Le Prime Periferiche:
General-Purpose I/O e Timer Hardware

Luigi Coppolino, Giovanni Mazzeo

Outline

- I General-Purpose I/O (GPIO):
  - Cosa sono e a cosa servono i GPIO
  - I GPIO sulla STM32F3-Discovery
  - Come funzionano ed esempi di Registri GPIO
- Anatomia di un progetto
- I General-Purpose Timer (GPT)

i GPIO

- I General-Purpose I/O (GPIO) sono dei pin generici configurabili dall'utente a *run-time*
- Permettono la comunicazione del microcontrollore con il mondo esterno sia in uscita che in ingresso
- Grazie ai GPIO è possibile avere un numero elevato di unità/periferiche in un microcontrollore mantenendo ridotto il numero di pin
- Possono funzionare a 4 differenti velocità di I/O: 5, 25, 50, 100 MHz

I GPIO sulla STM32F3 - 1/2

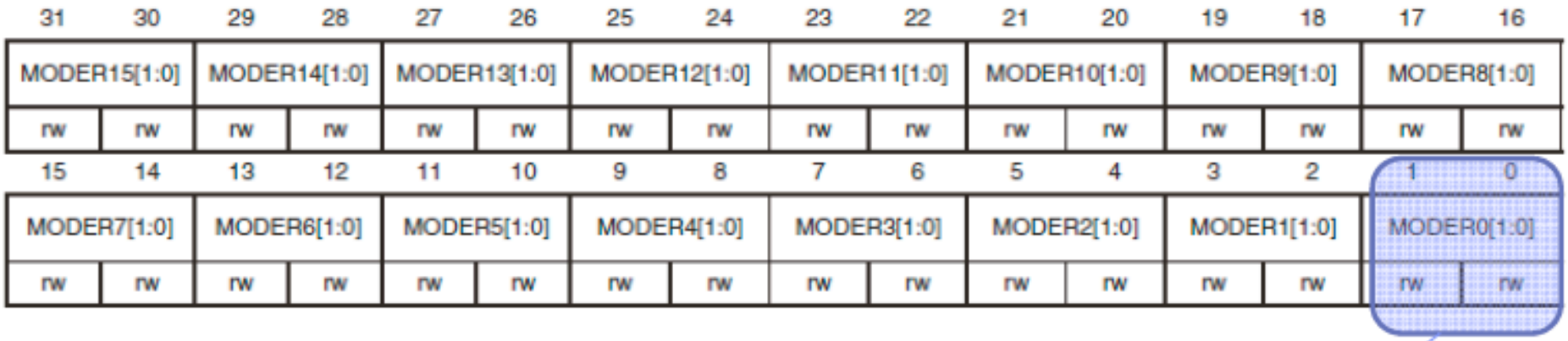
- Il microcontrollore sulla scheda STM32F3-Discovery è provvisto di 8 porti GPIO (definiti dalla A alla H).
- Ogni porto può gestire sino a 16 pins => 16 x 8 = 128 pins totali
- Ogni porto GPIO, inoltre, ha associato un insieme di registri (x = {A…H}) che fanno parte del suo modello di programmazione:
- 4 Registri di Configurazione da 32 bit
  - GPIO port mode register (GPIOx_MODER)
  - GPIO port output type register (GPIOx_OTYPER)
  - GPIO port output speed register (GPIOx_OSPEEDR)
  - GPIO port pull-up/pull-down register (GPIOx_PUPDR)

I GPIO sulla STM32F3 - 2/2

- 2 Registri di Dato da 32 bit
  - GPIO port input data register (GPIOx _IDR)
  - GPIO port output data register (GPIOx_ODR)

- 4 Registri di Controllo da 32 bit
  - GPIO port bit set/reset register (GPIOx _BSRR)
  - GPIO port configuration lock register (GPIOx_LCKR)
  - GPIO alternate function low register (GPI Ox_AFRL)
  - GPIO alternate function high register (GPIOx_AFRH)

GPIO port Mode Register



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

2 bits in the register define the properties of 1 Port pin

GPIO port Mode Register

- Il registro GPIOx_MODER è un registro di 32 bits dove ogni insieme di 2 bit consecutivi rappresenta il modo di un singolo pin
- Ad esempio i bits con posizione 0 e 1 del GPIOD_MODER rappresentano il modo del pin GPIO PC0, I bits di posizione 26 e 27 dello stesso registro rappresentano il modo del pin GPIO PC13
- I due bit possono essere definiti come segue:
  - "00" – Input Mode : definisce l'utilizzo del pin in modalità di input
  - "01" – Output Mode: definisce l'utilizzo del pin in modalità di output
  - "10" – Analog Mode: definisce l'utilizzo del pin in modalità analogica
  - "11" – Alternate Functions

Bit Set/Reset Register (BSRR)

- E‘ un registro a 32bit utile al set o reset di specifici pin

- I 16 bit superiori sono mappati su ciascun pin del porto GPIO. Scrivere un 1 in queste locazioni porterà a 0 (o farà il “reset“) del pin associato.

