UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**

**CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE**

# Analisi delle caratteristiche della famiglia di microcontrollori STM32F334 basati su architettura ARM Cortex M4

**Relatore: Prof. Carlo De Santi**

**Laureando/a: Gecchelin Enrico**

**ANNO ACCADEMICO 2022 – 2023**

**Data di laurea 21/11/2023**

*INDICE*

## *INTRODUZIONE*

Era il 1971 quando Intel immetteva sul mercato il primo microprocessore a 4 bit della storia, l'Intel 4004, con una frequenza di clock di 740 kHz e costituito da circa 2300 transistor in logica PMOS[1].

Nel 1976 venne invece introdotto l'Intel 8048, il primo microcontrollore che integrava al suo interno 64 byte di RAM in logica NMOS, nel mentre anche altre aziende cominciavano ad affacciarsi in questo mercato, dando il via ad una sana competizione per poter presentare ogni anno un nuovo chip che avesse migliori prestazioni, maggiore quantità di memoria e nuove funzionalità.

Nel corso degli anni successivi, con l'avanzata tecnologica dei processi di fabbricazione, fu possibile integrare all'interno dei chip un numero sempre maggiore di periferiche: dalle prime applicazioni di capture e compare fu poi possibile integrare timers per il controllo PWM, per poi passare alle prime implementazioni dei protocolli di comunicazione seriale, dei convertitori A/D, e della FLASH EEPROM, introdotta da Toshiba negli anni 80.[2]

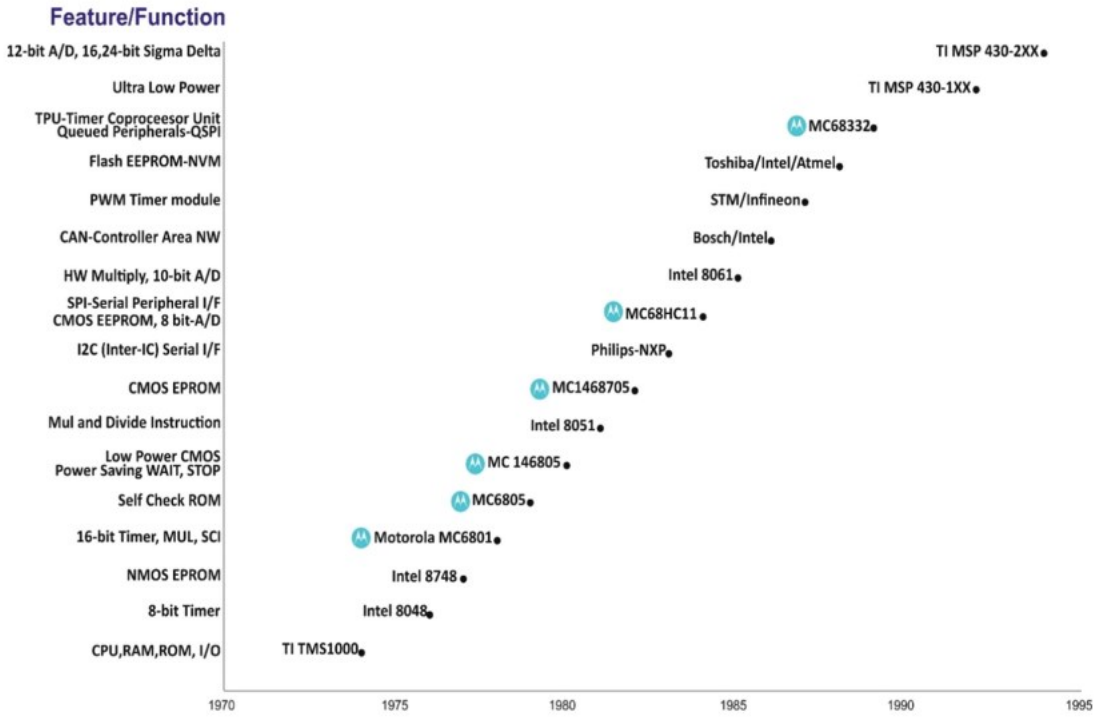


Figura 1: evoluzione delle periferiche presenti all'interno dei microcontrollori nel corso degli anni.[2]

All'inizio degli anni 90 i microcontrollori a 8 bit erano già una tecnologia presente i molti dispositivi e grazie alla corsa all'incremento prestazionale, cominciarono a divenire popolari architetture a 16 o 32 bit con elementi derivati dal mondo dei microprocessori, come l'organizzazione a pipeline, le

memorie cache e la gestione delle periferiche con DMA, in parallelo ad un incremento sempre maggiore delle frequenze di clock.

Ai giorni nostri, il livello di integrazione ha permesso di implementare miliardi di transistors all'interno di pochi centimetri quadrati di area di silicio e le unità di microcontrollori venduti dalla data della loro creazione è nell'ordine di grandezza delle migliaia di miliardi [1]; la presenza dei microcontrollori è ormai pervasiva in una moltitudine di applicazioni diverse che spaziano dalle applicazioni di controllo in tempo reale dei processi industriali, al mercato degli elettrodomestici, fino mondo in costante crescita dell'Internet Of Things.

Ciò detto, è quindi possibile affermare senza alcun dubbio che i circuiti integrati rappresentino una delle più importanti invenzioni del genere umano, con il loro contributo essenziale nel raggiungimento dell'attuale e futuro avanzamento scientifico della nostra civiltà.

# *1 I CORE ARM CORTEX E LA FAMIGLIA DI MICROCONTROLLORI STM32F334*

## *1.1 CENNI STORICI*

ARM (Advanced RISC Machines) nacque nel 1990 come joint venture tra Apple Computers, Acorn Computer Group e VLSI Technology. Il primo prodotto presentato sul mercato fu ARM6, microarchitettura RISC a 32 bit che venne utilizzata da Apple all'interno del suo Apple Newton Message Pad, un computer palmare antenato dei moderni iPhone.

Nel corso degli anni successivi, il modello di business adottato da ARM, basato solamente sulla vendita della licenza delle architetture e non sulla manifattura dei processori, ebbe successo e i suoi prodotti vennero adottati da un numero sempre maggiore di partners; all'inizio degli anni 2000, processori basati su architettura ARM erano presenti in una moltitudine di prodotti di enorme successo, tra i quali Game Boy Advance, Nintendo DS, iPod e Nokia N70.

Con la famiglia Cortex presentata nel 2005 l'offerta venne segmentata in 3 macrocategorie di prodotti distinti per prestazioni ottimizzati per applicazioni diverse:

- Cortex-A: processori di fascia high–end ad elevate prestazioni dedicati ad applicazioni integrate con sistemi operativi come smartphones, tablets, e televisioni.
- Cortex-R: processori ottimizzati per il controllo in tempo reale utilizzati all'interno di controllori per hard disk o in applicazioni automotive come gestione degli airbag o del sistema frenante.
- Cortex-M: architetture dedicate al mercato dei microcontrollori dove si sono imposte come standard, utilizzate dai principali produttori mondiali di chip come STMicroelectronics, NXP, Texas Instruments, Analog Devices e altri.

L'analisi di questa tesi si concentrerà sulle architetture Cortex-M ed in particolare modo su Cortex-M4.

## *1.2 ARM CORTEX M-4*

Attualmente la famiglia Cortex-M conta undici prodotti disponibili suddivisi in base a prestazioni e applicazioni. Tuttavia, vi sono alcune caratteristiche comuni che orientano la scelta dei progettisti verso questa categoria di prodotti, tra le quali troviamo sicuramente il buon rapporto tra prestazioni e consumo di potenza o l'efficiente ottimizzazione nella gestione delle interruzioni.

Cortex-M4 si pone nella fascia medio-alta della famiglia, con prestazioni e prezzo superiori a Cortex-M3 ma inferiori a Cortex-M7, il che lo rende un prodotto in grado di fornire buone prestazioni ad un costo contenuto. L'architettura fa uso di una pipeline organizzata in tre stadi (fetch, decode, execute),

con una Instruction Set Architecture dotata, tra le altre cose, di istruzioni per il calcolo in virgola mobile a singola precisione e per le applicazioni di Digital Signal Processing.

Attualmente ARM Cortex-M4 viene implementata da numerosi partners all'interno dei loro microcontrollori, tra cui per citarne alcuni:

- EFM32 Giant Gecko di Silicon Labs
- LPC4000 di NXP Semiconductors
- STM32F334 di StMicroelectronics

Il resto di questo elaborato verterà proprio sull'analisi di quest'ultima famiglia di microcontrollori.

### *1.3 LA FAMIGLIA DI MICROCONTROLLORI STM32F334*

La famiglia di microcontrollori che verrà presa in esame d'ora in poi si compone di otto prodotti diversi, essi sono:

- STM32F334C4
- STM32F334C6
- STM32F334C8
- STM32F334K4
- STM32F334K6
- STM32F334K8
- STM32F334R6
- STM32F334R8

Le poche differenze che intercorrono fra questi microcontrollori riguardano la quantità di memoria flash integrata (16, 32 o 64 kB) ed il package (32, 48 o 64 pins), le altre caratteristiche rimangono invece sostanzialmente invariate tra un modello e l'altro.

Tutti questi microcontrollori vengono venduti in formato LQFP (Low Profile Quad Flat Package) e perciò sono adatti al montaggio superficiale su scheda stampata; la stessa STMicroelectronics produce schede di sviluppo basate su questi microcontrollori come la NUCLEO-F334R8.

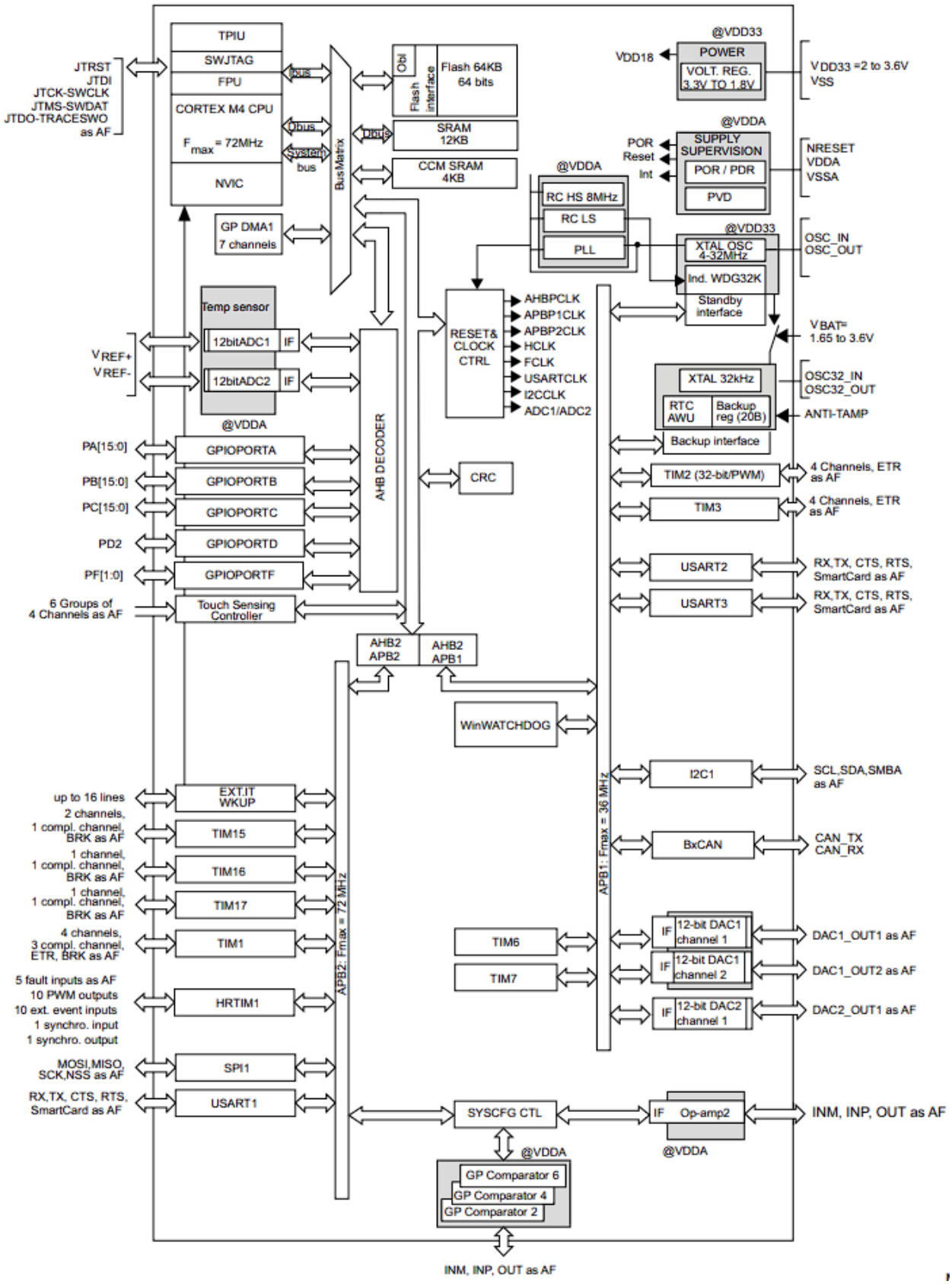


Figura 2: Schematico della struttura generale di un microcontrollore della famiglia STM32F334 [5]

La figura 2 mostra l'organizzazione circuitale dei microcontrollori in questione: partendo dall'angolo in alto a sinistra si trova il core RISC con FPU integrata ed operativo ad una frequenza di 72 Mhz, il quale comunica, tramite il bus matrix con le 3 memorie presenti, una flash e due RAM.

Collegati al bus ad elevate performance (Advanced High-performance Bus AHB) tramite un decoder si trovano i due ADC con il sensore di temperatura integrato, le porte di I/O (che possono arrivare ad un numero totale di 51 nei microcontrollori con package a 64 pin) e il touch sensing controller per ottimizzare l'eventuale implementazione di applicazioni con interfacce touch.

Al centro è presente il blocco di clock control, che, tramite opportuni oscillatori integrati o esterni, fornisce al programmatore un'ampia libertà nella gestione delle sorgenti di clock e delle loro relative frequenze.

Continuando a scendere, il bus AHB viene convertito nei bus per periferiche (Advanced Performance Bus APB), tramite due bridge AHB-APB, e la quasi totalità delle periferiche del sistema vengono direttamente connesse a questo bus. Tra di esse si trovano i numerosi timer presenti (fino a 12) aventi diverse funzionalità e caratteristiche, e i moduli di comunicazione distinti in in CAN, I2C, SPI, USART.

Sempre connessi al bus APB vi sono poi i due DAC e altre periferiche ausiliarie come i comparatori e l'amplificatore operazionale.

Infine, in alto a destra è presente il modulo di gestione delle tensioni di alimentazione.

# 2 MODI DI FUNZIONAMENTO E SISTEMA DI ALIMENTAZIONE

## 2.1 RISPARMIO ENERGETICO

I microcontrollori devono essere in grado di operare in una vasta gamma di situazioni diverse, per cui è necessario che il loro sistema di alimentazione e la loro architettura complessiva permettano l'implementazione di modalità di funzionamento in risparmio energetico. Soprattutto per quanto riguarda le applicazioni embedded, che sfruttano batterie per il loro funzionamento, è necessario tener conto del consumo di potenza che il dispositivo richiede e ottimizzare il progetto e la scelta dei componenti in funzione di questo parametro.

Per far fronte a queste esigenze, la famiglia di microcontrollori STM32F334 mette a disposizione del progettista tre diverse modalità di funzionamento a risparmio energetico:

- ***Sleep mode***

In questa modalità tutte le periferiche del microcontrollore continuano a funzionare e solamente il funzionamento della CPU viene sospeso mantenendo ad un livello logico basso il suo segnale di clock, per poi riprendere non appena venga ricevuta una interruzione.

