ELEC 3040/3050
Lab 6

Time-based interrupts (Digital stopwatch design)

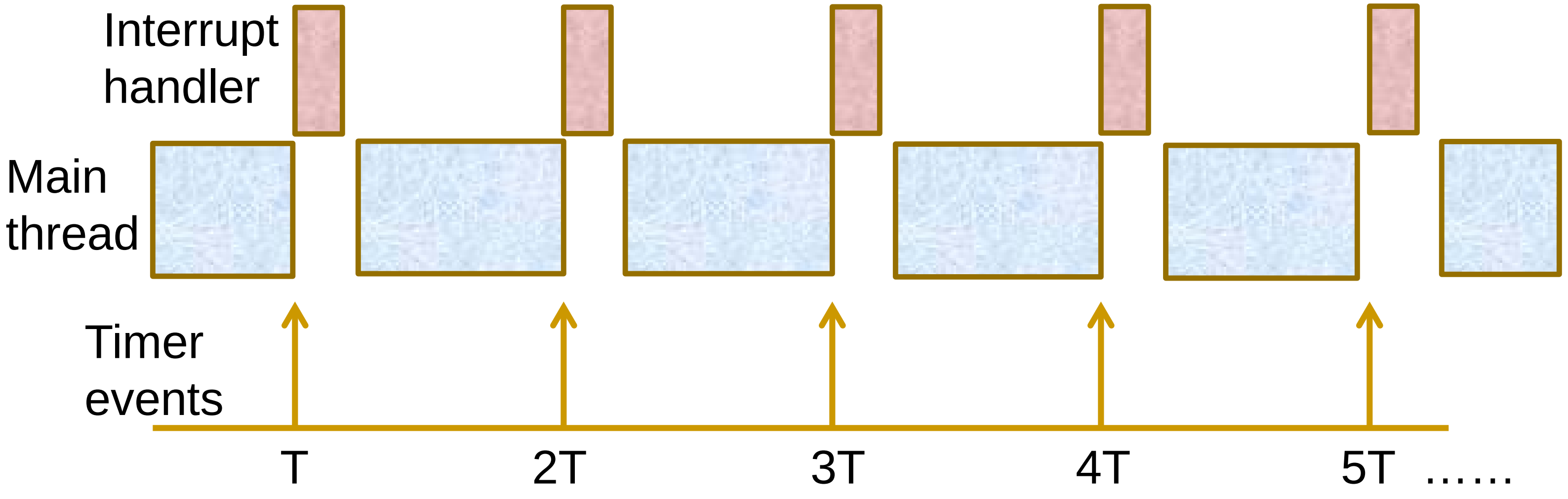
Reference: *STM32L100 Reference Manual, Chap. 18, General-Purpose Timers (TIM9/TIM10/TIM11)*

Timer usage in computer systems

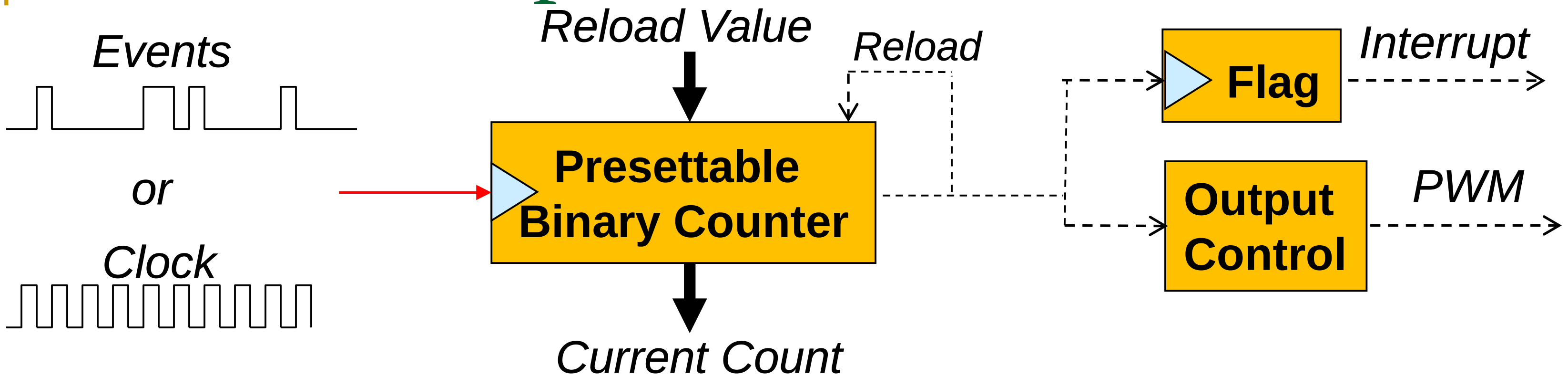
- Periodically interrupt CPU to perform tasks
  - Sample temperature/pressure readings
  - Generate music samples
- Provide accurate time delays
  - Instead of software loops
- Generate pulses or periodic waveforms
  - PWM signal for motor control
  - Strobe pulse for an external device
- Determine time/duration of an external event
  - Measure a tachometer signal from a motor