- Allo stesso modo scrivere un 1 nei 16bit inferiori servirà a portare a 1 (o fare il “set“) il pin associato

Gli altri Registri

- Altri esempi di registri sono:
  - GPIO port output speed register (GPIOx_OSPEEDR) – Registro di configurazione della velocità di un singolo pin di output (2MHz, 10MHz, 50MHz)
  - GPIO port input data register (GPIOx _IDR) – Registro utile alla trasmissione di dati in input
  - GPIO port output data register (GPIOx_ODR) – Registro utile alla trasmissione di dati in output

Modo di Funzionamento

- Come già detto il *mode register* serve al programmatore per configurare il modo di funzionamento del GPIO
- In pratica, il programmatore può decidere se:
  - Abilitare la comunicazione (input/output) del microcontrollore con una specifica unità presente sulla scheda di sviluppo STM32F3-Discovery.

N.B. STM32F3 != STM32F3-Discovery. STM32F3 è il microcontrollore, STM32F3-Discovery è la scheda che comprende poi il microcontrollore

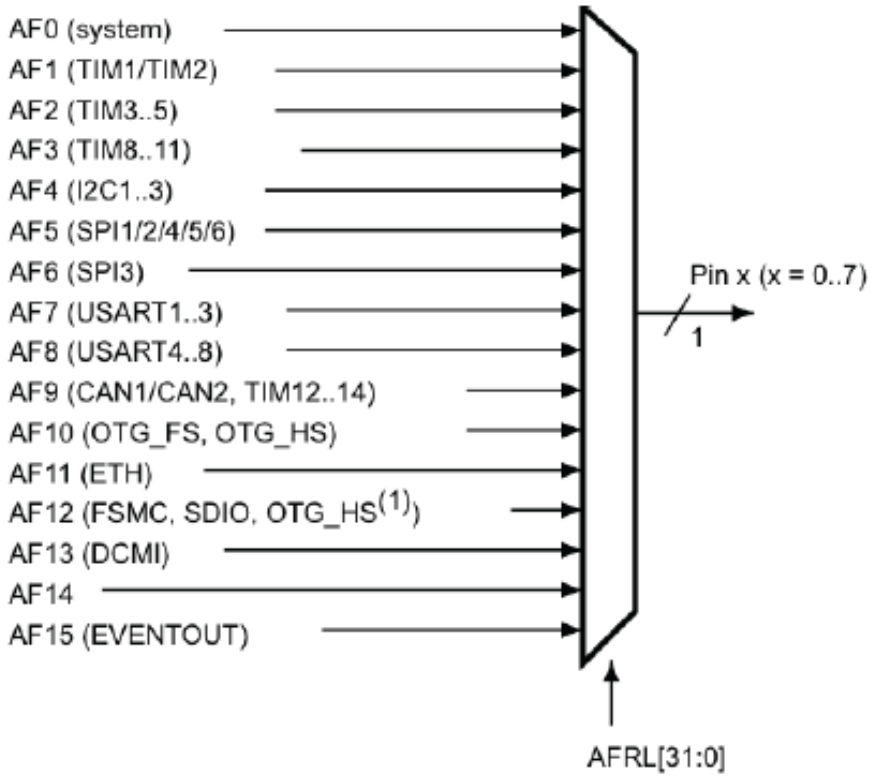
  - Abilitare la comunicazione con l'esterno di una periferica del microcontrollore (e.g. timer, SPI, I2C) (alternate function)
  - Abilitare la comunicazione con l'esterno di una periferica che trasmette segnali analogici (e.g. ADC/DAC, PWM)

Alternate Functions – 1/3

- Permetto di stabilire la comunicazione del mondo esterno con i dispositivi interni al SoC.
- Consentono di ridurre il numero di pin richiesti dal package
- Le Alternate functions permettono l'utilizzo del GPIO per periferiche come UART, SPI ed altri.
- Si noti che se la modalità è impostata su AF le configurazioni per quel pin degli altri registri saranno sovrascritte con le impostazioni del tipo di periferica adottata.