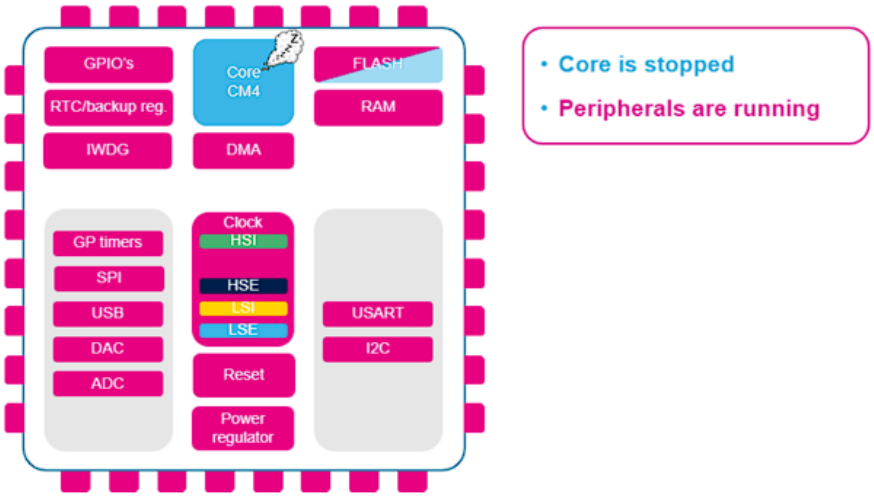


Figura 3: Parti del sistema attive e non in sleep mode.[4]

- ***Stop mode***

La stop mode permette di raggiungere il minimo consumo di potenza conservando i dati contenuti nei registri e nella SRAM. Tutti gli oscillatori per la generazione dei clock di sistema ad alte prestazioni sono disabilitati, mentre possono rimanere attivi quelli a basse prestazioni.

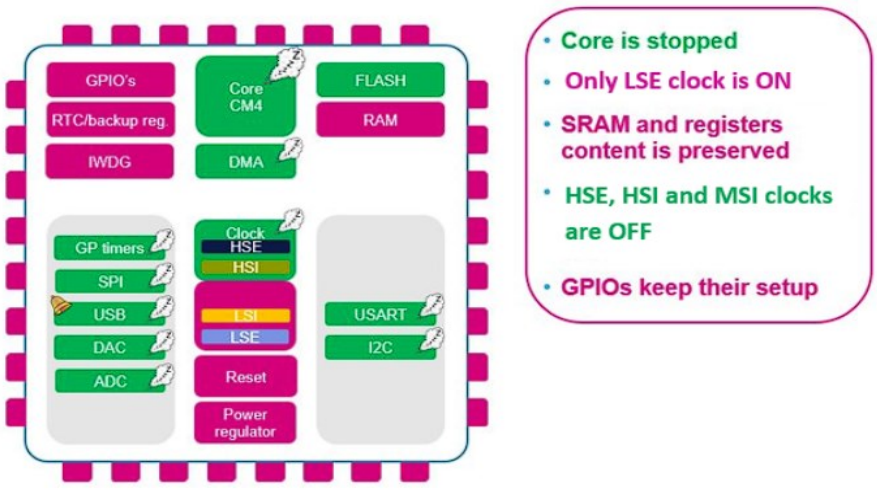


Figura 4: Parti del sistema attive e non in Stop mode.[4]

- ***Standby mode***

Questa modalità si distingue dalla precedente per il fatto che in questo caso i dati contenuti in RAM vengono persi e l'uscita dalla standby mode è equivalente alla riesecuzione completa del programma, analogamente a quanto succede dopo un reset di sistema. Questa modalità di funzionamento permette il massimo risparmio energetico fra tutte le modalità analizzate.

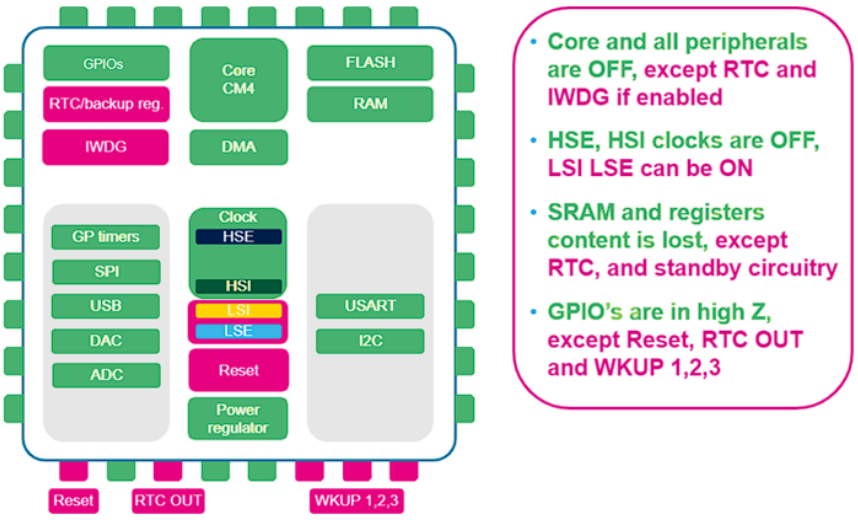


Figura 5: Parti del sistema attive e non in Standby mode.[4]

Una spiegazione maggiormente approfondita di tali modalità di funzionamento verrà fornita nei prossimi paragrafi.

### *2.2 ALIMENTAZIONE DEL MICROCONTROLLORE*

Diverse tensioni di alimentazione sono disponibili all'interno del microcontrollore, distinte in base a quali parti del microcontrollore devono alimentare:

$V_{SS}$ e $V_{DD}$ = da 2.0 a 3.6 V

$V_{SSA}$ e $V_{DDA}$ = da 2.0 a 3.6 V

$V_{DD18}$ = da 1.65 a 1.95 V

$V_{BAT}$ = da 1.65 a 3.6 V

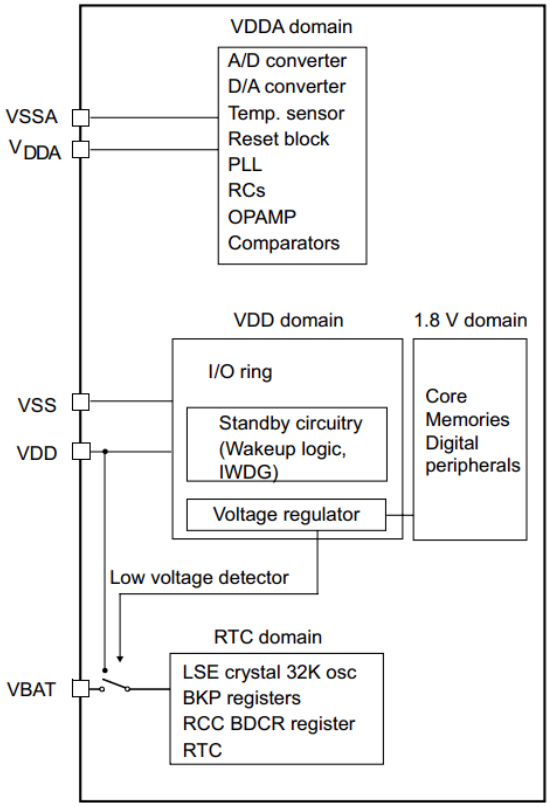


Figura 6: organizzazione del modulo di alimentazione. [3]

Dall'immagine si vede come le due tensioni $V_{SSA}$ e $V_{DDA}$, fornite dall'esterno, servano come tensioni di riferimento per ADC e DAC o alternativamente come tensioni di alimentazione per altre periferiche integrate all'interno di questi microcontrollori come il sensore di temperatura, il PLL, l'OPAMP e i diversi comparatori presenti.

$V_{SS}$ e $V_{DD}$, anch'esse fornite dall'esterno, servono invece ad alimentare la circuiteria adibita all'I/O ed inoltre forniscono l'alimentazione del core, delle memorie e in generale di tutta la circuiteria digitale in seguito alla regolazione di $V_{DD}$ al valore di 1.8V = $V_{DD18}$.

$V_{BAT}$ funge da alimentazione di "backup" nel caso venisse a mancare per un certo intervallo di tempo $V_{DD}$. In mancanza di tale tensione, infatti, viene attivato lo switch che connette $V_{BAT}$ ai registri di backup (utili per la memorizzazione di dati importanti quali configurazioni o dati di calibrazione), al Real Time Clock, al Low Speed External Oscillator permettendo quindi di fatto di svolgere alcune funzioni anche in caso di temporanea mancanza di alimentazione.

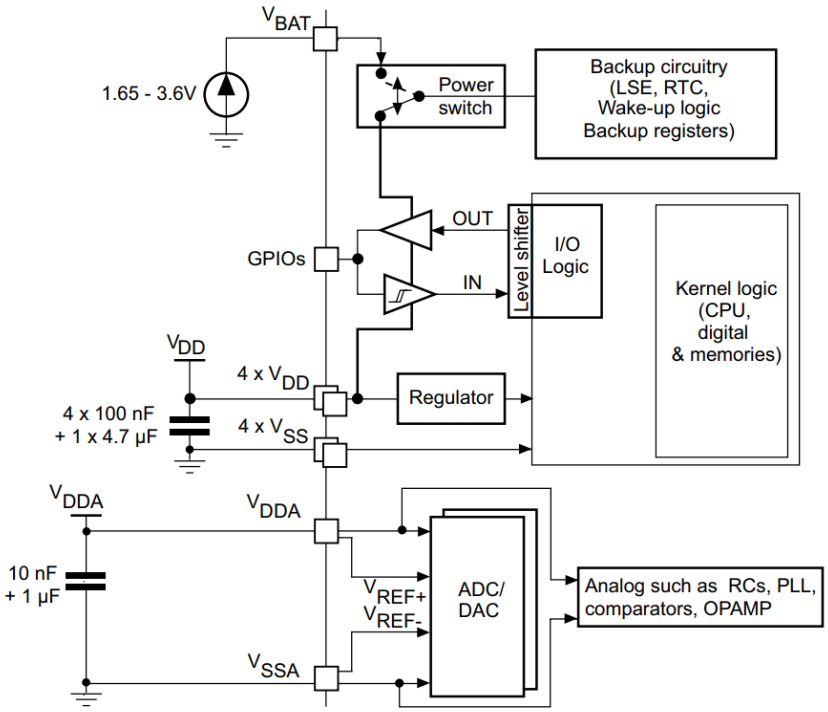


Figura 7: schema del modulo di alimentazione. [5]

Lo schematico in figura 7 chiarisce ancora meglio l'organizzazione circuitale sopra esposta; in particolare, si vede come ADC e DAC dispongano di due linee di alimentazione indipendenti, opportunamente schermate per ridurre la probabilità di errori durante le conversioni digitale/analogico o analogico/digitale. Inoltre, si noti anche come lo stesso schematico suggerisca di porre dei condensatori tra l'alimentazione e la massa: questa è una pratica comune per la riduzione di eventuali rumori alle alte frequenze e perciò contribuisce al mantenimento di una tensione di alimentazione stabile.

## *2.3 MODI DI FUNZIONAMENTO*

I microcontrollori possono essere utilizzati per molti scopi diversi, dal semplice monitoraggio di caratteristiche ambientali tramite sensori all'implementazione di algoritmi avanzati di controllo o signal processing. Nell'ottica della prima situazione, è necessario che essi mettano quindi a disposizione del programmatore delle modalità di funzionamento atte ad un miglior efficientamento delle risorse energetiche. Supponendo infatti di utilizzare un microcontrollore per il monitoraggio di un parametro ambientale poco variabile nel tempo (ad esempio la temperatura o l'umidità dell'aria), risulta poco sensato che esso venga costantemente misurato milioni di volte al secondo. In questo caso è quindi utile ricorrere ad una modalità di risparmio energetico che viene mantenuta tra una misurazione e l'altra, e scegliere una temporizzazione tra una misura e l'altra più adeguata (nel caso in esame potrebbe essere una misura al minuto).

La modalità operativa di default del microcontrollore è la run mode, ovvero la classica modalità di funzionamento in cui l'esecuzione delle istruzioni procede ad ogni ciclo di clock e nessuna modalità di risparmio energetico viene attivata; tuttavia, questo non significa che non sia possibile mettere in atto delle strategie aumentare l'efficienza del sistema pur rimanendo in run mode.

Il progettista ha infatti la possibilità di modificare tramite software le velocità di clock del sistema tramite gli opportuni registri dei prescaler, adattando il suo sistema alle specifiche effettivamente necessarie.

### *2.3.1 SLEEP MODE*

In questa modalità viene sospeso il funzionamento del solo core, mentre il regolatore di tensione continua funzionare normalmente fornendo la tensione di alimentazione di 1.8 V a tutte le periferiche logiche del microcontrollore. Pur essendo sospeso il core, altre periferiche come ADC e timers possono comunque continuare a funzionare anche se il sistema non esegue istruzioni attive, per poter gestire l'uscita dallo sleep mode in seguito a un'interruzione generata dall'esterno.

Il sistema entra in sleep mode tramite due possibili istruzioni presenti nell'Instruction Set Architecture di ARM Cortex M4, WFI (Wait For Interrupt) e WFE (Wait for Event); alternativamente è possibile far entrare il microcontrollore in sleep mode automaticamente alla fine dell'esecuzione della ISR a minima priorità settando a 1 il bit SLEEPONEXIT presente nel system control register del core.

| Mnemonic | Operands | Brief description | Flags |
|---|---|---|---|
| VMOV | Dd[x], Rt | Copy Arm core register to scalar | — |
| VMOV | Rt, Dn[x] | Copy scalar to Arm core register | — |
| VMRS | Rt, FPSCR | Move FPSCR to Arm core register or APSR | N,Z,C,V |
| VMSR | FPSCR, Rt | Move to FPSCR from Arm Core register | FPSCR |
| VMUL.F32 | {Sd,} Sn, Sm | Floating-point multiply | — |
| VNEG.F32 | Sd, Sm | Floating-point negate | — |
| VNMLA.F32 | Sd, Sn, Sm | Floating-point multiply and add | — |
| VNMLS.F32 | Sd, Sn, Sm | Floating-point multiply and subtract | — |
| VNMUL | {Sd,} Sn, Sm | Floating-point multiply | — |
| VPOP | list | Pop extension registers | — |
| VPUSH | list | Push extension registers | — |
| VSQRT.F32 | Sd, Sm | Calculates floating-point square root | — |
| VSTM | Rn{!}, list | Floating-point register store multiple | — |
| WFE | — | Wait for event | — |
| WFI | — | Wait for interrupt | — |

Figura 8: Nell'immagine sono evidenziate le due istruzioni WFE e WFI presenti nell'ISA del Cortex M4.[7]

Qualsiasi interruzione può far uscire il sistema dallo sleep mode se l'accesso a quest'ultimo è avvenuto tramite WFI o Sleep-on-exit.

### *2.3.2 STOP MODE*

In questa modalità il regolatore di tensione interno può funzionare normalmente oppure in low-power mode, attivabile modificando un opportuno bit del power control register del regolatore. A differenza dello sleep mode illustrato precedentemente, in questo caso molte periferiche vengono spente, perlomeno quelle aventi come clock le sorgenti ad alte prestazioni.

Lo stop mode permette di mantenere i dati presenti in RAM e le configurazioni dei pin di I/O ed anche in questo caso è accessibile tramite WFI o WFE, ma solo dopo aver posto a 1 il bit SLEEPDEEP del System Control Resister del core; inoltre utilizzando la low-power mode del regolatore di tensione, viene introdotto un ritardo nel passare dallo stop mode al run mode.