Performing periodic operations

- Operations to be performed every T seconds
  - Timer module interrupts main thread every T seconds
    - Timer **period** 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 on some uCs)
  - Count **direction** (up/down) may be fixed or selectable
  - Counter's **clock source** may be fixed or selectable
    - **Counter mode:** count **pulses/events** (e.g. odometer pulses)
    - **Timer mode**: periodic clock source;
      count value proportional to **elapsed time** (e.g. stopwatch)
  - Count **overflow/underflow action** configurable
    - Set a **flag** (testable by software)
    - Generate an **interrupt** when flag set (if enabled)
    - **Reload** counter with a designated value and continue counting
    - Activate/toggle a hardware output signal

STM32L1xx programmable timers

- 8 programmable timers (all use 16-bit counters)

Timers	**Count direction**	**Periodic interrupts**	**OC/IC/PWM* channels**
TIM2-3-4	up/down	yes	4
TIM9	up/down	yes	2
TIM10-11	up	yes	1
TIM6-7	up	yes	0

***OC** = Output Compare
IC = Input Capture
PWM = Pulse Width Modulated Waveforms

- Other available timers:
  - Watchdog (WWDG): 7-bit down-counter
    - Interrupt CPU if count reaches 0
    - Correctly-operating S/W periodically resets count to prevent this
  - Real-Time Clock (RTC)
    - Maintains time of day and calendar (with battery backup)
  - SysTick timer
    - 24-bit down-counter in all Cortex-M processors
    - Triggers periodic interrupts (“clock ticks”)

