflow event" if **TIMx_CNT** reaches **TIMx_ARR**
  - **UIF** (udate interrupt flag) sets and TIMx_CNT resets to 0.
  - If **UIE** = 1 (update interrupt enabled), interrupt signal sent to NVIC
- **Prescale** value (set by **TIMx_PSC**) multiplies input clock period (1/ Fclk) to produce counter clock period:

  $$Tcnt = 1/Fcnt = (PSC+1)\times(1/Fclk)$$

- Periodic time interval is **TIMx_ARR** (Auto-Reload Register) value times the counter clock period:

  $$Tout = (ARR+1)\times Tcnt = (ARR+1)\times(PSC+1)\times(1/Fclk)$$

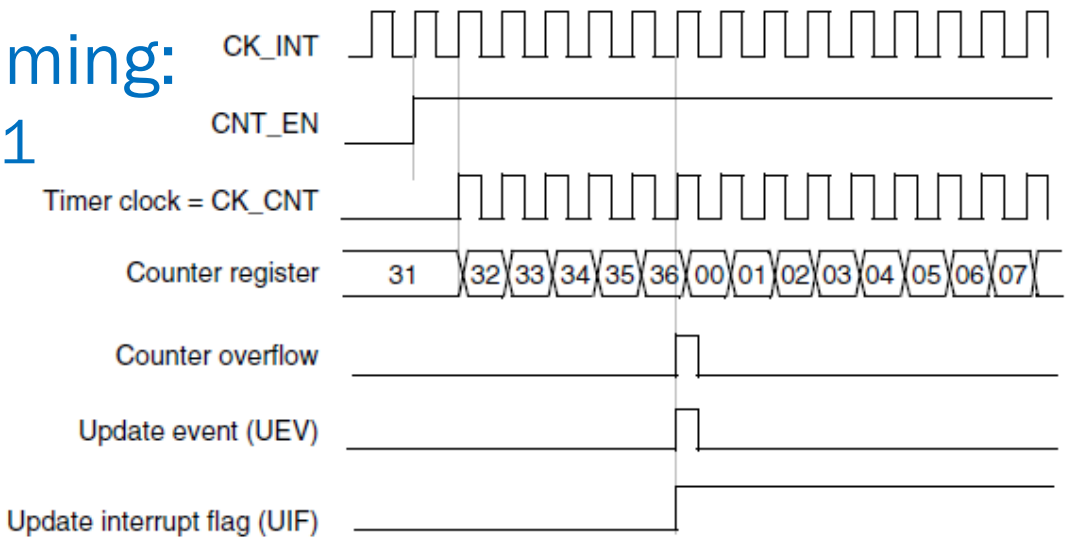
Example: For 1 second time period, given Fclk = 16MHz:

$$Tout = (10000 \times 1600) \div 16000000 = 1 \text{ second}$$

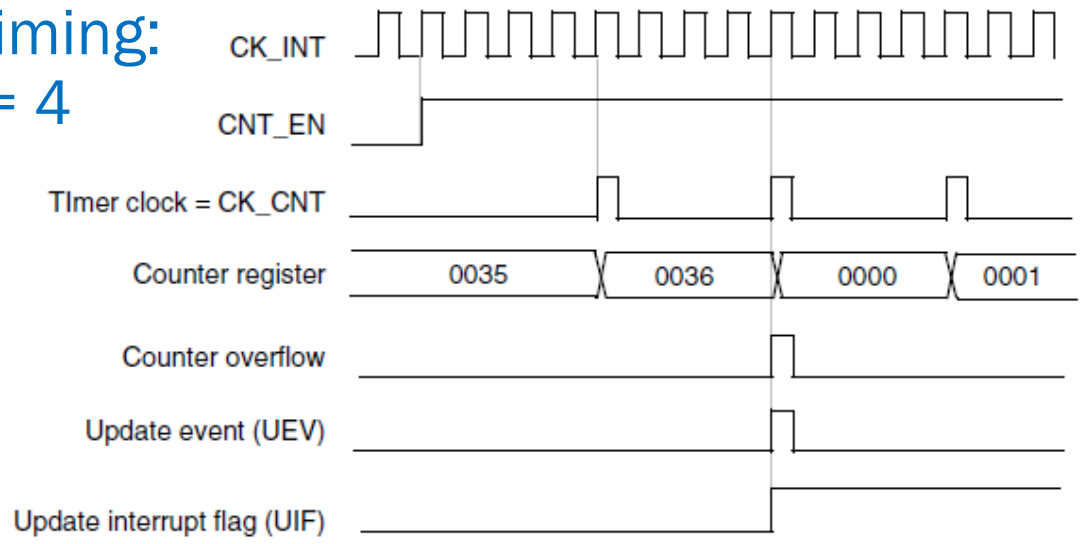
Set ARR = 9999 and PSC = 1599 *(other combinations can also be used)*