### *2.3.3 STANDBY MODE*

In questa modalità il regolatore di tensione è spento, con la conseguenza di avere un forte guadagno in efficienza energetica ma un importante aumento in termini del tempo necessario per uscire da tale modalità, che coincide ad un reset del sistema.

| Symbol | Parameter | Conditions | Typ. @$V_{DD}$, $V_{DD}$ = $V_{DDA}$ | | | | | | Max. | Unit |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2.0 V | 2.4 V | 2.7 V | 3 V | 3.3 V | 3.6 V | | |
| $t_{WUSTOP}$ | Wakeup from Stop mode | Regulator in run mode | 4.3 | 4.1 | 4.0 | 3.9 | 3.8 | 3.7 | 4.5 | µs |
| | | Regulator in low-power mode | 7.8 | 6.7 | 6.1 | 5.9 | 5.5 | 5.3 | 9 | |
| $t_{WUSTANDBY}$(1) | Wakeup from Standby mode | LSI and IWDG OFF | 74.4 | 64.3 | 60.0 | 56.9 | 54.3 | 51.1 | 103 | |
| $t_{WUSLEEP}$ | Wakeup from Sleep mode | - | 6 | | | | | | - | CPU clock cycles |

Figura 9: confronto dei tempi di latenza tra le diverse modalità di risparmio energetico [5]

In tabella si possono vedere confrontati i tempi relativi al ripristino del run mode per ogni modalità di funzionamento in risparmio energetico: si può notare come in stop mode l'utilizzo o meno della low-power mode del regolatore vada ad avere un effetto tutto sommato limitato sull'aumento dei tempi di latenza, cosa che non succede facendo ricorso allo standby mode, che invece porta ad un incremento sostanziale per il ripristino del normale funzionamento del sistema.

Per quanto riguarda la modalità sleep, quest'ultima è sicuramente la modalità che consente di ottenere il più basso tempo di ripristino; supponendo infatti di usare il microcontrollore alla sua massima frequenza di clock di 72 MHz, il tempo di latenza risulta limitato ad appena 0.083 µs.

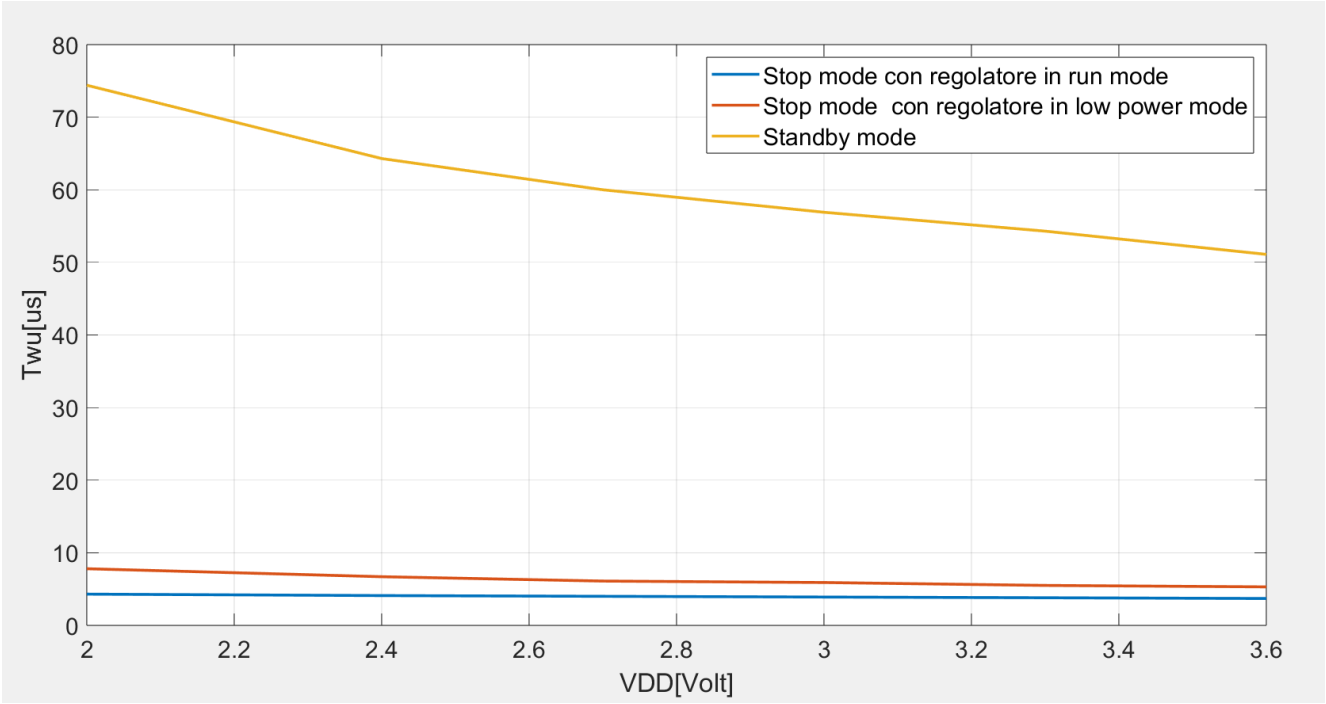


Figura 10: tempi di ripristino rappresentati graficamente.

In figura sono rappresentati i dati dei tempi di latenza in funzione della tensione di alimentazione: in blu è rappresentato lo stop mode con regolatore in run mode, in arancione lo stop mode con regolatore in low-power mode e in giallo lo standby mode; si nota a colpo d'occhio come l'ultima modalità abbia tempi di ripristino molto maggiori rispetto alle altre due, anche se comunque comparabile in termini di ordine di grandezza.

# *3 SISTEMA DI BUS E DMA*

## *3.1 PROTOCOLLI DEI BUS*

Facendo riferimento allo schematico di pagina 5 si nota che il sistema utilizza due diversi standard di bus, AHB e APB, facenti parti di AMBA (Advanced Microcontroller Bus Architecture), open standard introdotto per la prima volta nel 1996 da ARM e ora utilizzato in una ampia gamma di dispositivi, tra cui anche SoC per i moderni smartphones.

Lo standard AHB (Advanced High Performance Bus) viene utilizzato all'interno del sistema come protocollo principale, utile per connettere tra loro i bus del Core, il DMA, le porte di I/O e le memorie, ovvero le periferiche che necessitano di una comunicazione veloce e performante; esso permette infatti di raggiungere una elevata larghezza di banda grazie alla sua architettura a pipeline organizzata in due diverse fasi:

-Address phase: in questa fase il bus master trasferisce l'indirizzo della periferica con la quale vuole comunicare nel bus.

-Data phase: in questa fase avviene la effettiva comunicazione, con il master che invia o riceve dati da uno slave.

Alla fine di questa fase, inoltre, lo slave comunica al master la buona riuscita o meno della comunicazione.

La trasmissione di informazioni con le altre periferiche, tra cui possiamo trovare, tra le altre, DAC, timers e moduli I2C, avviene invece tramite uno standard diverso: APB (Advanced Peripheral Bus); esso presenta una complessità e un consumo di potenza minore rispetto all'AHB e per questo motivo viene utilizzato per il collegamento di periferiche che richiedono un minore dispendio di risorse, permettendo di ottenere maggior efficientamento del sistema in termini di costo e consumo di potenza.

Lo standard APB non è implementato con una struttura a pipeline e le comunicazioni possono richiedere anche diversi cicli di clock prima di essere completate.

La conversione tra uno standard ed un altro viene effettuata all'interno del microcontrollore tramite opportuni bridge, come si vede ancora una volta nello schematico a pagina 5.

### *3.2 IL BUS MATRIX*

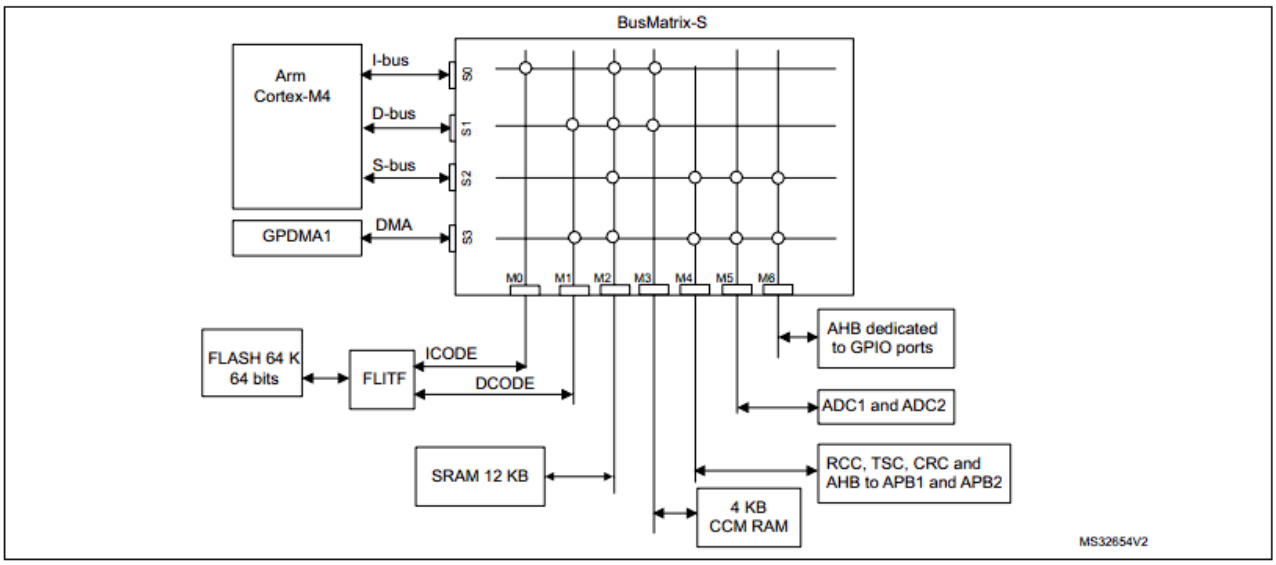


Figura 11: schema dei collegamenti del bus matrix. [3]

L'immagine mostra come sono organizzate le connessioni tra le diverse parti del sistema, composto da 4 masters e 7 slaves; verranno ora analizzati più nel dettaglio i collegamenti tra queste e le loro funzioni.

I 4 master che compongono il sistema sono:

- I-bus
- D-bus
- S-bus
- GPDMA1 (general purpose DMA)

Quest'ultimo verrà analizzato nel dettaglio nei prossimi paragrafi, mentre i primi tre sono i bus provenienti direttamente dal Core Cortex M4; l'insieme degli slaves è invece costituito dalle memorie (organizzate in una struttura gerarchica), dalle porte di I/O, dai due ADC e dai bridge AHB-APB.

I-bus è il bus istruzioni del core, esso infatti è messo in collegamento con le tre memorie del sistema dalle quali preleva le istruzioni; in particolare, la presenza di due memorie RAM permette di configurare il sistema con architettura Harvard, utilizzando due memorie distinte per istruzioni e dati.

Un discorso analogo vale per D-bus, che è invece il bus dati del core.

S-bus (System bus) è invece utilizzato per l'accesso di dati dalla RAM o dalle periferiche.

### *3.3 L'ALGORITMO ROUND ROBIN*

Il BusMatrix fa uso di un algoritmo chiamato Round Robin per la gestione delle richieste di comunicazione da parte dei masters; questo algoritmo è uno dei più semplici ma allo stesso tempo più largamente utilizzati per la gestione di questo tipo di processi.

L'idea alla base dell'algoritmo è quella di avere una coda di processi da eseguire, in cui l'ordine di esecuzione è dettato dalla sequenza temporale in cui le richieste di esecuzione sono arrivate, e di un intervallo di tempo fissato (chiamato "quantum time") entro cui queste devono essere eseguite.

Se il tempo di esecuzione del processo è minore del quantum time, la coda continua imperturbata nel suo svolgimento, altrimenti l'esecuzione corrente viene fermata e il processo viene spostato nell'ultima posizione della coda.

Un esempio aiuta a chiarire il concetto: si supponga di avere 5 processi, P1, …, P5 e di conoscere i loro burst time (ovvero il loro tempo di esecuzione); si supponga inoltre di avere un quantum time fissato pari a 4.

| Process | Burst time |
|---|---|
| P1 | 20 |
| P2 | 12 |
| P3 | 8 |
| P4 | 16 |
| P5 | 4 |

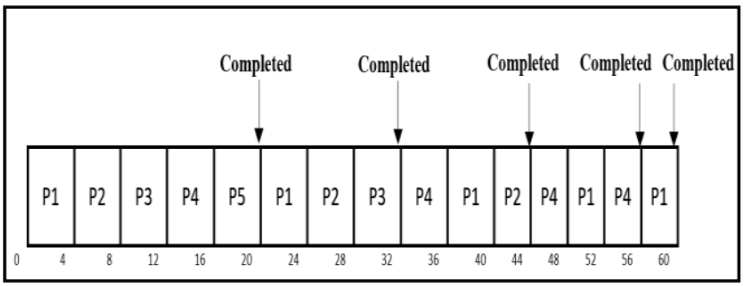


Figura 12: esempio di funzionamento dell'algoritmo di Round Robin.[4]

P1, avendo un burst time superiore a 4, non viene completato e viene quindi inserito alla fine della coda; lo stesso discorso vale per tutti gli altri processi tranne per P5, che invece viene completato e rimosso dalla coda. A questo punto il processo continua nello stesso modo fino al completamento di tutte le richieste rimaste in attesa.

Ovviamente l'efficienza di tale algoritmo è dettata da una scelta corretta del quantum time in quanto una scelta di un intervallo di tempo troppo grande porterebbe ad una implementazione FCFS (First Come,First Solved), mentre un intervallo troppo ristretto porterebbe ad avere tempi di esecuzione troppo elevati sui singoli processi.

### *3.4 DMA*

Nei moderni microcontrollori, anche in quelli di fascia più bassa, la gestione delle richieste di interruzione da parte delle periferiche di I/O viene delegata ad una periferica chiamata Direct Memory Access Controller, che è in grado di far comunicare tali periferiche con le memorie del sistema senza coinvolgere la CPU.

Questa organizzazione viene adottata anche all'interno della famiglia di microcontrollori qui presa in esame, dove il Direct Memory Access Controller è composto da sette canali diversi con differenti priorità ed è in grado di generare una richiesta di interruzione una volta che il trasferimento dei dati sia stato completato.

Per comprendere la struttura e il funzionamento di questo controllore è utile partire dall'organizzazione dei suoi canali:

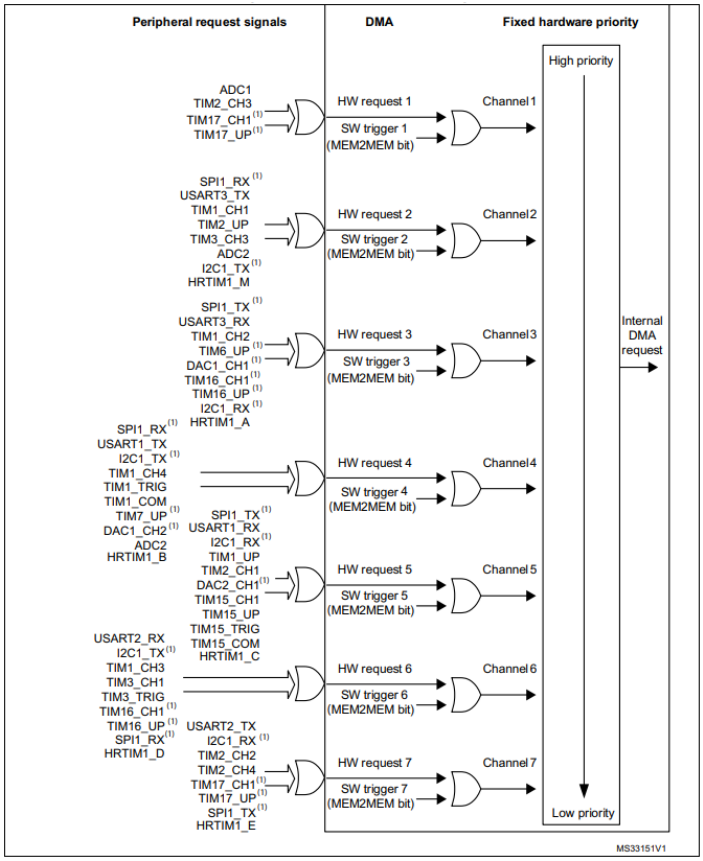


Figura 13: organizzazione dei canali del DMA. [3]

Si può vedere come i canali, numerati da 1 a 7, siano disposti in maniera tale da avere una priorità decrescente e come ad un singolo canale siano collegate anche diverse periferiche, che possono però essere gestite una alla volta essendo poste in OR logico tra di loro. Tramite software è possibile impostare quattro livelli diversi di priorità per ogni canale (molto alto, alto, medio, basso) e, in caso

di conflitto tra richieste aventi lo stesso livello, interviene l'organizzazione hardware a stabilire l'ordine di esecuzione delle stesse.