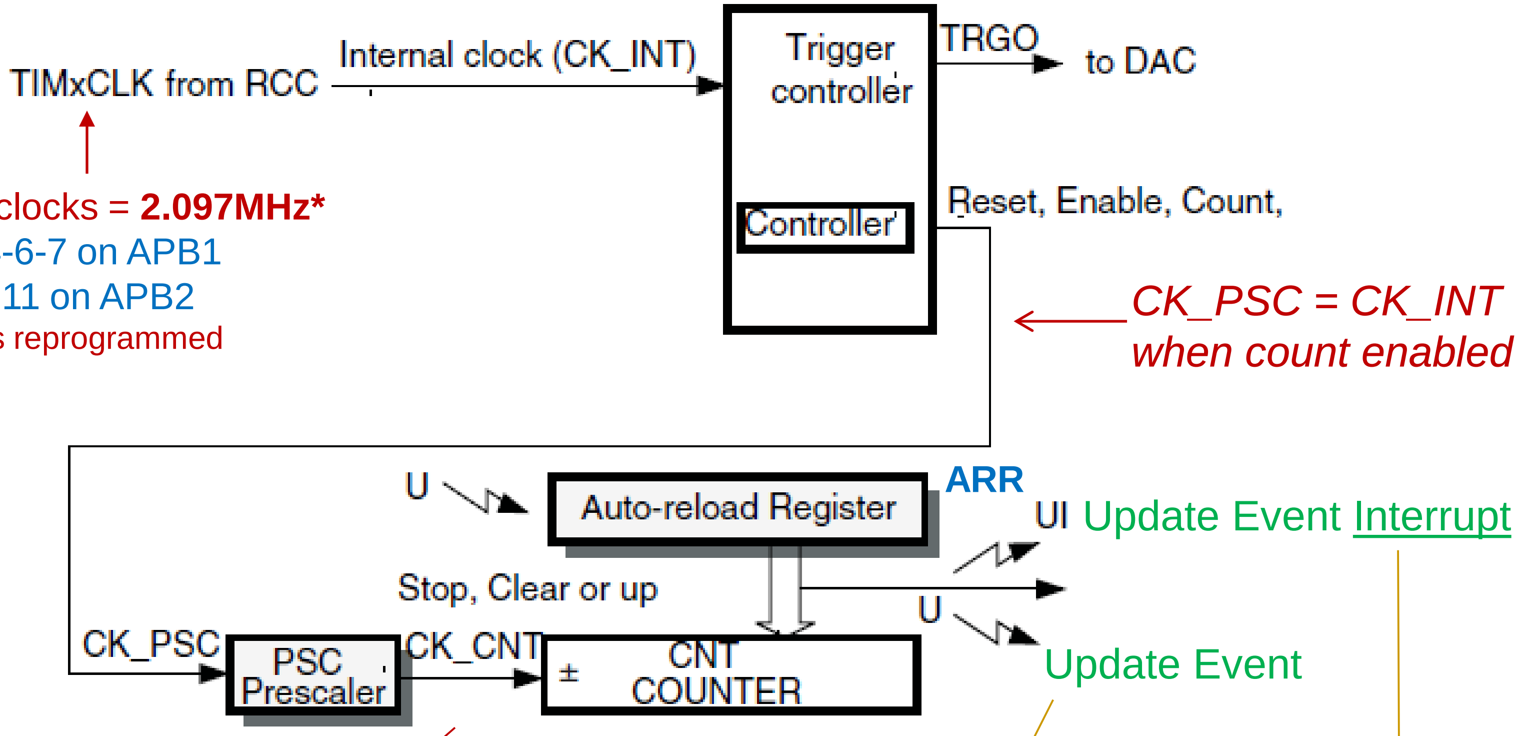
STM32L1xx Clock Options

- **MSI** (Multi-Speed Internal) DEFAULT AT POWER ON
  - Internal oscillator: 2.097 MHz (0x20_0000 Hz)
  - Default clock selected by startup code
- **HSI** (High-Speed Internal)
  - Internal oscillator: 16 MHz
  - Select at startup by executing:

```
RCC->CR |= RCC_CR_HSION;                    // Turn on 16MHz HSI oscillator
while ((RCC->CR & RCC_CR_HSIRDY) == 0);   // Wait until HSI ready
RCC->CFGR |= RCC_CFGR_SW_HSI;           // Select HSI as system clock
```

- **HSE** (High-Speed External)
  - Supplied via pins PH0/PH1: 8 MHz on STM32L100-Discovery
- **PLL** (Phase-Lock Loop)
  - Generate up to 32MHz clock from HSI or HSE source
  - More precise than internal oscillators

Basic timing function



All TIMx clocks = **2.097MHz***
TIM2-3-4-6-7 on APB1
TIM9-10-11 on APB2
* Unless reprogrammed