Alternate Functions – 2/3

- I pin di I/O sono connessi alle periferiche del SoC attraverso un MUX
- Ogni pin di I/O ha un MUX con 16 Alternate Functions



Alternate Functions – 3/3

MCU pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PA4	SPI1_NSS/ SPI3_NSS/ USART2_CK/ DCMI_HSYNC/ OTG_HS_SOF/ I2S3_WS/ ADC12_IN4/ DAC1_OUT	29	LRCK/AIN1x												16	

Gestione GPIO in ChibiOS

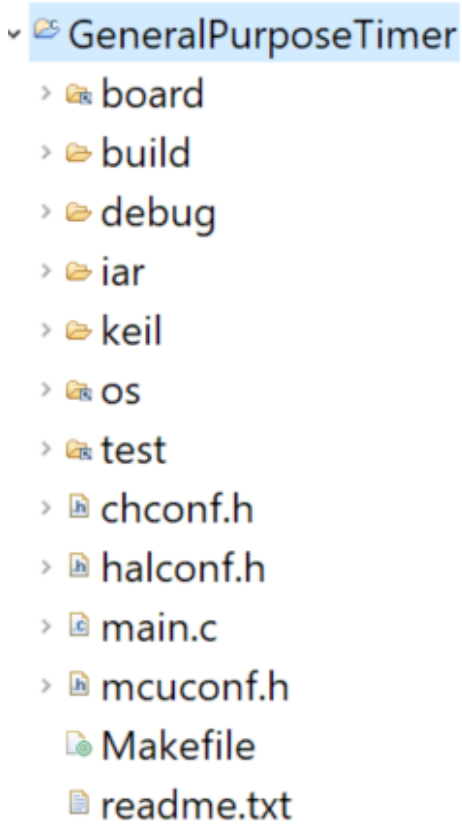
- ChibiOS offre delle librerie dette:
  - **I/O Ports Abstraction Layer (PAL)**
- Tutte le funzioni di questo tipo iniziano con la dicitura *pa*l
- L'header file contenente tutte le funzioni del PAL con la loro descrizione è posizionato in:
  - *os/hal/include/hal_pal.h*

Librerie a Supporto del Programmatore

- I produttori forniscono
  - Header files e Librerie per utilizzare il dispositivo
    - L'header file definisce tutti i tipi e strutture dati necessarie alla programmazione
    - Definisce costanti simboliche per accedere ai registri di controllo delle periferiche
    - Definisce costanti simboliche per i valori di configurazione dei registri di controllo
  - Source files con le effettive implementazioni delle funzioni di libreria