A titolo di esempio, supponendo di avere due richieste entrambe con un livello di priorità alto, ma appartenenti a canali diversi, il controller darà la precedenza a quella appartenente al canale numerato col numero inferiore.

Avendo capito come sono strutturati i canali interni al controller, si cercherà ora di comprendere come l'approccio DMA è implementato all'interno del sistema complessivo:

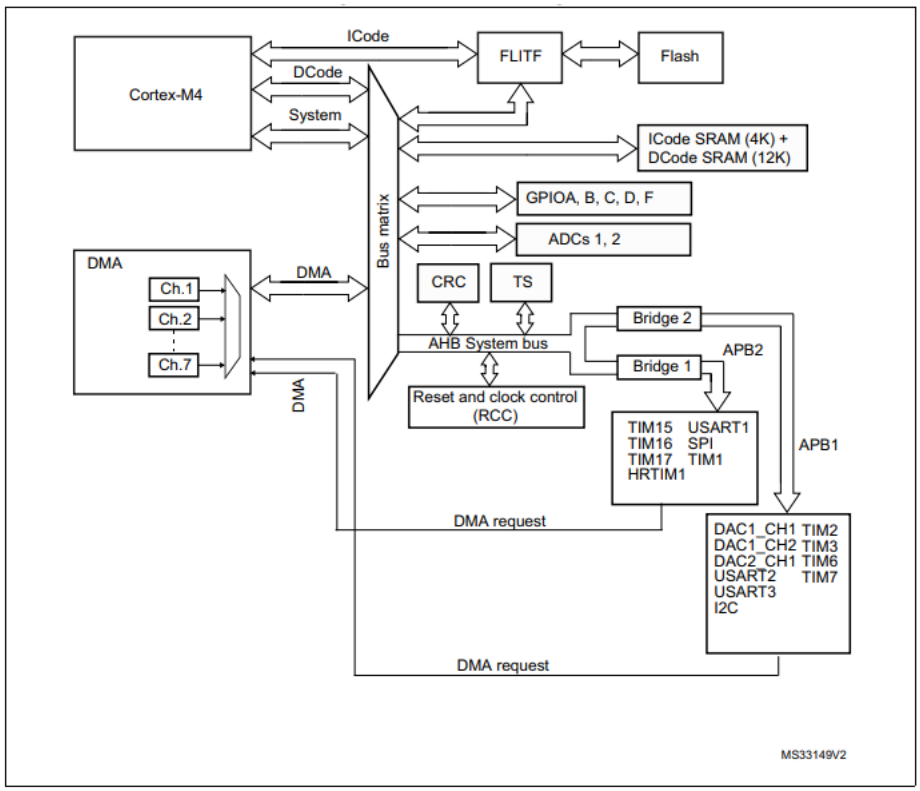


Figura 14: interconnessioni tra il DMAC e le altre parti del microcontrollore. [3]

Questa organizzazione circuitale in cui il BusMatrix viene condiviso tra il controllore e le memorie permette di effettuare trasmissioni dirette fra le due parti (o eventualmente anche tra periferica e periferica o memoria e memoria) senza l'intervento del processore.

Le periferiche poste in seguito ai bridge AHB-APB invece hanno delle linee dedicate di collegamento al controller, non essendo direttamente connesse al BusMatrix.

La comunicazione comincia in seguito alla ricezione da parte del controller di una richiesta di interruzione proveniente da una periferica (o via software), che verrà servita in base alla sua priorità in accordo con quanto esposto in precedenza. Una volta servita la richiesta e terminata la trasmissione, il controller invia un segnale di acknowledge alla periferica che, dopo averlo ricevuto, provvederà ad

abbassare la linea associata alla richiesta. Questa organizzazione permette chiaramente una comunicazione bilaterale tra le parti coinvolte.

La grandezza del singolo trasferimento è ovviamente programmabile, potendo scegliere di trasmettere 8,16 o 32 bit, modificando le opportune sezioni del registro di configurazione del controller. Inoltre, è anche possibile decidere di non trasmettere solamente uno di questi "pacchetti" ma bensì più di uno: in questo caso, è necessario impostare preventivamente il numero di singoli trasferimenti da effettuare in un apposito registro, il cui valore verrà quindi decrementato alla fine di ogni ciclo fino al raggiungimento dello zero.

## *4 GESTIONE DEL CLOCK*

I microcontrollori in esame integrano diverse modalità di gestione del clock e delle sorgenti dello stesso, offrendo al programmatore un'ampia libertà per la configurazione della velocità del sistema in base all'applicazione da sviluppare.

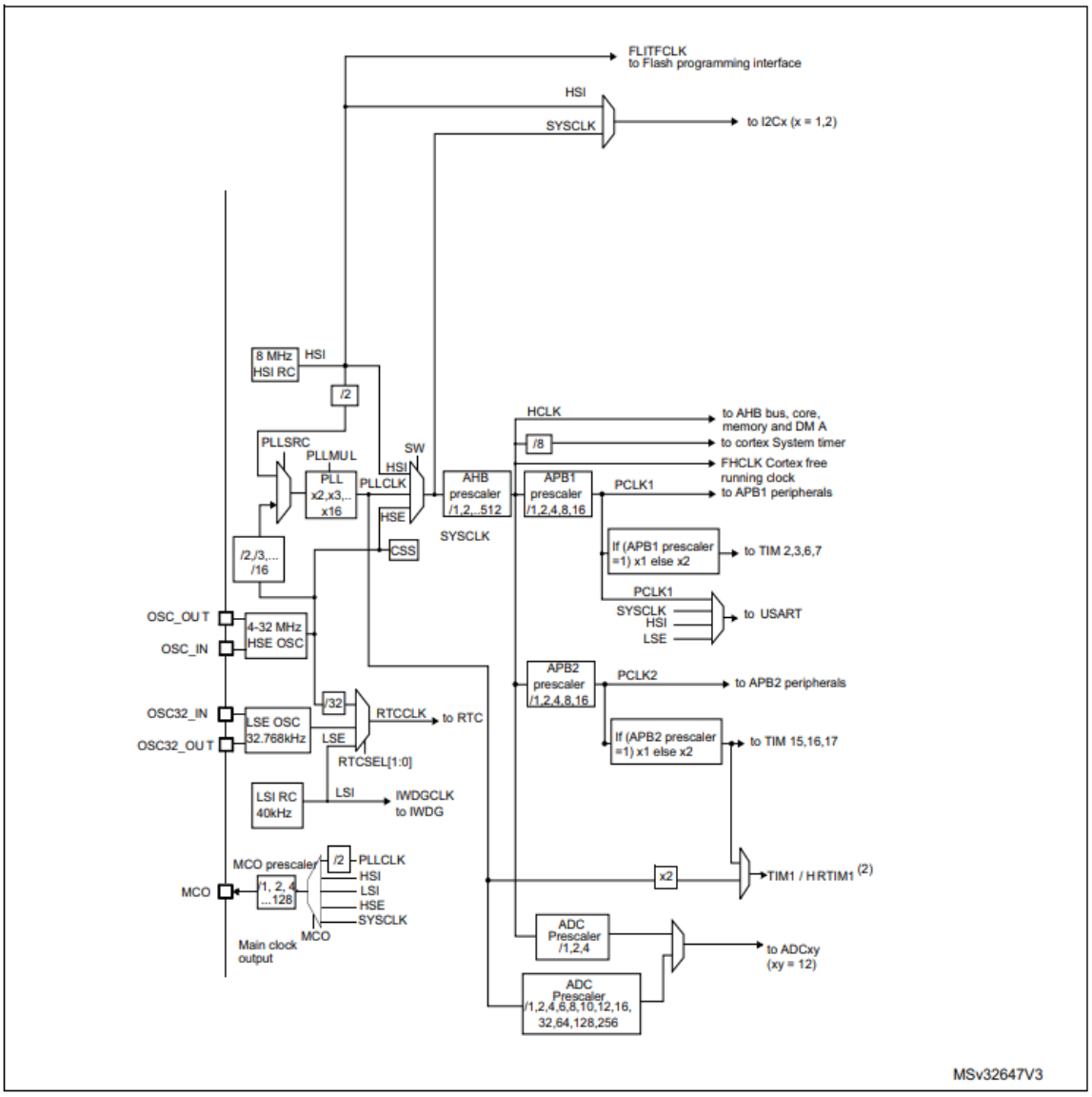


Figura 15: clock tree dei microcontrollori STM32F334. [3]

### *4.1 SORGENTI DI CLOCK DI SISTEMA*

La frequenza di clock di sistema, come si vede dallo schematico in Figura x, può essere originata da tre diverse sorgenti multiplexate tra di loro:

- HSE
- HSI
- PLL

Inoltre, la presenza di un prescaler per ogni bus garantisce un'ampia libertà nella gestione delle prestazioni del sistema: velocità di clock dei bus superiori implicano quindi anche maggiori prestazioni delle periferiche a esse connesse, causando però un aumento del consumo di potenza.

### *4.1.1 HSI (HIGH SPEED INTERNAL CLOCK SIGNAL)*

HSI è la sorgente di clock utilizzata di default dal microcontrollore e consiste in un oscillatore RC integrato in grado di generare un segnale di clock con una frequenza di 8MHz con un duty cycle compreso tra il 45% ed il 55%, che può essere utilizzato direttamente come clock oppure diviso per due ed utilizzato come input per il PLL. Questa implementazione ha il vantaggio di non richiedere circuiteria aggiuntiva per la generazione del segnale, ma la frequenza risulta meno accurata rispetto all'implementazione tramite oscillatore esterno.

| Symbol | Parameter | Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $f_{HSI}$ | Frequency | - | - | 8 | - | MHz |
| TRIM | HSI user trimming step | - | - | - | 1(2) | % |
| $DuCy_{(HSI)}$ | Duty cycle | - | 45(2) | - | 55(2) | % |
| $ACC_{HSI}$ | Accuracy of the HSI oscillator (factory calibrated) | $T_A$ = –40 to 105 °C | –2.8(3) | - | 3.8(3) | % |
| | | $T_A$ = –10 to 85 °C | –1.9(3) | - | 2.3(3) | |
| | | $T_A$ = 0 to 85 °C | -1.9(3) | - | 2(3) | |
| | | $T_A$ = 0 to 70 °C | -1.3(3) | - | 2(3) | |
| | | $T_A$ = 0 to 55 °C | –1(3) | - | 2(3) | |
| | | $T_A$ = 25 °C(4) | –1 | - | 1 | |
| $t_{su(HSI)}$ | HSI oscillator startup time | - | 1(2) | - | 2(2) | µs |
| $I_{DDA(HSI)}$ | HSI oscillator power consumption | - | - | 80 | 100(2) | µA |

1. $V_{DDA}$ = 3.3 V, $T_A$ = –40 to 105 °C unless otherwise specified.
2. Guaranteed by design, not tested in production.
3. Data based on characterization results, not tested in production.
4. Factory calibrated, parts not soldered

Figura 16: caratteristiche e prestazioni della sorgente HSI. [5]

In figura 16 sono riassunte le specifiche della sorgente di clock appena discussa; tra le altre cose, è interessante notare come l'accuratezza del dispositivo venga influenzata in maniera non trascurabile dalla temperatura di utilizzo.

### *4.1.2 HSE (HIGH SPEED EXTERNAL CLOCK SIGNAL)*

In questo caso la sorgente di clock non è integrata all'interno del microcontrollore, ma proviene dall'esterno tramite il collegamento al microcontrollore di oscillatori al quarzo.

Questo tipo di oscillatori funzionano sfruttando una particolare proprietà di alcuni materiali, la piezoelettricità, che permette loro di generare una differenza di potenziale se soggetti a deformazioni meccaniche, e allo stesso tempo di subire deformazioni meccaniche se sottoposti a differenze di potenziale. Si può dimostrare che un oscillatore al quarzo è descrivibile utilizzando un semplice circuito RLC, in cui i parametri di resistenza, capacità e induttanza sono ricavabili dalle caratteristiche costruttive del cristallo. Ricordando la formulazione del fattore di merito $Q = \frac{\text{pulsazione di risonanza}}{\text{larghezza di banda}}$, e sapendo che questi sistemi hanno fattori di merito molto elevati, nell'ordine di 100.000, si comprende il motivo per il quale essi vengono utilizzati per la generazione del segnale di clock.

Un altro vantaggio di questa tipologia di materiali risiede nella ridotta dipendenza dalla temperatura del loro comportamento, che rappresenta un vantaggio rispetto all'implementazione tramite oscillatore RC analizzato nel precedente paragrafo.

#### *4.1.3 PLL (PHASE-LOCKED LOOP)*

La terza e ultima sorgente potenziale del segnale di clock del sistema è l'anello ad aggancio di fase integrato, che permette di ottenere in output la massima frequenza di clock disponibile per il sistema pari a 72 MHz.

### *4.2 SORGENTI A BASSE PRESTAZIONI (LSE, LSI)*

La distinzione tra sorgente interna ed esterna è analoga a quella presentata precedentemente per le sorgenti ad alta velocità: anche in questo caso è presente un oscillatore RC interno con una frequenza di clock tipica di 40 KHz (LSI) ed è possibile connettere un oscillatore al quarzo con una frequenza di 32 KHz nel caso in cui invece si voglia utilizzare una sorgente esterna più precisa (LSE).

| Symbol | Parameter | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|
| $f_{LSI}$ | Frequency | 30 | 40 | 50 | kHz |
| $t_{su(LSI)}^{(2)}$ | LSI oscillator startup time | - | - | 85 | µs |
| $I_{DD(LSI)}^{(2)}$ | LSI oscillator power consumption | - | 0.75 | 1.2 | µA |

Figura 17: prestazioni di LSI

Confrontando le due tabelle di Figura 17 e Figura 16 è immediato notare la differenza di due ordini di grandezza sul consumo di potenza, che rende adatte LSI e LSE a fungere da sorgenti di clock nelle modalità di risparmio energetico.

# *5 PORTE DI I/O*

## *5.1 FUNZIONI GENERALI E ORGANIZZAZIONE CIRCUITALE*

Le porte di I/O rappresentano il mezzo tramite il quale i microcontrollori sono in grado di interfacciarsi con il mondo esterno; la famiglia di microcontrollori STM32F334 può arrivare, nei package equipaggiati con 64 pin, ad un totale di 51 porte di I/O, tutte configurabili via software in base alle necessità.