CK_PSC = CK_INT
when count enabled

UI Update Event Interrupt

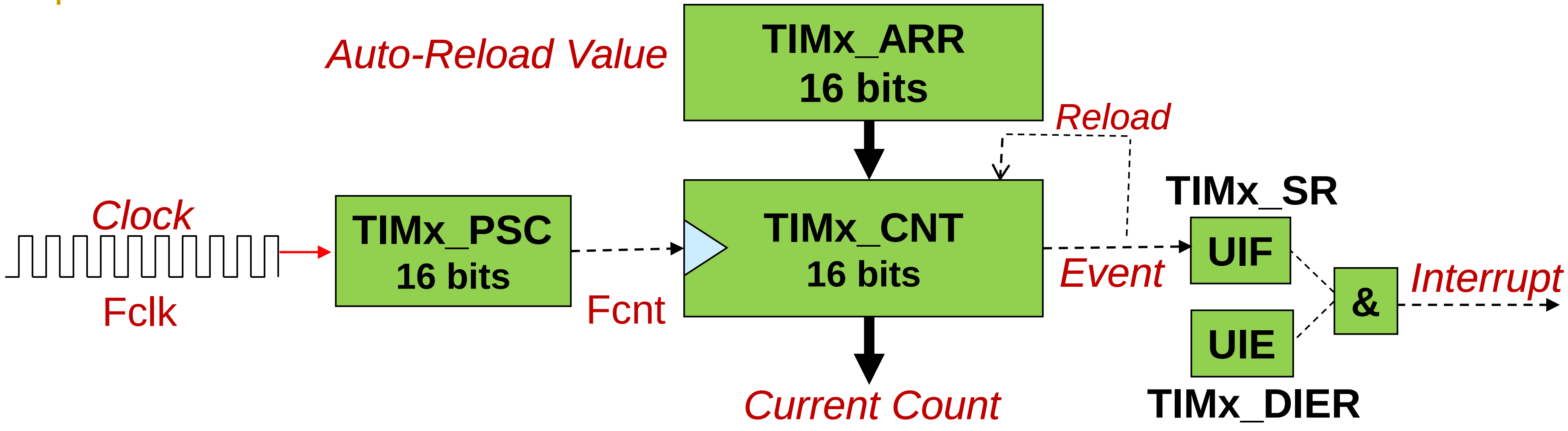
Update Event

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

Scaled clock triggers
up-counter/down-counter
$\mathbf{F}_{\mathbf{CK\_CNT}} = \mathbf{F}_{\mathbf{CK\_PSC}} \div$ 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

Timer as a periodic interrupt source



- Count-up "overflow event" if **TIMx_CNT** reaches **TIMx_ARR**
  - **1→UIF** (udate interrupt flag) 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:

$$T_{cnt} = 1/F_{cnt} = (\mathbf{PSC}+1)\times(1/F_{clk})$$

- Time interval = **TIMx_ARR** (Auto-Reload Register) value × counter clock period:

$$T_{out} = (\mathbf{ARR}+1)\times T_{cnt} = (\mathbf{ARR}+1)\times(\mathbf{PSC}+1)\times(1/F_{clk})$$

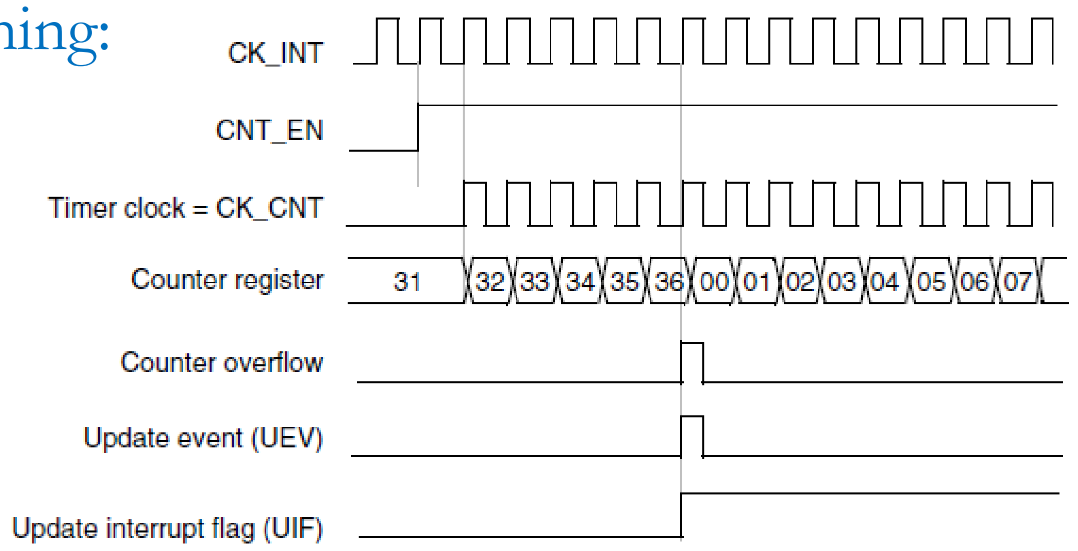
Example: For 1 second time period, given Fclk = 2.097MHz:

$$T_{out} = (0x2000 \times 0x100) \div 0x200000 = 1 \text{ second}$$

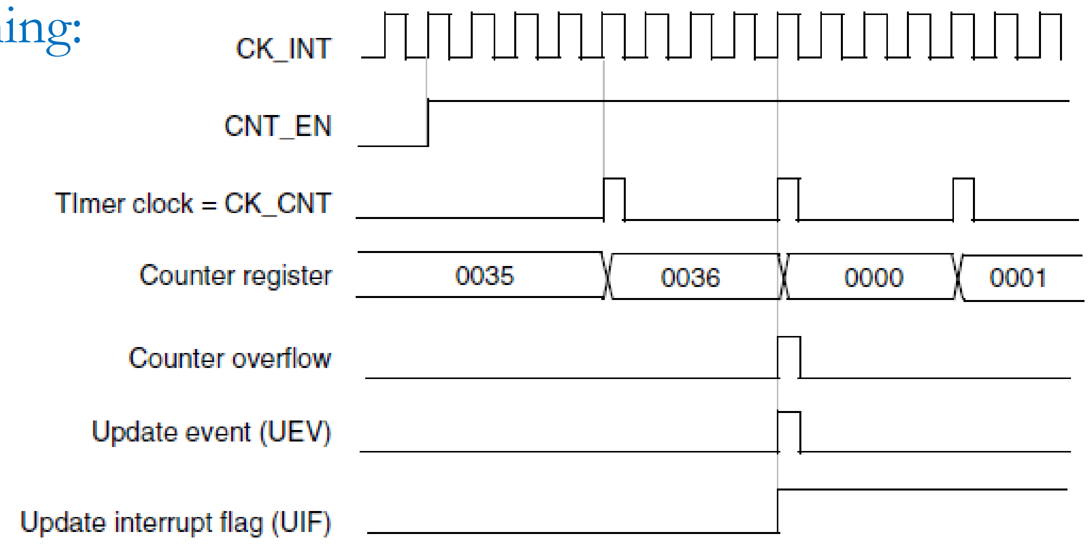
ARR = 0x1FFF & PSC = 0xFF *(or equivalent combination)*