T_{EVENT} = Prescale x Count x T_{CK_INT} = (PSC+1) x (ARR+1) x T_{CK_INT}

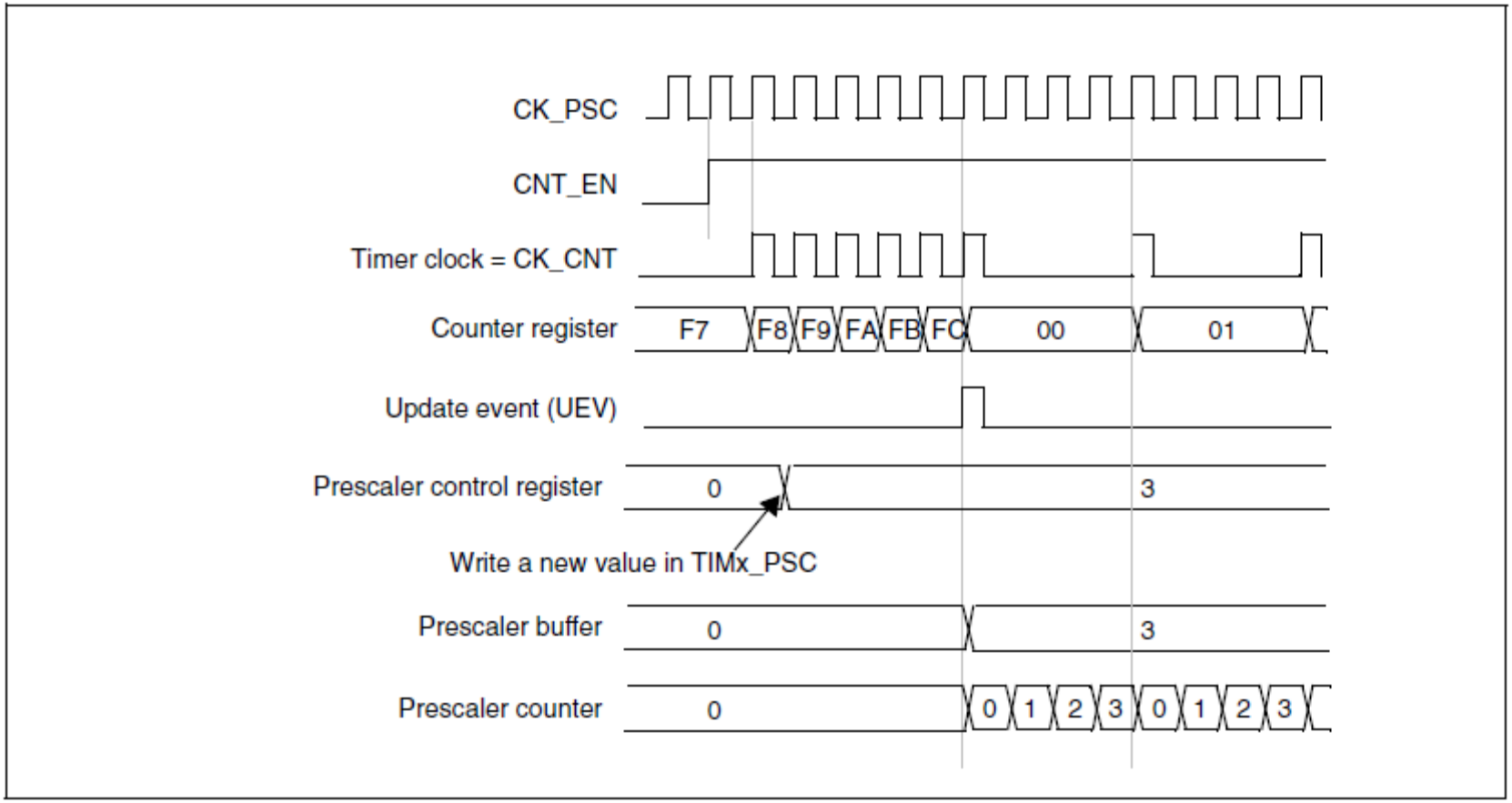
Counter timing:
Prescale = 1
ARR = 36



Counter timing:
Prescale = 4
ARR = 36



Counter timing (prescale changes 1->4)

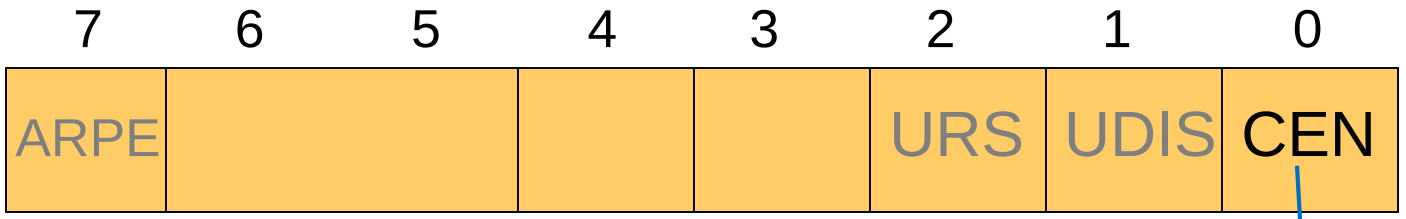


Basic timer function registers
(present in all 14 timers)

- **TIMx Counter** (TIMx_CNT, address offset 0x24)
  - 16-bit binary counter (32 in TIM2, TIM5)
  - Up counter in TIM6-TIM7, TIM9-TIM14
  - Up/down in TIM1-TIM5, TIM8
- **TIMx Prescale Register** (TIMx_PSC, address offset 0x28)
  - Clock prescale value (16 bits)
  - $f_{CK\_CNT} = f_{CK\_INT} \div$ prescale *(assuming CK_INT is clock source)*
- **TIMx Auto-Reload Register** (TIMx_ARR, addr. offset 0x2C)
  - 16-bit register (32 in TIM2, TIM5)
  - End value for up count; initial value for down count
  - New ARR value can be written while the timer is running
    - Takes effect immediately if ARPE=0 in TIMx_CR1
    - Held in buffer until next update event if ARPE=1 in TIMx_CR1

Timer System Control Register 1

TIMx_CR1 (default = all 0's)



Counter Enable

1 = enable, 0 = disable

CEN=1 to begin counting

(apply CK_INT to CK_PSC)

Examples:

TIM4->CR1 |= 0x01; //Enable counting

TIM4->CR1 &= ~0x01; //Disable counting

Other Options:

UDIS = 0 enables update event to be generated (default)

URS = 0 allows different events to generate update interrupt (default)