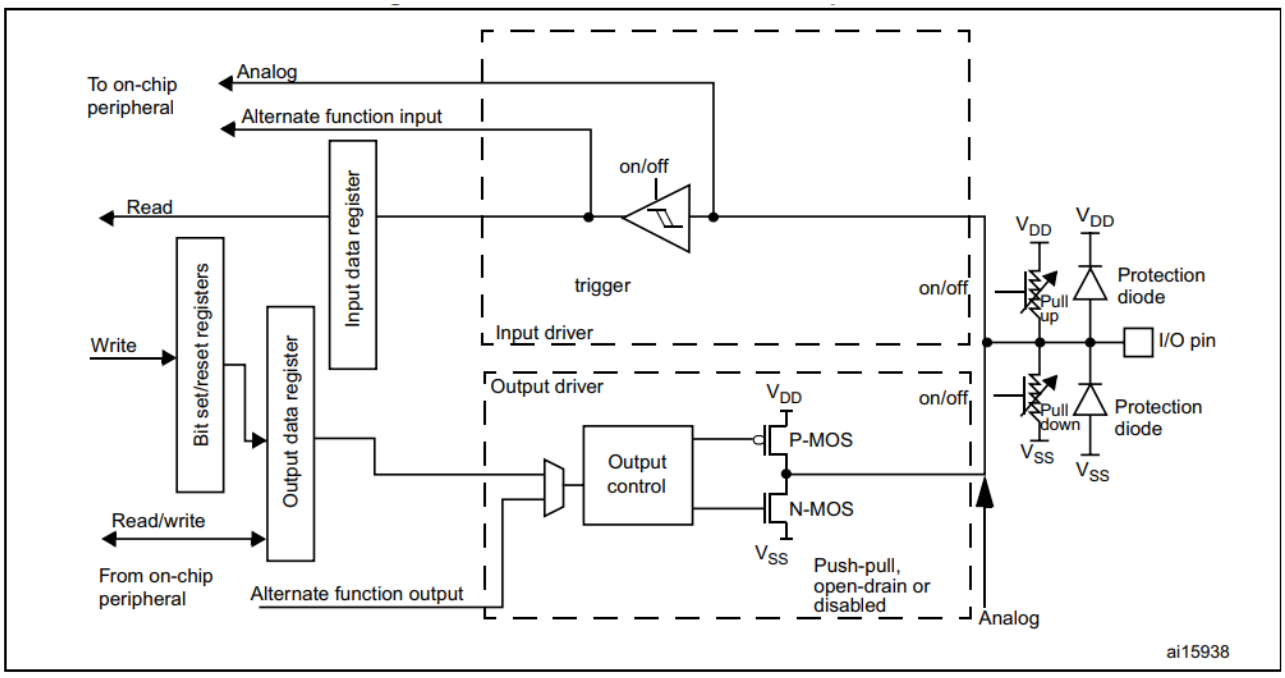


Figura 18: organizzazione circuitale generale di una porta di I/O. [3]

Ogni porta di I/O è composta da una organizzazione circuitale abbastanza complessa come mostrato in figura, che può essere attiva solamente in alcune delle sue parti al variare della modalità impostata via software. L'intero circuito fa capo a due registri mappati in memoria da 16 bit ciascuno, Input Data Register e Output Data Register, che hanno la funzione di salvare il dato corrente in input nel caso del primo e da inviare in output nel caso del secondo.

Le modalità con le quali queste porte possono essere configurate sono:

- Input
- Output
- Analog
- Alternate Function

Per ognuna di queste modalità vi sono poi ulteriori alternative tra cui scegliere che verranno analizzate nel dettaglio nei prossimi paragrafi insieme alla loro corrispettiva struttura hardware.

### *5.2 DIGITAL INPUT*

In modalità input tutta la parte relativa alla circuiteria di Output risulta disattivata come si vede in figura:

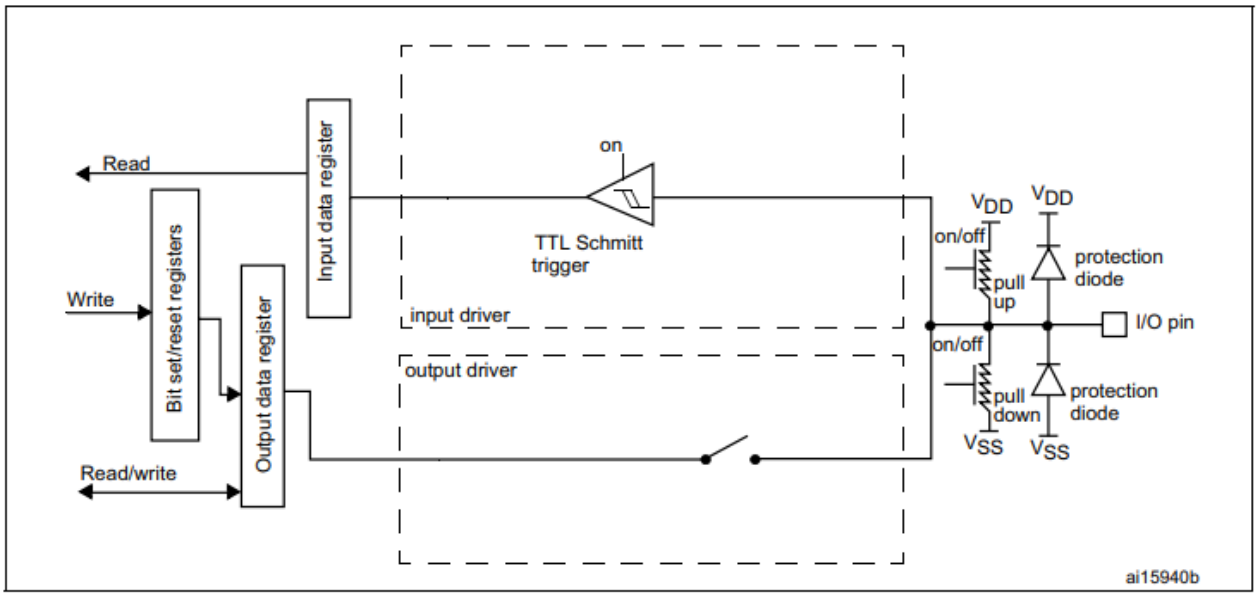


Figura 19: organizzazione circuitale dei pin di I/O in modalità input. [3]

I diodi presenti subito dopo l'I/O pin hanno lo scopo di proteggere il microcontrollore dalle scariche elettrostatiche; questi eventi si possono verificare in seguito al contatto tra due oggetti carichi che causano forte movimento di elettroni e quindi la generazione di correnti elevate che, se non gestite correttamente, possono provocare seri danneggiamenti ai componenti elettronici. I due diodi di protezione hanno quindi lo scopo di scaricare verso l'alimentazione queste correnti nel momento in cui una sovratensione dovuta a questo fenomeno si presenti in ingresso, salvaguardando così il resto del circuito.

Tramite software è possibile scegliere se utilizzare il pin in modalità input pull-op o input pull-down, e ciò va quindi ad attivare una delle due rispettive resistenze; queste hanno la funzione di mantenere il pin ad uno stato noto (VSS per la resistenza di pull-down e VDD per quella di pull-up), per evitare che il rumore elettrico o eventuali accoppiamenti capacitivi portino il pin in uno stato non desiderato.

Infine, il trigger di Schmitt, essendo un circuito bistabile, elimina eventuali fluttuazioni del segnale di ingresso e permette perciò di ottenere alla sua uscita uno stato alto o basso ben definito, che verrà salvato all'interno dell'Input Data Register e trasferito sul bus AHB ad ogni suo ciclo di clock

Alternativamente, il pin può essere utilizzato anche in floating mode con entrambe le resistenze di pull-up e pull-down disconnesse dal circuito; scegliere di utilizzare questa modalità senza predisporre delle resistenze esterne di pull-up o pull-down, però, non è una scelta oculata in quanto tutte possibili variazioni dovute al rumore potrebbero causare una continua attivazione del trigger di Schmitt incrementando il contributo della potenza dinamica assorbita dal sistema.

### *5.3 DIGITAL OUTPUT*

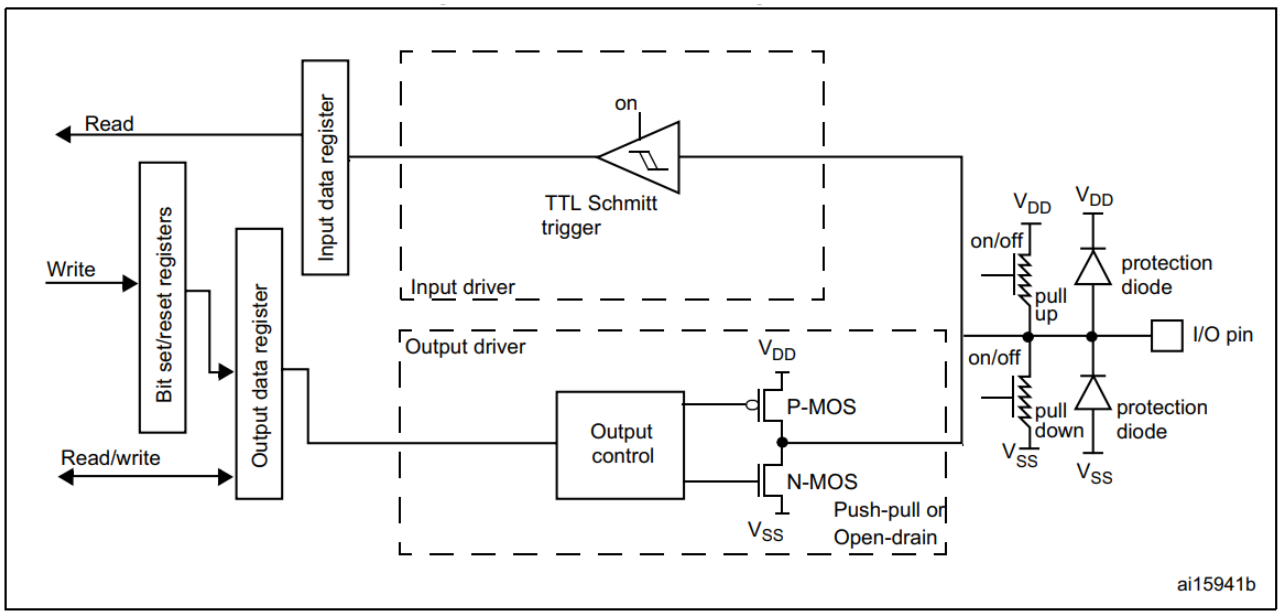


Figura 20: organizzazione circuitale dei pin di I/O in modalità output. [3]

In questa modalità la circuiteria relativa all'output è attiva, come mostrato in Figura 20. Ovviamente in questo caso il segnale procederà in direzione inversa rispetto a quanto visto per la modalità di input, partendo dal salvataggio dello stato all'interno dell'Output Data Register, per arrivare in uscita al pin del microcontrollore.

Anche per la modalità output è possibile impostare, tramite software, due modalità di funzionamento diverse: output push-pull e output open-drain.

La prima utilizza entrambi i MOSFET per il suo corretto funzionamento. Il blocco di Output Control riceve in ingresso il valore salvato all'interno dell'Output Data Register e in base ad esso attiverà uno dei due transistor: in caso il valore salvato sia 0, viene attivato l'N-MOS forzando l'uscita ad un segnale basso, mentre nel caso in cui sia 1, viene attivato il P-MOS, forzandola ad un valore alto.

La modalità open-drain, invece, fa uso solamente dell'N-MOS che funziona analogamente a quanto descritto precedentemente nel caso in cui il valore da portare in uscita sia basso.

Se invece il segnale che deve essere trasportato al pin di output corrisponde ad un livello logico alto, si rende necessario l'utilizzo di una resistenza di pull-up (interna o esterna), in quanto il P-MOS è disconnesso dal circuito. Questa modalità di funzionamento è utile nel momento in cui si voglia pilotare un carico avente una tensione di alimentazione diversa da quella del microcontrollore.

### *5.4 ANALOG*

La modalità analog si differenzia dalle altre due esaminate finora per due motivi:

- Il pin può fungere sia da input che da output
- Il segnale di interesse non è uno stato logico, ma un valore analogico

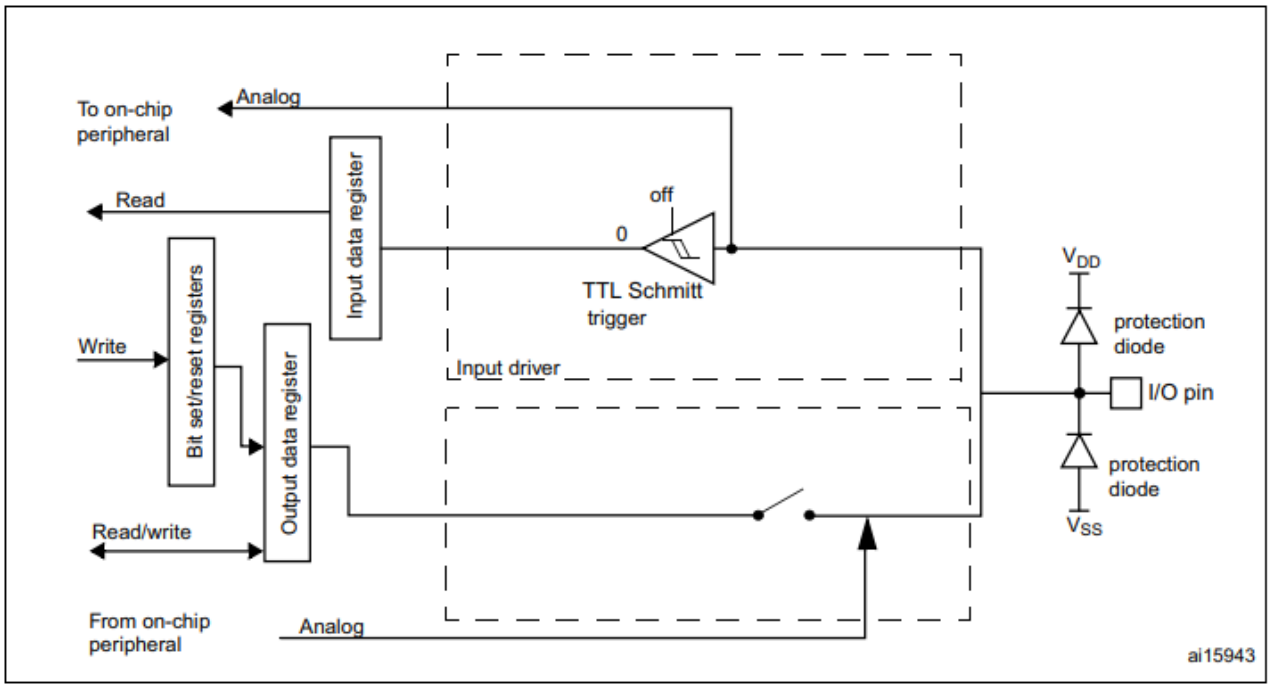


Figura 21: organizzazione circuitale dei pin di I/O in modalità analog.[3]

La configurazione in questo caso è abbastanza semplice: tutta la parte di circuito relativa alla gestione dell'output è scollegata ed il segnale analogico proveniente dal microcontrollore, o più correttamente dal DAC, viene portato direttamente in uscita.

Per quanto riguarda la gestione di un segnale analogico in input, invece, si vede come esso sia prelevato e portato in ingresso al microcontrollore prima del suo passaggio attraverso il trigger di Schmitt, la cui uscita viene mantenuta fissa ad un valore basso, annullando il suo contributo al consumo dinamico di potenza del sistema.

### *5.5 ALTERNATE FUNCTION*

Le modalità di funzionamento analizzate servono a svolgere le classiche funzioni di interfaccia di I/O tra il microcontrollore e l'esterno, come è lecito aspettarsi da pin di questo tipo; la tipologia di funzionamento analizzata in questo paragrafo, invece, si discosta da queste semplici funzioni di base, e mira allo svolgimento di funzioni più avanzate.

La modalità alternate function serve per permettere a periferiche quali timers e interfacce di comunicazione di comunicare con l'esterno; alcuni pin del microcontrollore sono infatti dotati di un multiplexer che li permette di essere collegati con un massimo di 16 linee diverse provenienti da tali periferiche, con uno schema che varia in base al package scelto per il microcontrollore in uso.

Tra le possibili funzioni che questi pin possono svolgere si trovano:

- Canali di input per timers in modalità input capture
- Linee MISO, MOSI, SCK, SS per la comunicazione SPI
- Linee SDA e SCL per la comunicazione I2C

Per quanto riguarda l'organizzazione circuitale, in questo caso nessuna parte del circuito è scollegata per cui si può fare riferimento a quanto rappresentato in Figura 18.