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

Counter timing:
PSC = 0
(prescale x1)
ARR = 36



Counter timing:
PSC= 3
(prescale x4)
ARR = 36

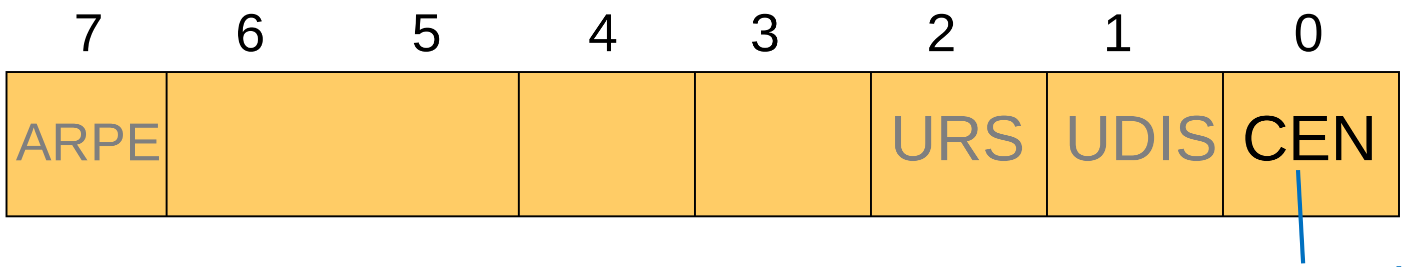


Timer registers

- **TIMx Counter** (TIMx->CNT) *(x = timer #)*
  - 16-bit binary counter, operating at f_{CK_CNT}
  - Up counter in TIM6, TIM7, TIM10, TIM11
  - Up/down counter in TIM2, TIM3, TIM4, TIM9
- **TIMx Prescale Register** (TIMx->PSC)
  - 16-bit clock prescale value
  - $f_{CK\_CNT} = f_{CK\_INT} \div \text{prescale}$ *(CK_INT is the clock source)*
- **TIMx Auto-Reload Register** (TIMx->ARR)
  - 16-bit auto-reload value
  - 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

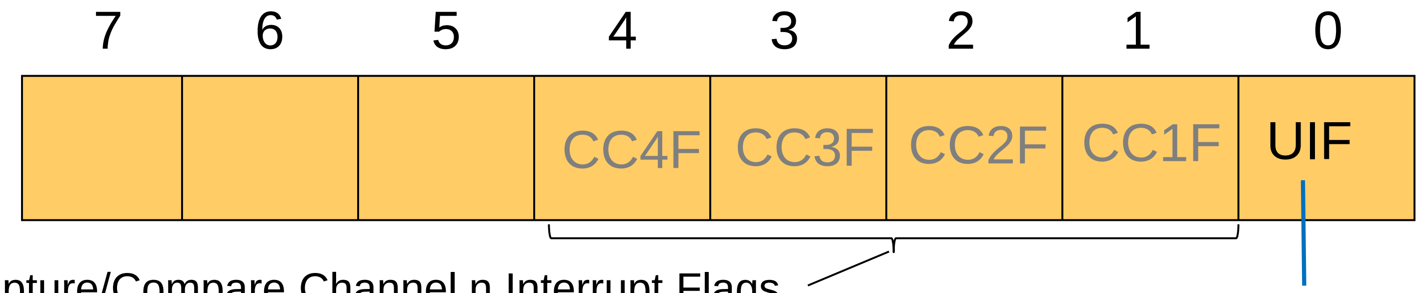
ARPE = 0 allows new ARR value to take effect immediately (default)

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)

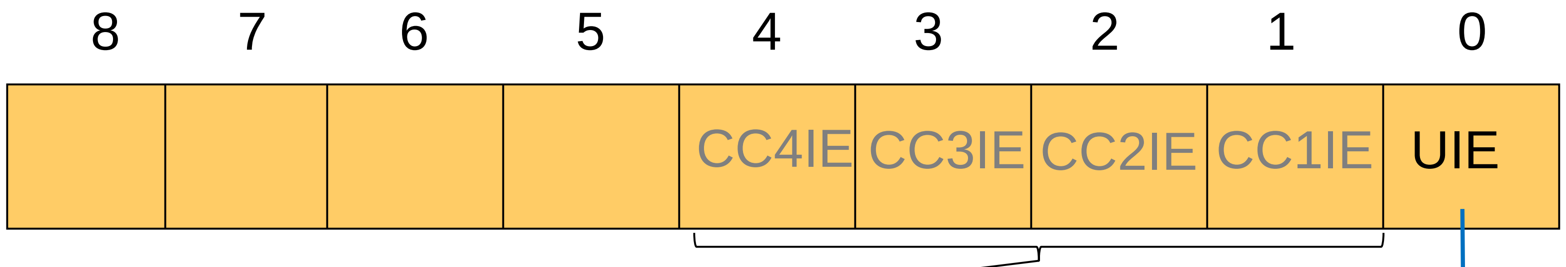
Cleared by software
(write 0 to UIF bit)