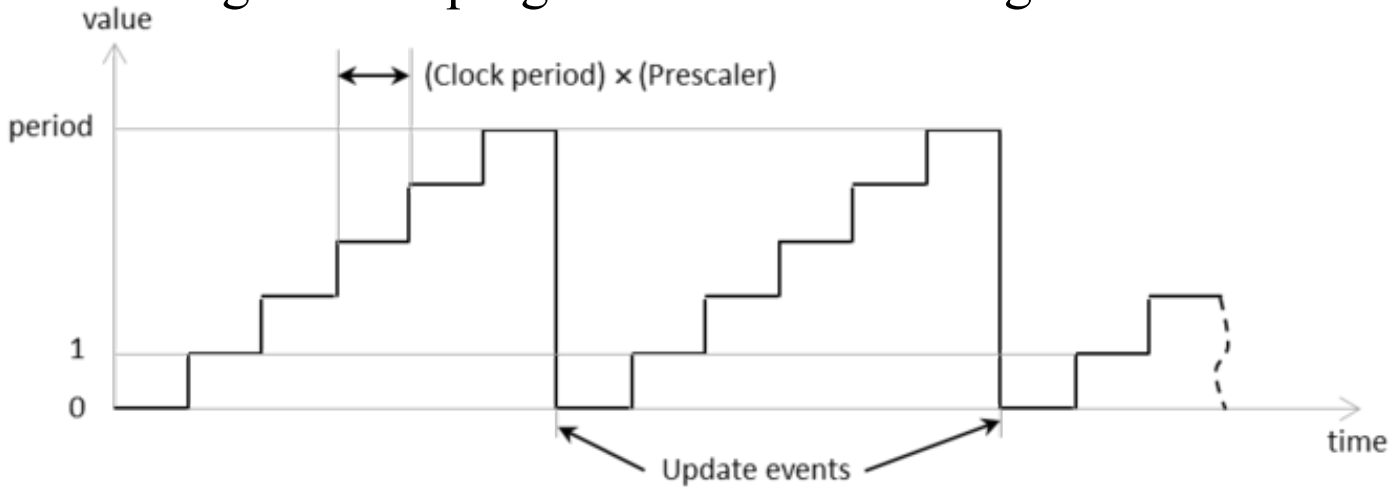
Anatomia di un Progetto

- Ogni progetto realizzato con ChibiStudio avrà al suo interno:
  - Una cartella build dove saranno contenuti i binari da caricare sulla scheda, .hex per ST-Link Utility e .elf per il debug
  - Configuration Headers (halconf.h e mcuconf.h) dove l'utente può abilitare le periferiche di interesse
  - Kernel configuration header (chconf.h) dove poter configurare il SO



Timer Hardware

- I timer hardware sono usati per:
  - Generare segnali a varie frequenze
  - Generare pulse-width-modulated output
  - Misurare il tempo trascorso tra due eventi di interesse
- Un timer hardware è un contatore che conta ad una certa velocità, definita dall'utente, da zero sino ad uno specifico valore di periodo preimpostato, generando *eventi* quando tale valore di periodo è raggiunto.
- Esegue in background al programma che si sta eseguendo.