# *6 TIMERS*

## *6.1 PANORAMICA GENERALE*

I microcontrollori della famiglia STM32F334 integrano al loro interno diversi timers, basati sulla stessa architettura scalabile, ed in grado di svolgere diverse specifiche funzioni, tra cui figurano le classiche modalità di input compare e generazione di segnali PWM, ma anche funzioni più avanzate orientate al controllo di motori elettrici e alla conversione di potenza.

Di seguito viene presentata una lista dei timer presenti e una breve descrizione delle loro specifiche:

- HRTIM: questo timer si discosta dagli altri per il suo livello di complessità, in quanto possiede ben dieci canali indipendenti ed ha una frequenza di conteggio nell'ordine dei GHz, ottenibile grazie ad un moltiplicatore di frequenza. Questo timer è in grado quindi di generare impulsi PWM di alta qualità ideali per applicazioni di controllo di motori ed alimentazione elettrica.
- TIM1: timer avanzato a 16 bit molto flessibile in grado di implementare diverse modalità di conteggio e studiato per applicazioni di controllo avanzate
- TIM 6/7: i timers più semplici presenti, sono facili da configurare e possono essere utilizzati per generiche applicazioni di temporizzazione
- TIM 2/3/15/16/17: general purpose timers, possono essere utilizzati per implementare le classiche funzioni di capture e compare e differiscono tra di loro per il numero di bit del contatore ed il numero di canali indipendenti che mettono a disposizione.
- RTC: modulo di Real Time Clock utile per applicazioni di temporizzazione in tempo reale in quanto esso è in grado di tener traccia del giorno, dell'ora e del minuto corrente e funge quindi da calendario di sistema.
- 2 watchdog per il reset del dispositivo nel momento in cui venga riscontrato un problema.

## *6.2 TIM 16/17 GENERAL PURPOSE TIMERS*

Il seguente paragrafo si concentrerà sull'analisi della struttura e del funzionamento dei due più semplici general purpose timers presenti all'interno della famiglia di microcontrollori STM32F334. Essi dispongono di un solo canale e possono essere utilizzati per applicazioni di input capture, output compare, generazione di segnale PWM e one-pulse mode output.

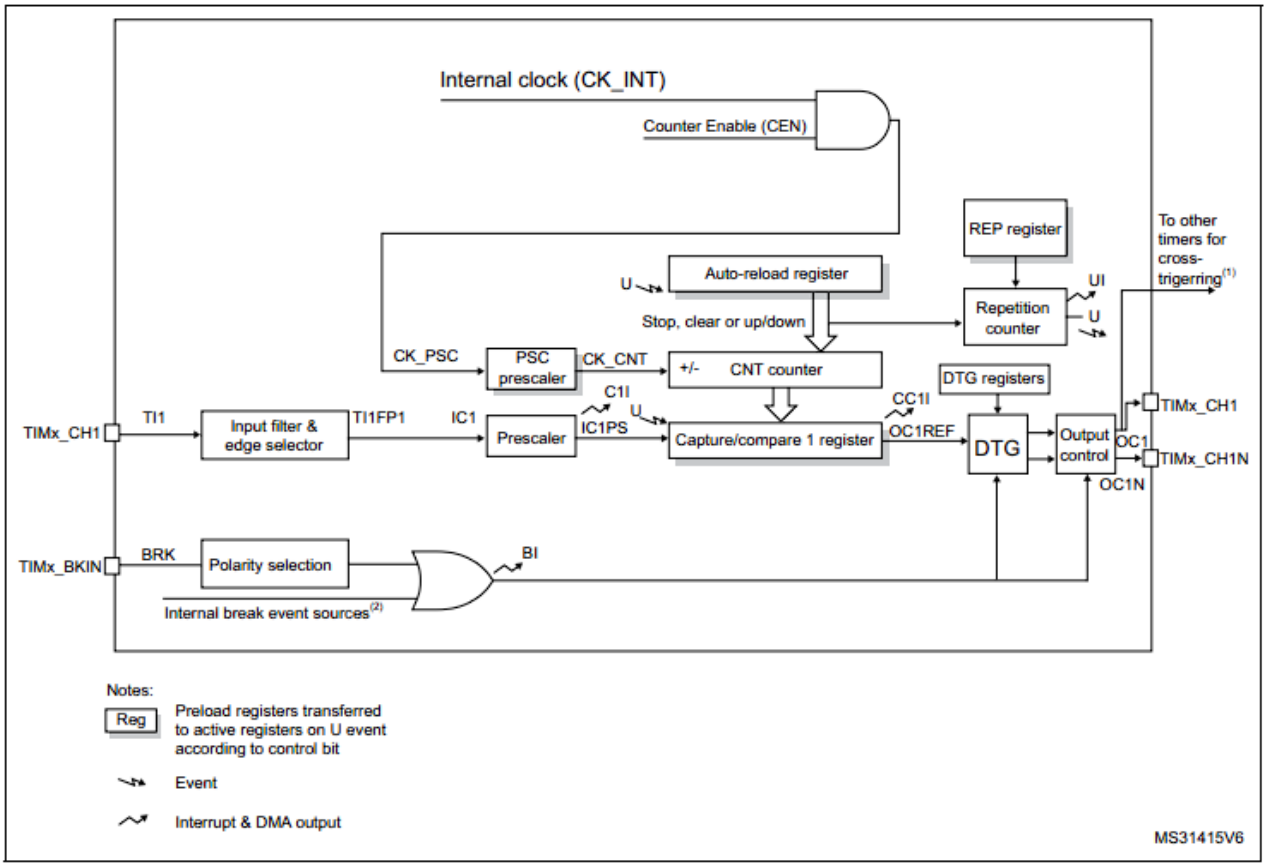


Figura 22: organizzazione circuitale dei TIM 16/17. [3]

Il blocco fondamentale di questi sistemi è rappresentato dal contatore a 16 bit (CNT counter in Figura 22), avente frequenza di conteggio programmabile tramite l'opportuno registro di prescale che permette di dividere la frequenza di clock (CK_INT) per un opportuno fattore compreso tra 1 e 65535, in base alle scelte del programmatore.

### *6.2.1 FUNZIONAMENTO DELL'OPERAZIONE DI PRESCALE*

Un esempio aiuta a chiarire quali segnali intervengono e come avviene effettuata la divisione della frequenza di clock all'interno di questi timers.

Innanzitutto, dalla Figura 23 a pagina seguente, è possibile notare come il segnale CEN funga da segnale di enable per il conteggio, che comincia solamente nel momento in cui esso viene portato alto; questo comportamento è coerente con l'organizzazione circuitale in cui CEN è posto in AND logico con il segnale di clock che permette al contatore di funzionare. Si vede inoltre come in seguito all'aggiornamento del valore salvato all'interno del prescaler control register, e contemporaneamente all'arrivo del segnale di Update Event, la frequenza di conteggio viene modificata e procede ad un quarto di quella precedente.

Questo è reso possibile grazie all'ulteriore contatore a 16 bit presente all'interno del blocco di prescale, il cui valore di saturazione coincide proprio con quello salvato all'interno di Prescale Control Register, che quindi funge da contatore ausiliario per permettere l'operazione di divisione di frequenza.

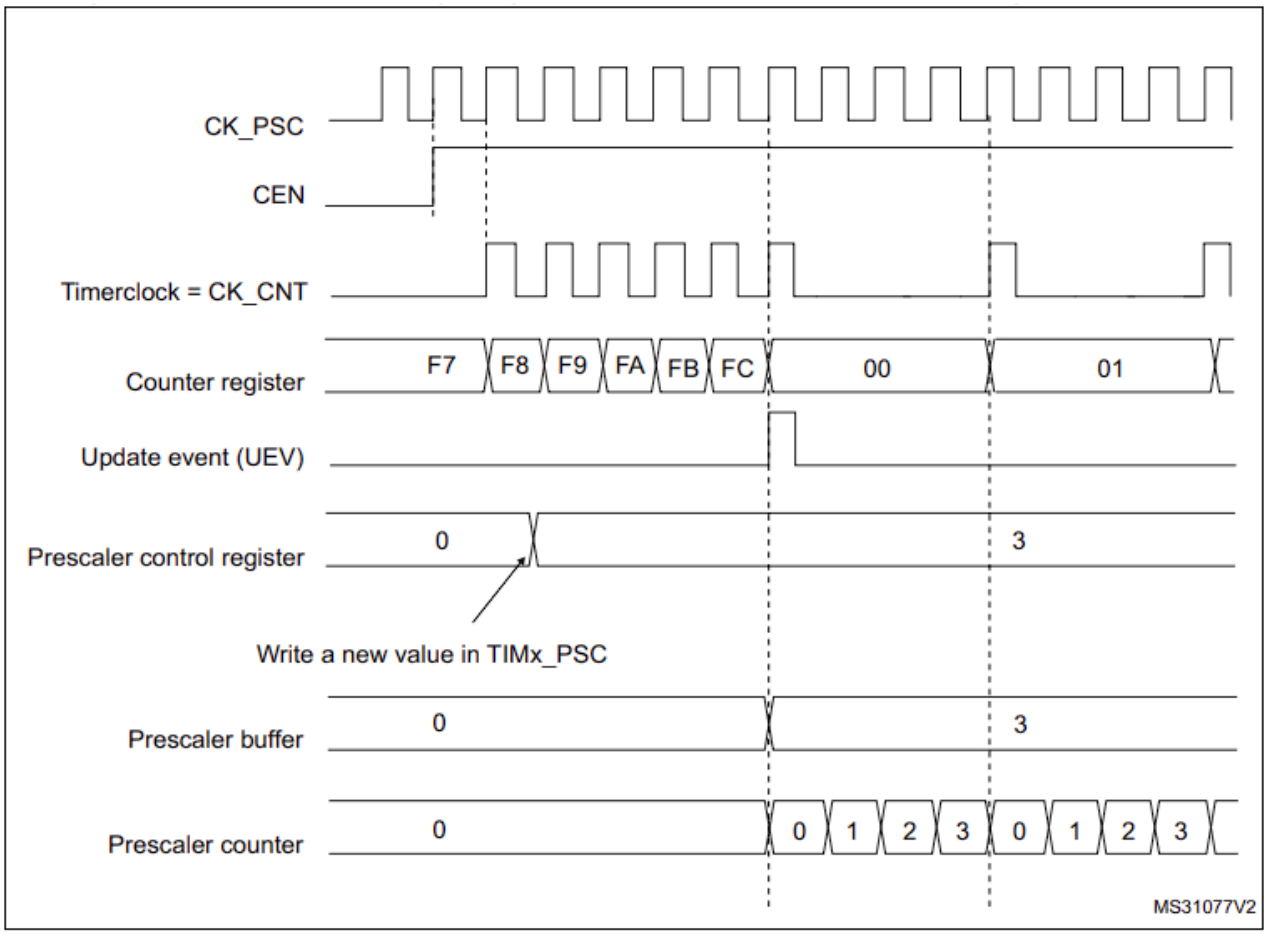


Figura 23: diagramma temporale del passaggio della divisione di frequenza da un fattore 1 ad un fattore 4. [3]

### *6.2.2 AUTO-RELOAD REGISTER E REPETITION COUNTER*

Il blocco di auto-reload register permette di impostare la condizione di saturazione del contatore, per la quale il conteggio viene azzerato e fatto ripartire dal valore iniziale. Questo blocco si compone in realtà di due registri separati, per permettere che l'effettivo aggiornamento della condizione avvenga solamente in conseguenza all'attivazione del segnale di Update Event, il quale può essere inoltre utilizzato per la generazione di un interrupt o di una richiesta per il DMAC. Il registro d'appoggio (preload) è quindi modificabile via software, mentre il registro attivo (shadow register) viene effettivamente integrato con le altre parti del sistema e l'aggiornamento del suo valore avviene solamente nel momento in cui UEV venga portato alto.

Supponendo di utilizzare il timer in upcounting mode, in cui il valore del conteggio viene azzerato al raggiungimento della condizione di saturazione, vi sono tre possibili modalità tramite le quali il segnale UEV viene generato:

- Al raggiungimento della condizione di saturazione da parte del contatore
- UEV è disabilitato via software per cui qualsiasi modifica ai registri di prescale o di auto-reaload non ha effetto
- Al raggiungimento del numero di saturazioni impostate all'interno del Repetition Counter.

Quest'ultima modalità è resa possibile dal blocco di Repetition Counter, contatore addizionale che permette al segnale UEV di venire generato solamente dopo l'avvenimento di N condizioni di overflow del conteggio, con N impostato via software.

### *6.2.3 BLOCCHI DI INPUT E OUPUT*

Procedendo nell'analisi dello schematico di Figura 22, nella parte di sinistra si riconosce l'implementazione del blocco di input, utile per permettere al timer di collegarsi con uno dei GPIO del microcontrollore e realizzare la modalità di input capture. All'interno di questa sezione sono presenti un edge detector programmabile per permettere il riconoscimento di fronti di salita o di discesa (o entrambi) del segnale in ingresso e un filtro che permette di eliminare eventuali transizioni non volute causate dal rumore.

Successivamente, un prescaler permette di realizzare il capture dell'evento ogni 2, 4 o 8 transizioni del segnale, per ridurre il carico di lavoro del processore nel caso in cui si stia lavorando con segnali ad alta frequenza. Infine, il registro di capture permette di salvare il valore attuale del conteggio al sopraggiungimento dell'evento considerato ed è quindi possibile generare un interrupt o effettuare la misura di intervalli di tempo di eventi sul pin considerato.

A destra del registro di capture/compare si trova il Dead Time Generator, utile alla gestione automatica del tempo morto nelle applicazioni di controllo PWM; senza di esso, infatti, si rischia il cortocircuito nel momento in cui il transistor di accensione e quello di spegnimento vengano a trovarsi entrambi accessi a causa della non istantaneità della transizione ON/OFF.

Infine, il rimanente della circuiteria compone l'emergency stop, che blocca il timer ed eventuali segnali PWM in uscita nel caso in cui si verifichi un malfunzionamento all'interno del sistema (ad esempio un clock failure), o un evento sull'apposito pin BKIN. Questa funzionalità può essere utile, ad esempio, nel caso di un controllo di un motore, dove l'arresto potrebbe avvenire in seguito ad un surriscaldamento eccessivo o ad una sovratensione rilevati tramite il pin BKIN.

### *6.3 LA SCALABILITÀ DELL'ARCHITETTURA*

L'organizzazione circuitale presentata nel paragrafo precedente, sebbene sia riferita a solamente due dei timer presenti all'interno del microcontrollore, risulta comunque utile per la comprensione anche dei restanti timer general purpose presenti. Considerando ad esempio la struttura dei basic timer TIM 6 e TIM 7, si nota come il blocco fondamentale di contatore, prescaler e auto-reload register siano analoghi a quelli presentati nel paragrafo precedente.

Ovviamente tali timers possono essere utilizzati solamente semplici applicazioni di temporizzazione software, in quanto non avendo i canali di input e output non possono implementare modalità più complesse come la generazione di segnali PWM o l'input capture.