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
    - TIM2-3-4-6-7 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
  - `STM32L1xx.h` defines symbols for bit patterns

      RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //same as 0x00000001
      RCC->APB2ENR |= RCC_APB2ENR_TIM9EN; //same as 0x00000004

- *Default STM32L1xx startup code sets all bus/timer clocks to **2.097MHz** (0x200000 Hz) on the Discovery board*
  - *Reprogram if higher frequency desired*

Timer interrupt vectors

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

  (refer to the startup file and Table 48 of STM32L100xx Reference Manual)
  - IRQ# determines vector position in the vector table
    - IRQ#: IRQ25 – 26 – 27 – 28 – 29 – 30 – 43 - 44

      Timer#: *TIM9 - 10 - 11 - 2 - 3 - 4 - 6 - 7*
  - Symbols for IRQ#'s for NVIC functions defined in *stm32l1xx.h*

    *NVIC_EnableIRQ(**TIM9_IRQn**);* *// TIM9 = IRQ25*

    *NVIC_ClearPendingIRQ(**TIM7_IRQn**);* *// TIM7 = IRQ44*
  - Default interrupt handler names* in the startup file:

    TIM9_IRQHandler(); //handler for TIM9 interrupts

    TIM7_IRQHandler(); //handler for TIM7 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 in register TIMx DIER

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

     or:
     ```
     TIM4->DIER |= TIM_DIER_UIE;
     ```
     - Interrupt triggered if UIF is set while UIE = 1
     - Interrupt handler must reset UIF **(write 0 to it)**

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

     ```
     NVIC_EnableIRQ(TIM9_IRQn);   // enable TIM9 interrupts
     ```
     - NVIC "Pending Flag" resets automatically when interrupt handler entered

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

     ```
     __enable_irq();
     ```

Example: Initialize TIM4 for periodic interrupts

- Enable the clock to timer TIM4

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

- Calculate prescale and auto-reload values for desired period and write to PSC and ARR registers:

// $T_{INT} = T_{CLK} * (PSC + 1) * (ARR + 1)$

```
TIM4->PSC = psc_value;   //prescaler value
TIM4->ARR = arr_value;   //auto-reload value
```

- Enable interrupts on timer update events

```
TIM4->DIER |= TIM_DIER_UIE;   //Enable TIM4 to signal an interrupt
NVIC_EnableIRQ(TIM4_IRQn);    //Enable TIM4 interrupt in NVIC
__enable_irq();               //Enable interrupts in CPU
```

- Enable the timer to start counting

```
TIM4->CR1 |= TIM_CR1_CEN;     //Bit CEN=1 to enable counting
                              //Bit CEN=0 to stop counting
```

Interrupt handler format

void TIM4_IRQHandler () { //handler for TIM4

- Perform the desired actions
  - Update a time/display
  - sample external sensor data
  - perform a control action
  - etc.

- Clear timer's update interrupt flag (cancel interrupt request)

  TIM4->SR &= ~0x01; //UIF is bit 0 of SR

  or *TIM4->SR &= ~TIM_SR_UIF;*

- (Should be unnecessary): clear pending flag in NVIC

  NVIC_ClearPendingIRQ(TIM4_IRQn);

}

Stopwatch design

- Control watch with two keypad buttons
  - Button 0: start and stop the timer
    - Freeze the displayed time when stopped
    - Continue from displayed value if restarted
  - Button 1: clear time to 0.0, but only if watch is stopped

- While watch is running, display elapsed time from 0.0 to 9.9 seconds, and repeat until stopped
  - Display two BCD digits on two sets of four LEDs.
    - Use previous port configuration (PC7-PC0) to output two 4-bit values
    - Use previous Static IO window setup to display the values on LEDs
    - Verify count sequence with the logic analyzer
  - Timing must be precise, requiring timer interrupts.
    - Verify timing precision with the oscilloscope