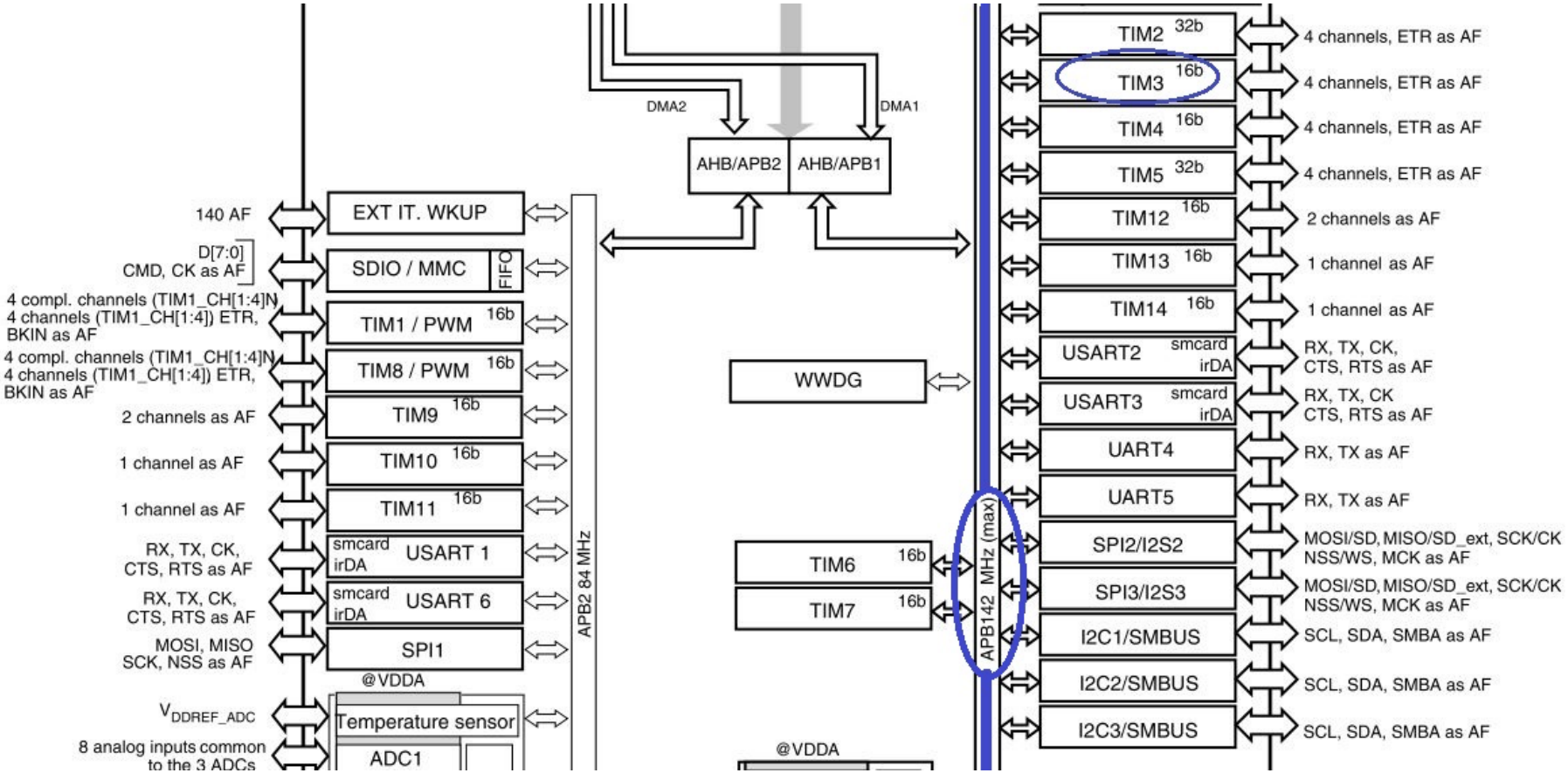
STM32F3 Timers #1

- Il microcontrollore della STM32F3-Discovery board è provvisto di dieci timer, di cui:
  - 6 sono *General-Purpose Timers* (TIM2 to TIM4 e TIM15 to TIM17) a 16 o a 32 bit
  - 2 sono *Advanced Timers* (TIM1 e TIM8) per, e.g., motor control
  - 2 sono *Basic Timers* (TIM6 and TIM7) usati ad esempio per fornire una base dei tempi per fare il ”trigger” dei convertitori DAC o ADC

STM32F3 Timers #2

- Tutti i timer chiaramente dipendono dalla frequenza del clock del processore, nella nostra board i timer:
  - Ricevono in input un clock a 72 o a 36MHz
  - Fanno uso di un *Prescaler* (a 16 bit) per dividere la frequenza del clock in input e gestire quindi frequenze di proprio interesse
- Possono contare sia verso l'alto (Upcounting), sia verso il basso (Downcounting)
- Raggiunto il valore di "*Auto-Reload*" il timer *setta* un *flag* nel suo status register per comunicare l'avvenuto raggiungimento del valore di auto-reload. Se configurato, il timer può generare un evento (un **interrupt**) per il microprocessore.