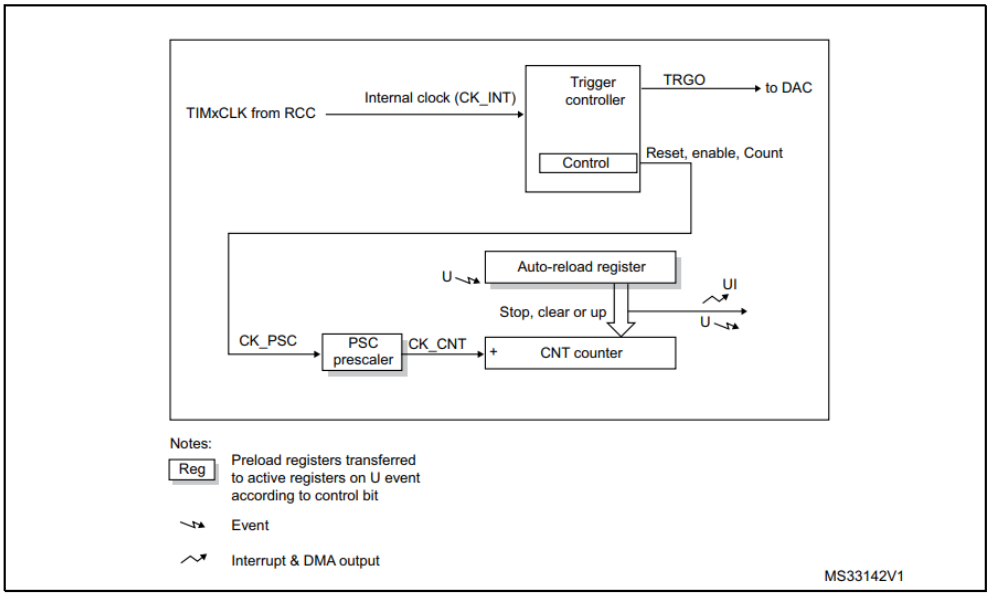


Figura 24: organizzazione circuitale dei timers TIM 6/7. [3]

Un discorso analogo vale anche per timers più complessi di quelli presentati, come TIM 15 ad esempio, in cui l'incremento della complessità architetturale è dettata dal maggior numero di canali indipendenti, ma i blocchi circuitali descritti e le loro funzioni rimangono le stesse.

Rimangono invece esclusi da queste considerazioni HRTIM, RTC e i due watchdog in quanto la loro struttura è appositamente studiata ed ingegnerizzata per le specifiche funzioni che essi devono svolgere.

### *6.4 RTC PER LA GESTIONE DEI WAKE-UP INTERRUPTS*

Il real time clock timer integrato all'interno dei microcontrollori STM32F334 fornisce una indicazione precisa del calendario attuale ed ha perciò una organizzazione strutturale diversa da quanto discusso precedentemente. Il presente paragrafo non ha tanto l'obiettivo di descrivere in che modo le funzionalità di calendario siano implementate a livello circuitale, ma bensì la discussione si concentrerà su un'altra funzionalità importante svolta da questo modulo, ovvero la possibilità di generare wake-up interrupts.

Rimandando allo schematico di figura 15, si vede come sia possibile scegliere tra HSE, LSE e LSI come sorgenti di clock per il modulo in esame. Rimandando a quanto discusso nel primo capitolo, si ricorda che le sorgenti di clock Low Speed (LSE e LSI) rimangono attive anche nelle modalità di risparmio energetico, con la conseguenza che in queste modalità anche l'RTC può continuare a funzionare indisturbato. Inoltre, LSE continua a funzionare anche in caso di mancanza di alimentazione principale grazie all'alimentazione di backup; quindi, se questa sorgente viene utilizzata per l'RTC, è possibile renderlo immune a problemi di questo tipo.

Queste caratteristiche rendono RTC il modulo adatto per la generazione di wake-up interrupts che permettano al microcontrollore di uscire dalla modalità di risparmio energetico (Stop Mode o Standby Mode) con una cadenza prefissata dal programmatore. All'interno dell'RTC è presente un down-counter da 17 bit in grado di generare un interrupt una volta raggiunto lo zero; considerando inoltre che tramite opportuni prescaler, è possibile dividere la sua frequenza di conteggio fino a 1Hz, questo si traduce nella possibilità di far uscire il microcontrollore dalla modalità di risparmio energetico al massimo ogni 36 ore.

# *7 DAC*

## *7.1 PANORAMICA GENERALE*

I convertitori digitale-analogico sono un componente essenziale all'interno di un microcontrollore, e grazie alla loro versatilità, possono essere utilizzati in una miriade di applicazioni diverse, tra cui la generazione di segnali musicali in sistemi audio e della tensione di riferimento negli ADC ad approssimazioni successive, o per la generazione di segnali di controllo di vario tipo.

I microcontrollori in esame integrano al loro interno due DAC (Digital to Analog Converters) per la generazione di segnali analogici, con la possibilità di scegliere una risoluzione di 8 oppure 12 bit. Ricordando che la loro alimentazione proviene da $V_{DDA}$, che, come discusso nel primo capitolo, può assumere un valore minimo di 2V, allora la massima risoluzione ottenibile è pari a $\frac{V_{DDA}}{4096} = 0.48$ mV.

I due DAC presenti si differenziano per il numero di canali e per la loro complessità generale:

- DAC1 ha due canali e una struttura leggermente più complessa, con la possibilità di implementare anche alcune funzioni particolari come la generazione di un segnale di rumore
- DAC2 ha un solo canale ed una struttura più semplice.

## *7.2 DAC2*

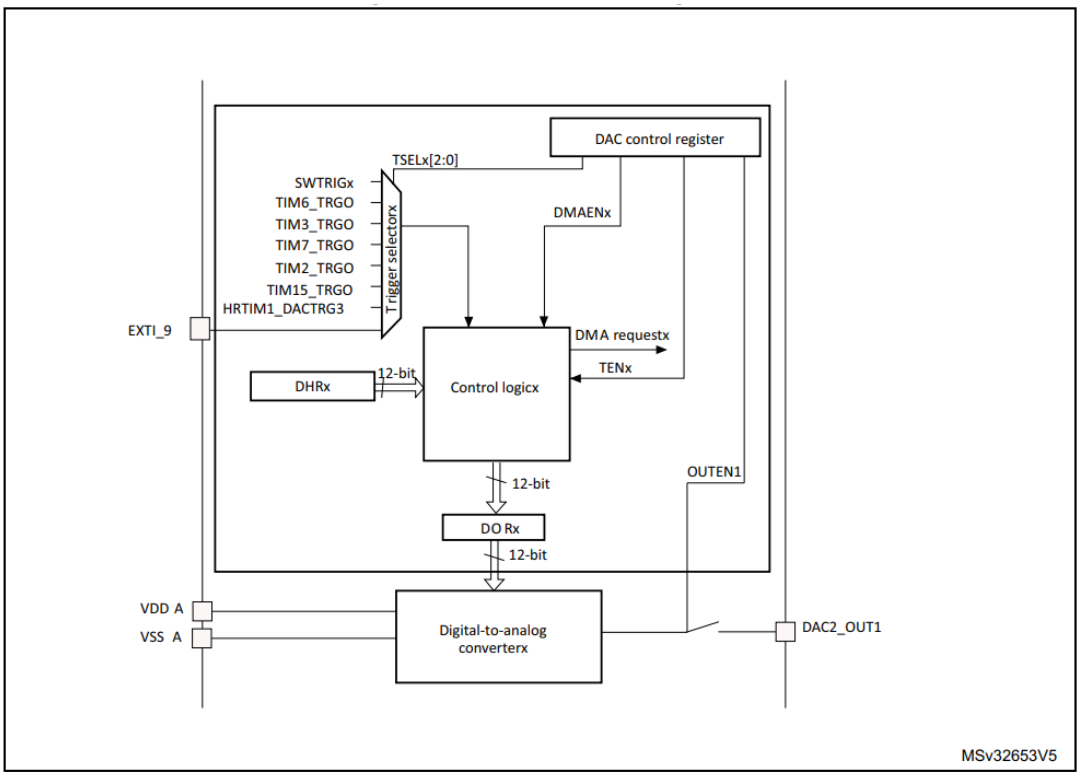


Figura 25: organizzazione circuitale del DAC2. [3]

Nello schematico in Figura 25 è rappresentato DAC2, comprendente il convertitore digitale-analogico vero e proprio e tutta la circuiteria di controllo che lo circonda; si nota innanzitutto come lo switch collegato al pin di uscita venga chiuso da un segnale di enable proveniente dal control register, tramite un opportuno bit dello stesso. Il DAC2 inoltre, a differenza del più complesso DAC1, non presenta alcun buffer sul pin di uscita e potrebbe quindi rendersi necessaria l'aggiunta di un amplificatore operazionale esterno per l'adattamento d'impedenza.

Il registro DHR (Data Holding Register) ha invece lo scopo di memorizzare il dato digitale che verrà poi convertito nel corrispettivo livello analogico, ed esso viene memorizzato in un formato a 8 oppure 12 bit in base alla scelta effettuata via software; nel caso in cui non venga scelta una temporizzazione esterna, questo dato viene poi trasmesso al registro DOR (Data Output Register) ad ogni ciclo di clock del bus APB1 a cui DAC2 è connesso.

Questa trasmissione può essere però alternativamente temporizzata dall'esterno tramite dei timer del microcontrollore: in questo caso, tramite il registro di controllo, viene generato il segnale di selezione per il multiplexer, che provvede a selezionare l'ingresso opportuno tramite cui generare la temporizzazione per il trasferimento del dato al DOR.

La generazione del segnale di output vero e proprio avviene infine dopo un certo tempo di conversione, che dipende dalla tensione di alimentazione e dal carico in uscita, come si vede in Figura 26.

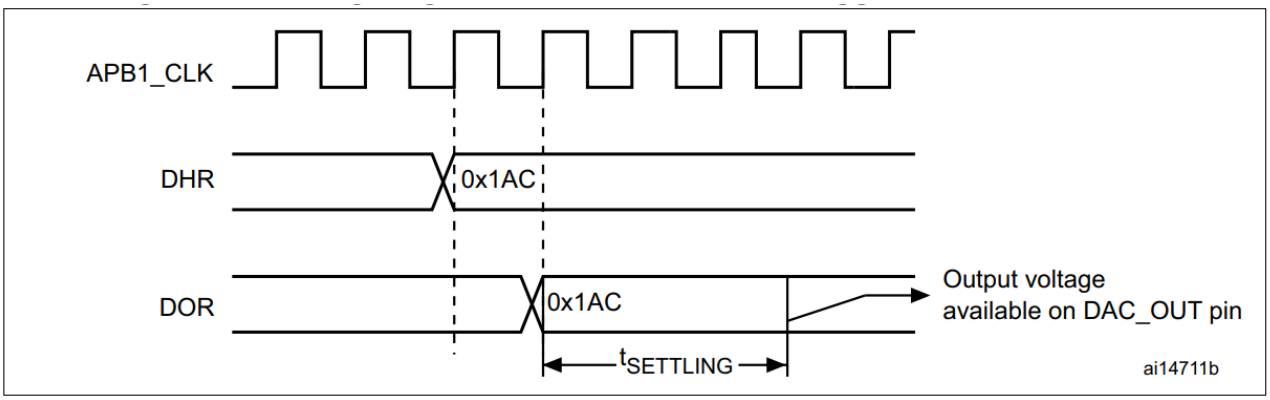


Figura 26: diagramma delle temporizzazioni per il trasferimento e la conversione del dato. [3]

Nel caso preso in esame in figura, la temporizzazione è effettuata direttamente tramite il clock del bus e si vede come il dato è reso effettivamente disponibile al pin di output dopo circa due periodi di clock e mezzo.

Tornando allo schematico di Figura 25, si può infine notare come il control register possa inoltre abilitare la comunicazione del DAC con il DMAC, ed è quindi possibile generare una richiesta di

gestione tramite la linea DMA request all'arrivo del segnale di trigger. In questo modo è quindi possibile trasferire il dato da convertire dalla memoria al registro DHR senza l'intervento della CPU, diminuendone il carico. Nel caso si faccia uso di tale configurazione è però importante notare come sia necessario che fra un dato e l'altro ci sia un intervallo di tempo sufficiente per permettere il trasferimento del primo dalla memoria al registro, in quanto la linea di richiesta di interruzione al DMAC è solamente una e non è possibile implementare delle code. Se questa condizione non viene verificata, viene generato un errore di underrun e si renderà quindi necessaria una diminuzione della frequenza di clock del DAC o una riduzione del carico di lavoro del sistema di DMA.

# *8 CENNI SUGLI ADC*

I due ADC integrati all'interno della famiglia di microcontrollori in esame presentano una struttura circuitale ed una organizzazione interna parecchio complesse, che rendono queste periferiche in grado di operare in una miriade di diverse configurazioni, utili per le più svariate necessità. Il presente capitolo si limiterà a presentare le loro prestazioni e modalità operative, senza scendere maggiormente nel dettaglio per quanto riguarda la loro architettura.

## *8.1 PRESTAZIONI E CARATTERISTICHE*

Come già detto, i microcontrollori STM32FF34 possiedono 2 ADC ad approssimazioni successive, connessi al bus AHB e aventi una risoluzione variabile tra 6,8,10, oppure 12 bit in base alla scelta fatta via software. Il tempo di conversione è compreso tra 0.19 e 0.21 µs per una risoluzione di 12 bit, ma viene ridotto nel momento in cui si decida di utilizzare una risoluzione minore (ad esempio, scegliendo 10 bit di risoluzione, il tempo di conversione è pari a 0.16 µs); questi bassi tempi di conversione permettono di ottenere più di 5 MS/s e quindi ottime prestazioni.

Queste periferiche, inoltre, sono anche altamente modulabili: oltre ad avere diverse modalità operative, hanno anche la possibilità di ricevere in input segnali di tipo single-ended (normalmente riferiti a massa) o di tipo differenziale, in cui la conversione viene eseguita sulla differenza tra due segnali in ingresso.

Infine, essi presentano una funzionalità di auto calibrazione, spesso eseguita all'avvio della periferica, che permette di compensare gli errori di offset e di guadagno e di limitare gli effetti di non idealità legati alla temperatura, il cui valore è misurato da un apposito sensore integrato in questi microcontrollori.

## *8.2 SORGENTI DI CLOCK*

Richiamando il clock tree rappresentato in Figura 15, è possibile notare come il programmatore abbia la libertà di scegliere tra due diverse sorgenti di clock:

- Clock in uscita dal PLL, eventualmente diviso tramite opportuni prescaler.
- Clock del bus APB, che anche in questo caso può essere diviso tramite prescaler.

La scelta di questi due segnali deve ovviamente essere effettuata tenendo a mente i relativi punti di forza delle stesse. In linea generale il segnale uscente dal PLL (se non diviso) permette di ottenere la massima frequenza di clock possibile, pari a 72 MHz, utile nel caso in cui sia necessaria una frequenza di conversione molto elevata. La seconda sorgente presenta invece il vantaggio di essere

maggiormente sincronizzata con il resto del sistema, e quindi permette di evitare problemi relativi alla gestione di clock diversi e permette inoltre una maggiore facilità di configurazione.

### *8.3 MODALITÀ OPERATIVE*

**- *Continuous conversion mode:*** In questa modalità l'ADC acquisisce dati analogici e li converte in modo continuo, fornendo un flusso di dati digitali costante in uscita; l'utilizzo del convertitore analogico digitale in questa modalità può essere utile nel momento in cui sia necessario un monitoraggio costante del segnale analogico in ingresso.

- ***Single conversion mode:*** in questo caso la conversione non è continua, ma viene attivata solamente in seguito all'arrivo di un opportuno segnale di trigger proveniente da un timer; il vantaggio nell'utilizzo del convertitore in questo modo è la possibilità di scegliere una temporizzazione precisa per la generazione dei segnali di trigger, che permette quindi di ottenere facilmente la frequenza di campionamento desiderata. Inoltre, in alcune applicazioni, potrebbe non essere necessaria una conversione continua dei dati, a causa per esempio della scarsa variabilità nel tempo del segnale da misurare; in questo caso la single conversion mode permette di risparmiare consumo di potenza rispetto alla conversione continua, senza dover ricorrere all'utilizzo di modalità di risparmio energetico dell'intero microcontrollore.

# *9 ALTRE PERIFERICHE INTEGRATE ALL'INTERNO DEI MICROCONTROLLORI*

## *9.1 COMPARATORI*

I microcontrollori STM32F334 integrano al loro interno 3 comparatori ad alte prestazioni che, sebbene siano delle periferiche abbastanza semplici, presentano ugualmente una complessità tale da renderli adatti a diverse situazioni.

I tre comparatori sono di tipo rail-to-rail, e permettono quindi di gestire tensioni in tutto il range di valori della tensione di alimentazione analogica (che si ricorda essere compresa tra 2.0 e 3.6 V) senza dare origine a fenomeni di saturazione.