1 restricts update interrupt to counter overflow/underflow

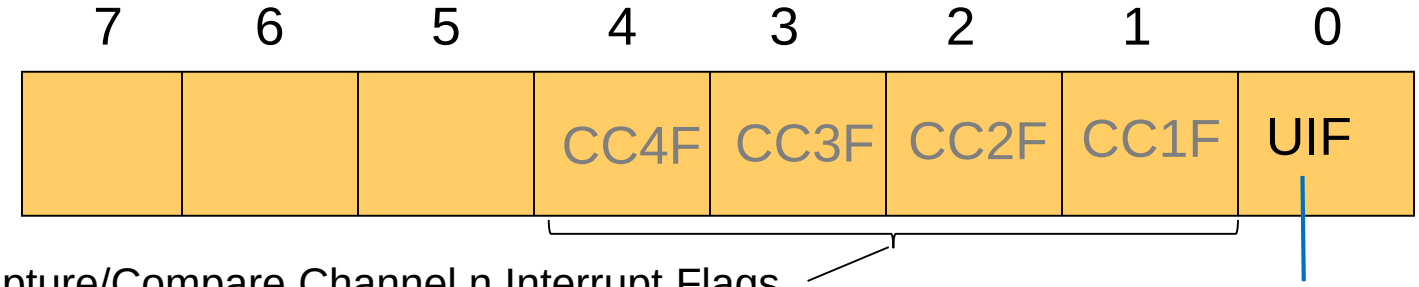
ARPE = 0 allows new ARR value to take effect immediately (default)

1 enables ARR buffer (new value held in buffer until next update event)

TIM2-4 and TIM9 include up/down direction and center-alignment controls

Timer Status Register

TIMx_SR (reset value = all 0's)



Capture/Compare Channel n Interrupt Flags
(to be discussed later)

Update Interrupt Flag

1 = update interrupt pending
0 = no update occurred

Set by hardware on update event
(CNT overflow)

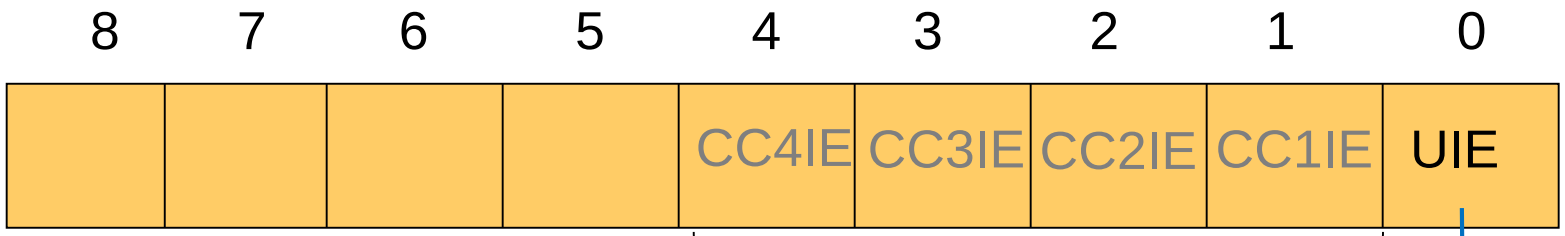
Cleared by software
(write 0 to UIF bit)

Example: do actions if UIF=1

```
if (TIM4->SR & 0x01 == 0x01) {   //test UIF
    .. do some actions
   TIM4->SR &= ~0x01;  //clear UIF
}
```

Timer Interrupt Control Register

TIMx_DIER (default = all 0's)



Capture/Compare n Interrupt Enable
(To be discussed later)

Update interrupt enable
1 = enable, 0 = disable
(interrupt if UIF=1 when UIE=1)

Examples:

```
TIM4->DIER |= 0x01;      //Enable interrupt
TIM4->DIER &= ~0x01;  //Disable interrupt
```

Timer clock source

- Clock TIMx_CLK to each timer module TIMx must be **enabled** in the RCC (reset and clock control) module
  - TIMx_CLK is derived from a peripheral bus clock
    - TIM2-3-4-5-6-7 on APB1 *(peripheral bus 1)*, enabled in RCC->APB1ENR
    - TIM9-10-11 on APB2 *(peripheral bus 2)*, enabled in RCC->APB2ENR
    - Example: enable clocks to TIM2 and TIM9:

```
RCC->APB1ENR |= 0x00000001;  //TIM2EN is bit 0 of APB1ENR
RCC->APB2ENR |= 0x00000004;  //TIM9EN is bit 2 of APB2ENR
```

- *Default STM32F4xx startup code sets all bus/timer clocks to **16MHz** on the Discovery board*