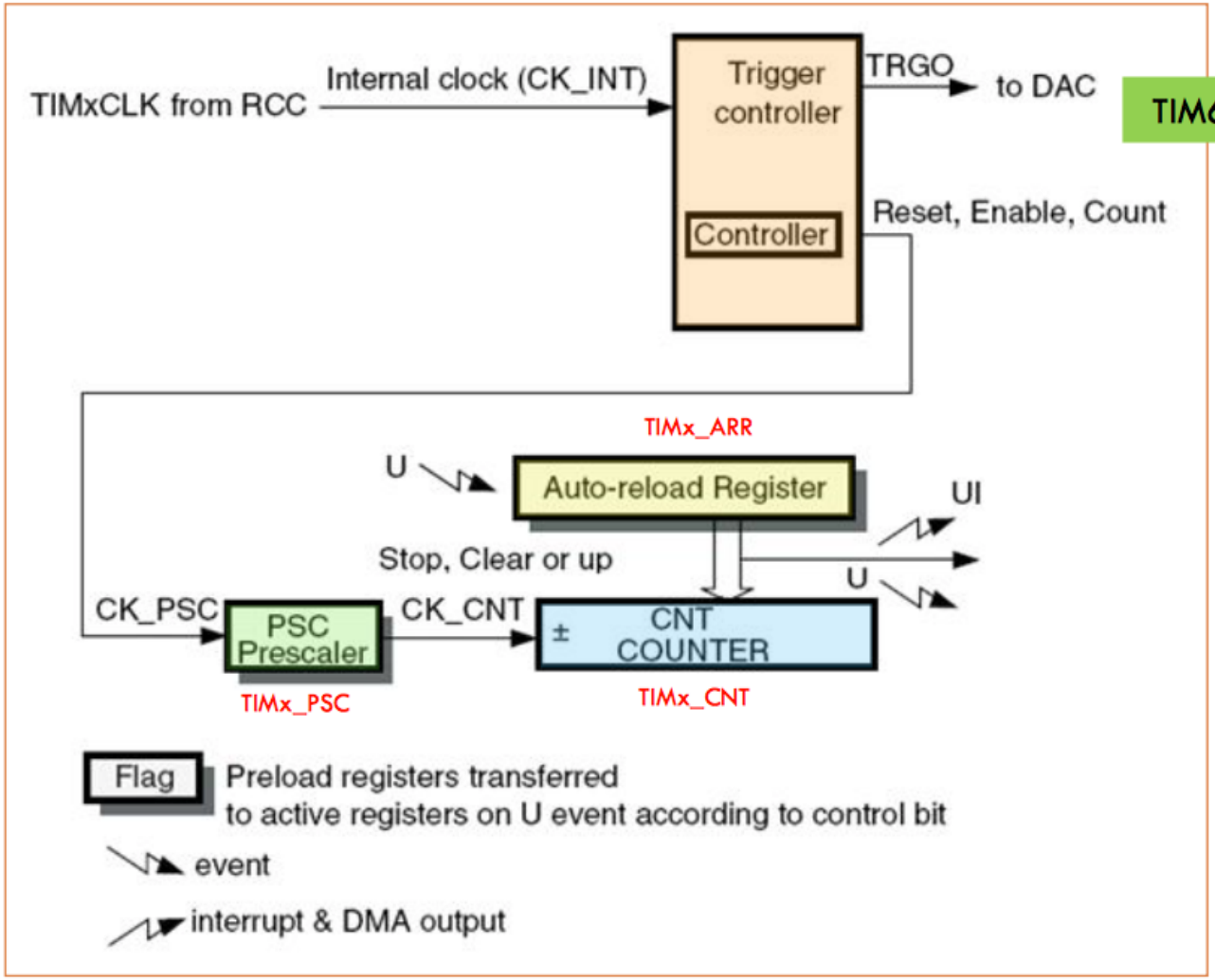
Il TIM nel SoC STM32F3



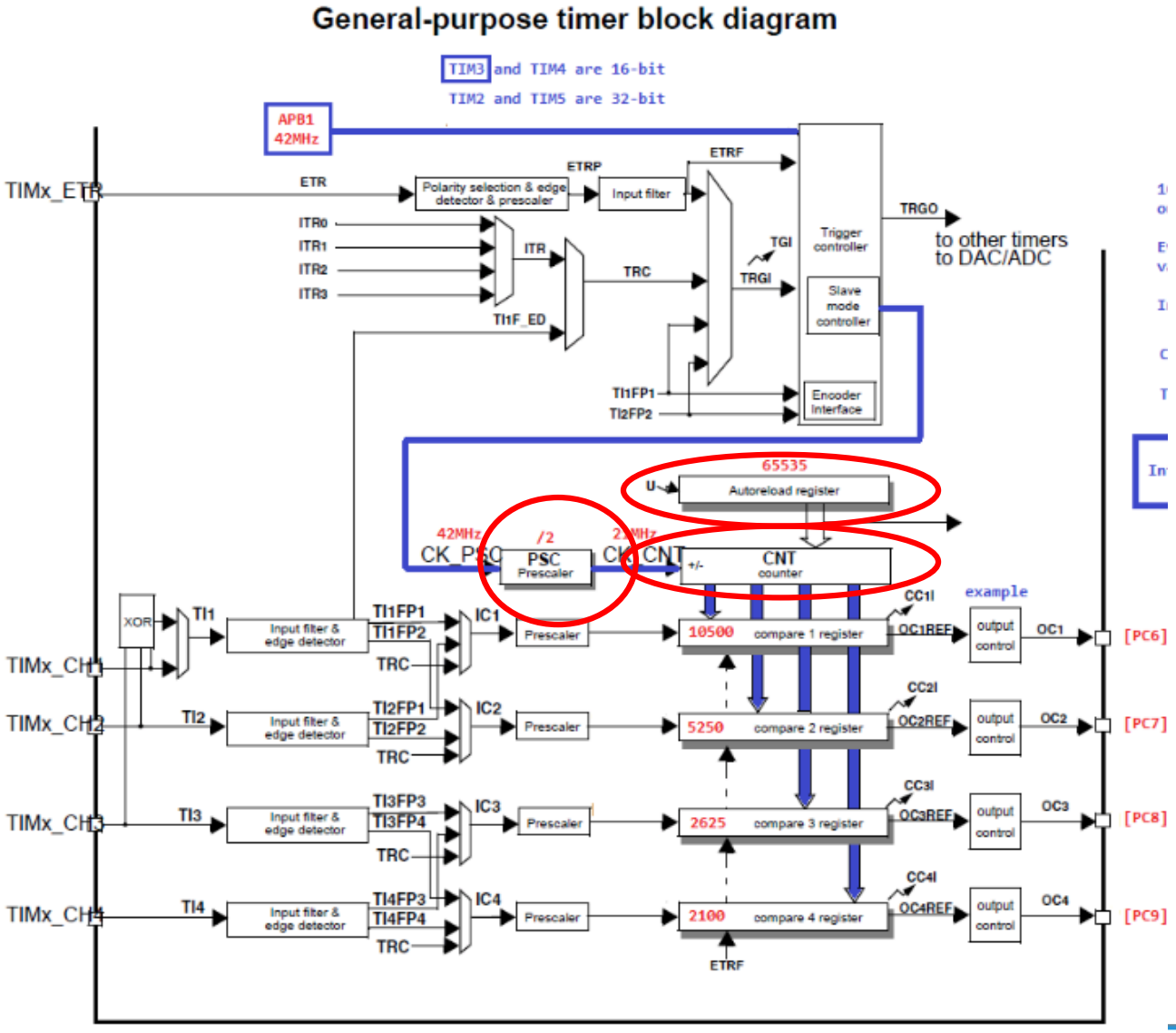
Principali Unità di un TIM

- I tre registri fondamentali nel counter sono:
  - Counter Register – contenente il valore corrente del contatore
  - Prescaler Register – registro al cui interno l'utente scrive (via software) il valore di divisione del clock
  - Auto-Reload Register – registro che contiene il valore finale di conteggio
- Dal Manuale:
  - "In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event."