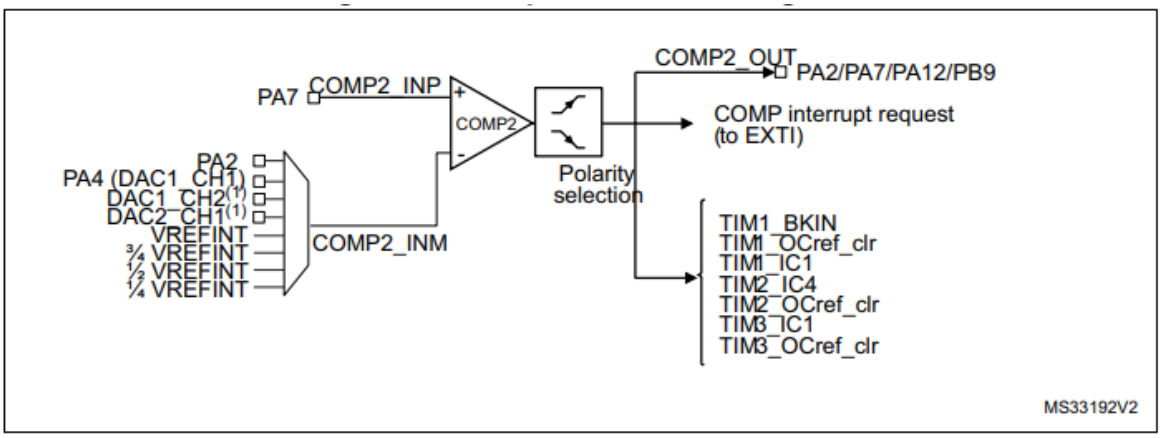


Figura 27: organizzazione circuitale di COMP2. [3]

In Figura 27 è rappresentato lo schematico circuitale di COMP2, uno dei tre comparatori presenti; è possibile vedere come i segnali di input possano provenire da diverse sorgenti, tra cui pin di I/O, uscite dei DAC o alternativamente anche la tensione di riferimento interna o dei suoi sottomultipli. Ovviamente tutti questi segnali sono multiplexati tra di loro e tramite software è possibile decidere quale utilizzare come input per la periferica. Successivamente al comparatore vero e proprio è poi presente il blocco di "polarity selection", che aumenta la flessibilità del dispositivo dando la possibilità di scegliere se utilizzare una polarità normale o invertita. Andando ad analizzare il blocco di output, invece, si vede come sia possibile collegare il segnale in uscita dal comparatore alternativamente ad uno dei pin digitali o ad alcuni dei timer integrati, in entrata ad esempio al blocco di emergency stop input.

Ovviamente gli ingressi di input e output ed i collegamenti con le altre periferiche non sono gli stessi per tutti e tre i comparatori: ognuno di essi è infatti internamente connesso a timer o a pin digitali diversi per fare in modo che un largo numero di periferiche li possa sfruttare.

Infine, è utile notare come i comparatori siano in grado di generare un interrupt in corrispondenza di un loro cambio di stato. Questa loro caratteristica li rende adatti anche alla generazione di wake-up interrupts per l'uscita dalle modalità di risparmio energetico; in alcune applicazioni potrebbe infatti non essere necessario mantenere il microcontrollore in normal-mode indefinitamente, ma potrebbe tornare utile la possibilità di ripristinare tale modalità solamente al raggiungimento di una certa soglia da parte di uno dei pin di I/O.

## *9.2 AMPLIFICATORE OPERAZIONALE*

### *9.2.1 CARATTERISTICHE CIRCUITALI*

Oltre ai comparatori analizzati nel precedente paragrafo, i microcontrollori in esame integrano anche un amplificatore operazionale configurabile, come mostrato in Figura 28.
L'implementazione di tale periferica è una scelta oculata data l'ampia gamma di utilizzo di questi dispositivi, e la sua disponibilità integrata (nel caso in cui non si stiano ricercando prestazioni o caratteristiche particolari) permette di ottenere un risparmio in termini di spazio e costi.

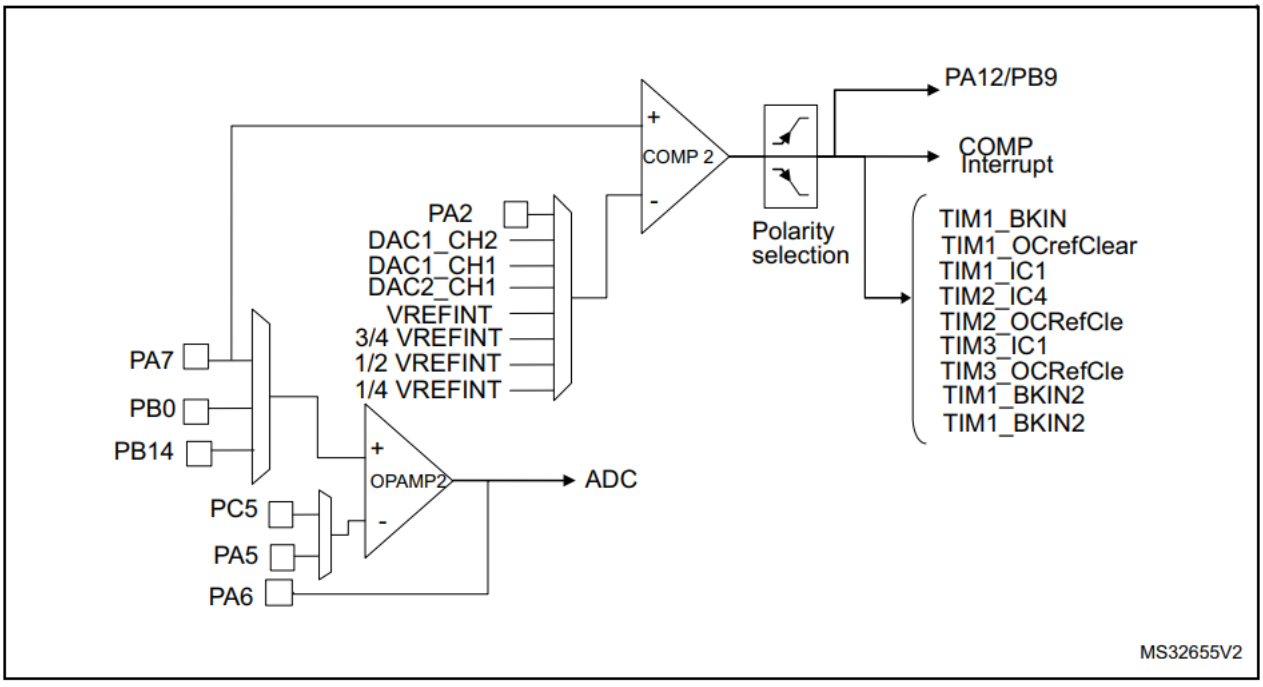


Figura 28: organizzazione circuitale di OPAMP2.[3]

Anche in questo caso la periferica è di tipo rail-to-rail ed inoltre presenta una bassa tensione di offset in uscita, il che migliora la precisione generale dei segnali forniti in uscita.

Come si vede in Figura, i segnali di ingresso ai morsetti invertente e non invertente provengono da alcuni dei pin di I/O, e uno tra questi è anche condiviso con il comparatore per specifiche applicazioni di controllo di motori; in uscita invece il segnale può essere utilizzato come input per

l'ADC per successive elaborazioni digitali oppure può essere direttamente connesso ad un pin di I/O. I vantaggi nel suo utilizzo come stadio precedente all'ADC sono evidenti, in quanto esso fornisce la possibilità di implementare semplici blocchi di condizionamento del segnale

### *9.2.2 MODALITÀ OPERATIVE*

L'amplificatore operazionale in questione può essere utilizzato come dispositivo standalone o alternativamente come amplificatore a guadagno programmabile, nel seguito vengono descritte queste due modalità di funzionamento.

#### *OPAMP COME DISPOSITIVO STANDALONE*

In questa modalità, attivabile mediante la scrittura di alcuni bit nel Control Register della periferica, la configurazione circuitale ed il guadagno vengono definiti dalla scelta delle connessioni e dei componenti esterni che il programmatore decide di utilizzare, permettendo quindi la massima flessibilità a discapito di una realizzazione che deve essere effettuata esternamente.

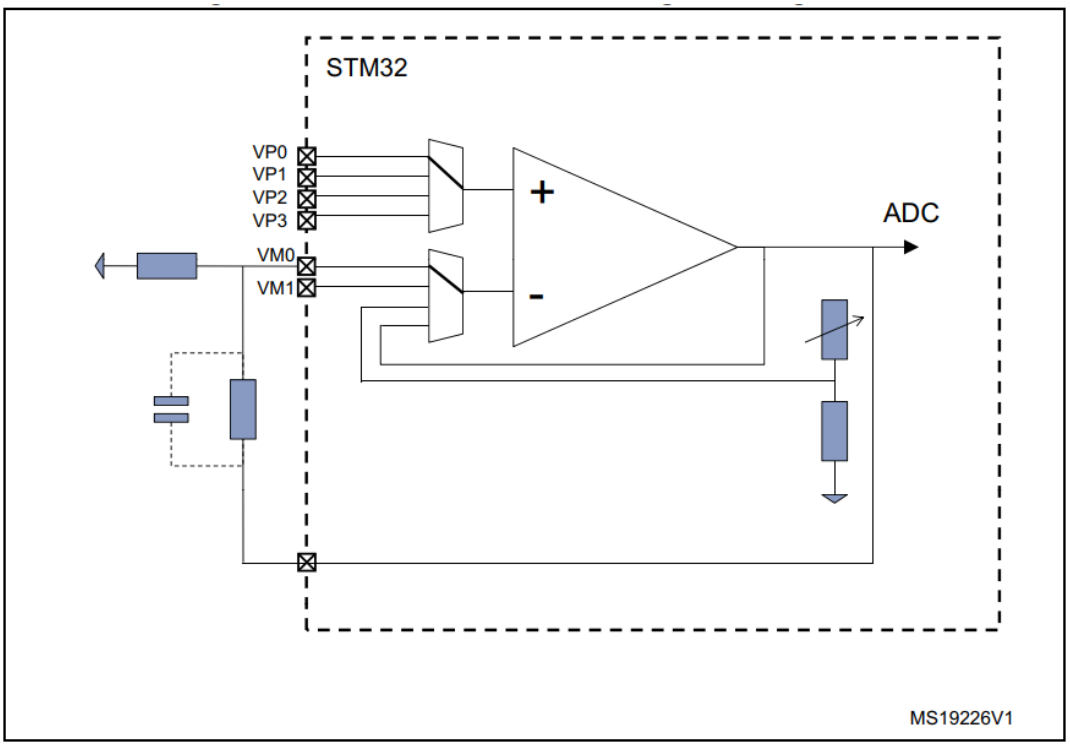


Figura 29: esempio di configurazione dell'OPAMP in modalità standalone.[3]

In figura 29 è rappresentato un esempio di una realizzazione di una configurazione invertente tramite l'utilizzo di due bipoli esterni connessi al morsetto invertente, ed opportunamente selezionati come ingressi da un multiplexer. Ovviamente la libertà di configurazione in questo caso è totale, con la possibilità di implementare anche configurazioni più complesse (sommatori, integratori, ecc).

### *OPAMP COME INSEGUITORE*

In questa modalità l'OPAMP è connesso a buffer e nessun segnale proveniente dall'esterno è connesso al morsetto invertente.

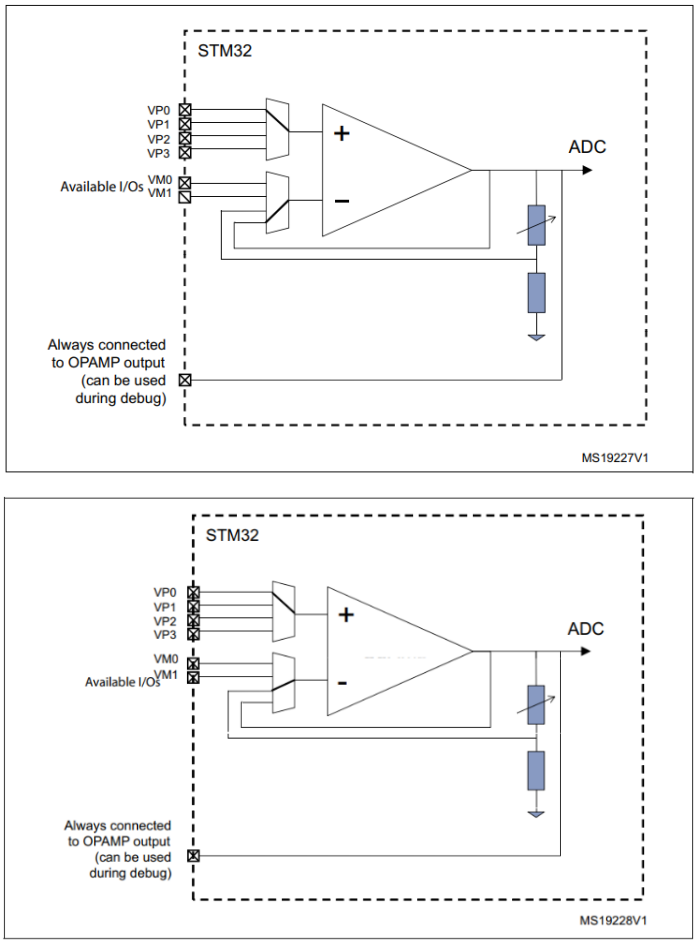


Figura 30: configurazione di OPAMP come inseguitore e come amplificatore a guadagno programmabile. [3]

L'OPAMP utilizzato in questo modo replica in uscita il segnale del morsetto non invertente, in quanto non è presente nessun elemento nella rete di retroazione; la periferica in tale configurazione permette quindi di adattare le impedenze dei diversi stadi che si vogliono realizzare.

### *OPAMP COME AMPLIFICATORE A GUADAGNO PROGRAMMABILE*

In questa modalità il guadagno è selezionabile via software tramite la scrittura di un opportuno registro ed è possibile scegliere un valore compreso tra 2 e 16. In base al valore scelto, verrà modificato il valore della resistenza variabile e con essa la rete di retroazione.

## *10 CONCLUSIONI E RINGRAZIAMENTI*

In questo lavoro di tesi sono stati analizzati, in alcune delle loro parti, i microcontrollori della famiglia STM32F334. Oltre alla analisi delle singole parti del sistema e delle loro funzioni, una particolare attenzione è stata posta nel cercare di comprendere e illustrare come esse siano interconnesse tra di loro e i vantaggi di tali interconnessioni.

Questo elaborato mette in luce l'estrema complessità architetturale attualmente raggiunta anche da dispositivi relativamente semplici come i microcontrollori, e la voglia di comprendere più nel dettaglio il loro funzionamento è stato il motore nella stesura di questa tesi.

Vorrei ringraziare il professor Carlo De Santi per la supervisione e per le sue delucidazioni, sempre chiare e precise, e per avermi introdotto all'argomento durante il corso di "Microcontrollori e DSP".

## *BIBLIOGRAFIA*

[1] https://it.wikipedia.org/wiki/Intel_4004

[2] K. R. Raghunathan, "History of Microcontrollers: First 50 Years," in IEEE Micro, vol. 41, no. 6, pp. 97-104, 1 Nov.-Dec. 2021, doi: 10.1109/MM.2021.3114754.

[3] STMicroelectronics. “RM0364 Reference Manual”, 2020. https://www.st.com/en/microcontrollers-microprocessors/stm32f334.html

[4] https://wiki.st.com/stm32mcu/wiki/Getting_started_with_PWR

[5] STMicroelectronics.”STM32F334x4, STM32F334x6, STM32F334x8 Datasheet”, DS9994, 2018. https://www.st.com/en/microcontrollers-microprocessors/stm32f334.html

[6] STMicroelectronics “STM32 microcontroller GPIO hardware settings and low-power consumption”, AN4899, 2022

[7] STMicroelectronics “STM32 Cortex-M4 MCUs and MPUs programming manual”, PM0214, 2020