Initialize the TIM4 with CMSIS

- Enable clock to Timer4

```
RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;
```

- Set the auto-reload

```
TIM4->ARR = arr_value;
```

- Set the prescaler

```
TIM4->PSC = psc_value;
```

- Enable the update interrupt

```
TIM4->DIER |= TIM_DIER_UIE;
```

- Enable counting

```
TIM4->CR1 |= TIM_CR1_CEN;
```

Assembly:

```
RCC_TIM4EN  EQU  0x04
arr_value   EQU   4999
psc_value   EQU   9999
DIER_UIE    EQU   1
CR1_CEN     EQU   1

ldr   r0,=RCC
ldr   r1,[r0,#APB1ENR]
orr   r1,#RCC_TIM4EN
str   r1,[r0,#APB1ENR]
ldr   r0,=TIM4
mov r1,#arr_value
str   r1,[r0,#ARR]
mov r1,#psc_value
str   r1,[r0,#PSC]
ldr   r1,[r0,#DIER]
orr    r1,#DIER_UIE
str   r1,[r0,#DIER]
ldr   r1,[r0,#CR1]
orr   r1,#CR1_CEN
str   r1,[r0,#CR1]
```

Timer interrupt vectors

- Each timer has its own interrupt vector in the vector table

  (refer to the startup file and Table 61 in the STM32F4xx Reference Manual)

  - IRQ# determines vector position in the vector table
    - IRQ#: IRQ28 – 29 – 30 – 54 - 55

      Timer#: *TIM2 - 3 - 4 - 6 - 7*
  - Default interrupt handler names* in the startup file:

    TIM4_IRQHandler(); //handler for TIM4 interrupts

    TIM6_DAC_IRQHandler(); //handler for TIM6 interrupts

**Either use this name for your interrupt handler, or modify the startup file to change the default to your own function name.*

Enabling timer interrupts

- Timer interrupts must be enabled in **three places**

1. In the timer: UIE bit (bit 0) in register TIMx_DIER

   TIM4->DIER |= 1; *// UIE is bit 0*

   or: *TIM4->DIER |= TIM_DIER_UIE;* *// symbol from header file*

   - Interrupt triggered if UIF is set while UIE = 1
   - Interrupt handler must reset UIF **(write 0 to it)**

2. In the NVIC – set enable bit in NVIC_ISERx for each IRQn source:

   NVIC_EnableIRQ(TIM9_IRQn); *// enable TIM9 interrupts*

   - NVIC "Pending Flag" should reset automatically when the interrupt handler is entered

3. In the CPU – enable CPU to respond to any configurable interrupt

   __enable_irq();

Interrupt Handler

- Interrupts should be enabled in the CPU

  __enable_irq() ;

  Assembly: CPSIE I

- CMSIS ISR name: *TIM4_IRQHandler*

  NVIC_EnableIRQ(TIM4_IRQn); //n = 30 for TIM4

  Assembly: Enable TIM4_IRQn (bit 30) in NVIC_ISER0

- ISR activities
  - Do the ISR's work
  - Clear pending flag in timer module

    TIM4->SR &= ~TIM_SR_UIF;

    Assembly: write 0 to UIF (bit 0) of TIM4_SR
  - Optional: Clear pending IRQ in NVIC (NVIC does this automatically)

    NVIC_ClearPendingIRQ(TIM4_IRQn);

    Assembly: write 1 to bit 30 of NVIC_ICPR

Example: Stopwatch

- Measure time with 100 us resolution
- Display elapsed time on an LCD, updating screen every 10 ms

  04 : 21 : 48 (Min : Sec : Hundredths)
- Assume timer TIM4
  - Timer interrupt handler increments a counter, every 100 us
  - LCD Update every 10 ms
    - Update LCD every nth periodic interrupt
      - n = 10 ms/100us = 100
    - If LCD update is slow, don't increment in ISR!
    - Instead set flag LCD_Update in ISR, poll it in main loop
    - Usually the ISR is only for updating the timer or for delaying (precise timing!)