  - "The prescaler can divide the counter clock frequency by any factor between 1 and 65536. It can be changed on the fly. The new prescaler ratio is taken into account at the next update event"

Basic Timer Scheme



Extended Block Diagram TIM



Modello di Programmazione

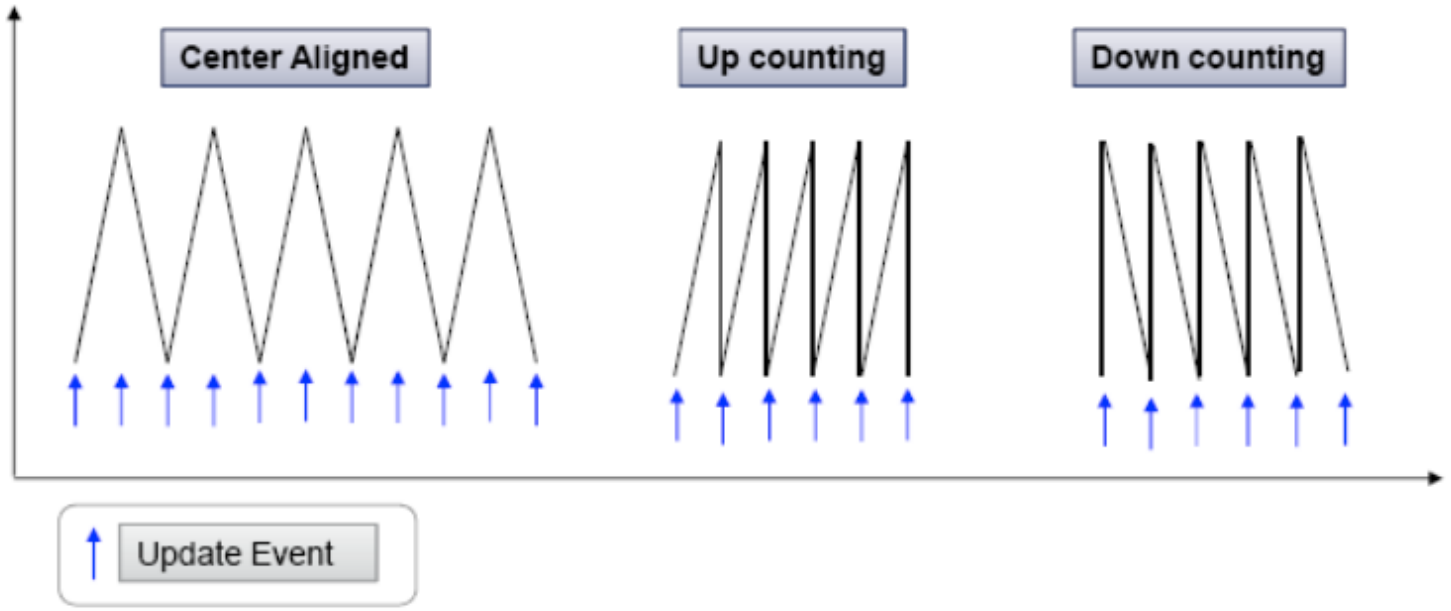
- Come ogni periferica, prima di essere utilizzata deve essere inizializzata impostando l'identificativo del TIM che si vuole usare (TIM1, TIM2, …), valore del prescaler, del periodo, ecc:

  (Vedere il manuale per informazioni dettagliate sui registri)

```
TIM_HandleTypeDef TimHandle;
TimHandle.Instance = TIMx;
TimHandle.Init.Period = PeriodValue;
TimHandle.Init.Prescaler = PrescalerValue;
TimHandle.Init.ClockDivision = 0;
TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;
HAL_TIM_Base_Init(&TimHandle)
```

Counting Modes #1

- Il counter può essere programmato per contare in tre modi differenti:
  - Up counting mode
  - Down counting mode
  - Center-aligned mode



Modi di Utilizzo del TIM

- Il timer TIM può essere utilizzato in due possibili modi:
  - In Polling: una volta impostata la frequenza desiderata si controlla di continuo il valore del contatore leggendo un registro specifico(all'interno di un ciclo while).
  - Con Interrupts: si imposta il timer affinché generi un interrupt (o un evento) quando il contatore raggiunge il valore di periodo impostato.

Esempio di Applicazione

- Solitamente il TIM lo si utilizza con le interruzioni. A cosa può servire il timer TIM con Interrupts?
  - Un esempio di applicazione di un timer potrebbe essere quello di un controllore di temperatura esterna su un automobile per far si che il guidatore riceva una notifica quando la temperatura è inferiore allo zero.
  - La temperatura di certo non può cambiare drasticamente ogni millisecondo. Dunque invece di controllare in *polling* (quindi in un ciclo *while*) il contenuto del registro sovraccaricando inutilmente il microcontrollore impostiamo un timer che ogni 30 secondi, ad esempio, verifiche il contenuto del registro dove è memorizzato il valore di temperatura misurato dal sensore della macchina

Dietro le Quinte

- Quando lo sviluppatore vuole effettuare un conteggio con interrupt cosa succede realmente?

Durante la fase di inizializzazione l'utente deve configurare il timer

bit

l

18.4.1 TIMx control register 1 (TIMx_CR1)

18.4.5 TIMx status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res.	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.
0: No update occurred.
1: Update interrupt pending. This bit is set by hardware when the registers are updated:

- At overflow or underflow (for TIM2 to TIM5) and if UDIS=0 in the TIMx_CR1 register.
- When CNT is reinitialized by software using the UG bit in TIMx_EGR register, if URS=0 and UDIS=0 in the TIMx_CR1 register.

When CNT is reinitialized by a trigger event (refer to the synchro control register description), if URS=0 and UDIS=0 in the TIMx_CR1 register.

l valore

preload

egister

re

Gestione Timer in ChibiOS

- ChibiOS offre delle librerie per la gestione dei timer: *hal_gpt*
- Per programmare un timer è necessario:
  1. *Definire* una *struttura* ***GPTConfig*** utile alla configurazione della frequenza di clock e alla definizione di una eventuale funzione di callback (se si opera con interrupt).
  2. *Inizializzare* ed attivare il timer utilizzando la funzione ***gptStart*** alla quale passare l'identificativo del timer che si vuole inizializzare e la struttura di configurazione prima definita
  3. Definire la modalità di funzionamento (ad es. Continuous mode) ed il periodo in *ticks*

Definizione del Clock Timer e del periodo di Tick

- Supponiamo di volere che il nostro Timer TIM2 invii un interrupt ogni ½ secondo, come definire il valore del clock timer e del periodo?
  - Una possibile soluzione sarebbe quella di impostare la **frequenza del clock in ingresso al timer** a 10000Hz=10KHz
  - Per avere un interrupt ogni ½ secondo dovremo contare ogni 5000 campioni della frequenza del clock in ingresso
  - Dunque il valore del **periodo** sarà di 5000
  - La formula da applicare è la seguente:
    - $t = (1/f_{ck}) * T_{tick}$