**ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA**
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA, INFORMATICA
E TELECOMUNICAZIONI

# Controllo ultra-low power di un modulatore back-scattering per applicazioni di localizzazione in sistemi RFID

Elaborato in:
**Elettronica dei Sistemi Digitali**

Relatore:
Prof. **Aldo Romani**

Presentata da:
**Marco Guarino**

Correlatore:
Dott. **Matteo Pizzotti**

Sessione: I
Anno Accademico: 2015/2016

*alla mia Famiglia...*

# Indice generale

# Abstract

In questa tesi si è voluto elaborare un applicativo ULP (Ultra-Low Power) tramite un microcontrollore, per implementare la procedura di controllo dei diversi circuiti di un TAG RFID.

Questo TAG e'stato pensato per lavorare in assenza di batteria, da cui la necessita' di implementare un sistema ULP. Inoltre, la sua attivazione deve essere comandata attraverso un'architettura tipo WuR (Wake up Radio), in cui un segnale di controllo radio attiva il circuito. Al fine di realizzare tale architettura la rete di demodulazione del segnale e' stata collegata direttamente ad un microcontrollore attraverso la sua interfaccia sseriale.

Nel capitolo 1 verrà introdotto il tema dell'Energy Harvesting, verranno inoltre brevemente spiegate le nozioni base inerenti alle conoscenze necessarie per una comprensione più accurata della tesi.

Nel capitolo 2 verrà spiegato il Sistema nel complesso, rappresentato con schemi illustrativi.

Nel capitolo 3 verrà spiegato dettagliatamente il funzionamento del microcontrollore scelto, per avere una panoramica di tutte le periferiche, le modalità configurabili sul micro e per descrivere i loro possibili funzionamenti.

Il capitolo 4 sarà dedicato alla spiegazione di tutto il firmware implementato per svolgere le operazioni fondamentali imputate al micro per la parte di controllo.

Verrà inoltre introdotto il codice Vhdl sviluppato tramite la programmazione di un FPGA della famiglia Cyclone II, per emulare una trasmissione seriale lato WuR verso il microcontrollore.

Nel capitolo 5 verrà presentata una stima dei consumi del micro al variare del numero di attivazioni al secondo del micro stesso, ovvero numero di ricezioni possibili al secondo.

A tal proposito si è poi attuato un confronto con un altro micro, con migliori prestazioni e a basso consumo, che potrebbe rappresentare infine un'alternativa valida di progetto.

Nei capitoli 6 e 7 ritroveremo sviluppi futuri possibili e le conclusioni del progetto.

Specifiche di progetto rilevanti della tesi sono:

1. minimo dispendio energetico del microcontrollore ULP(ultra-low power)
2. elevata rapidità di risposta per la ricezione dei TAGs, per garantire la ricezione di un numero maggiore possibile di indirizzi (almeno 20 indirizzi al secondo), in un range di tempo limitato
3. generazione di un segnale PWM a 100KHz di frequenza di commutazione con duty cycle 50% per controllare la modulazione in back-scattering

# 1. Introduzione

Di seguito verrà presentata una breve descrizione del progetto di tesi realizzato.

In particolare verranno presentate brevemente le tecnologie e sottoblocchi componenti del progetto, che verte nell'implementare la parte logica di un TAG RFID passivo, ovvero il vero e proprio cuore del TAG in questione.

Per la progettazione ci si è basati su schemi presi da un articolo di riferimento[6], in quanto l'argomento trattato è oggetto di studi da parte di Unibo.

Il nodo logico sopracitato si è deciso di implementarlo tramite un microcontrollore Ultra-Low Power, al fine di limitare il consumo di potenza, e una velocità di esecuzione tale da non influire sulla potenza disponibile sul chip, in modo da garantire agli altri blocchi del TAG una sufficiente disponibilità energetica.

Il tutto è stato possibile, grazie all'ausilio di una Wake Up Radio (WuR) collegata al micro, responsabile della ricezione RF del segnale di risveglio del micro, e appunto dell'alimentazione stessa di quest ultimo.

Al giorno d'oggi le Wake Up Radio sono realizzabili tramite svariate architetture differenti tra loro, ma basti sapere che in generale è possibile arrivare alla realizzazione delle stesse tramite la cascata di blocchi discreti a bassissimo consumo energetico, con un'elevata scalabilità e quindi con ingombro limitato.

Inoltre la tensione di alimentazione necessaria può raggiungere livelli di soglia molto bassi, diminuendo ulteriormente la dissipazione di potenza.

I requisiti di progetto che ci si è preposti di rispettare sono stati quindi fortemente caratterizzati dai blocchi che verranno spiegati di seguito nei paragrafi inerenti all'introduzione del progetto.

Il Microcontrollore, interfacciato tramite una Wake Up Radio, viene risvegliato dalla circuiteria di comparazione della WuR, in seguito alla trasmissione di una Portante UHF, che viene appunto demodulata dalla WuR.

Una volta che il Micro si risveglia, comunica serialmente con la WuR ricevendo una trama di 8bit da essa, contenente l'indirizzo univoco del TAG.

Nel caso in cui il microcontrollore riconosca un indirizzo valido, fa commutare un pin ad una frequenza preimpostata, mandando un segnale PWM sul pin in questione, il quale andrà a comandare uno Switch MOS HF, incaricato della gestione del Back-Scattering;

La procedura spiegata sopra, serve ad attivare appunto la modulazione in Back-Scattering, che cambia la polarità dell'impedenza vista dalla sorgente, in modo da riflettere in modalità differenti il segnale ad alta frequenza ricevuto.

## 1.1 Energy Harvesting

Nel corso degli anni, il mercato dell'elettronica e dei Sistemi Embedded si è sempre più evoluto verso lo scaling della componentistica, con conseguente riduzione proporzionale delle tensioni di lavoro, cosa che ha consentito la riduzione dei consumi a parità di capacità computazionale in maniera sempre più evidente.

Altro fattore trainante dell'industria elettronica è la richiesta di risparmiare energia, di progettare sistemi in grado di rispettare l'ambiente, e di riuscire a reinvestire l'energia spesa per ridurre al minimo perdite o sprechi della stessa.

È quindi sempre più strategico il tema dell'Energy Harvesting o Energy Scavenging, ovvero del processo per cui l'energia, proveniente da sorgenti ambientali, viene catturata e salvata.

Le cosiddette forme di energia alternative, sono tutte quelle sorgenti disponibili nell'ambiente;

tale processo si pone il compito di convertirle (non solo quelle alternative) in energia elettrica direttamente riutilizzabile.

Tipicamente gli harvester forniscono un'energia molto piccola, ma l'energia presente nell'ambiente è quella liberamente disponibile come energia di fondo, quindi potenzialmente infinita.

Le sorgenti alternative possono essere fonti di energia, come ad esempio energia termica, energia cinetica, energia chimica, energia potenziale o solare, disperse liberamente nell'ambiente.

### *1.1.1 Modalità di Conversione*

Le conversione dell'energia avviene in modalità differenti a seconda della fonte che si vuole andare a racimolare:

- l'energia spuria proveniente dalle trasmissioni a radiofrequenza o teoricamente da qualsiasi emissione elettromagnetica può essere raccolta. Un tipico utilizzo di questa tecnica è usato per alimentare gli identificatori RFID.
- La conversione del moto meccanico può avvenire tramite cristalli piezoelettrici o particolari polimeri, che sottoposti a sforzi da deformazione meccanica, generano piccoli potenziali elettrici.
- L'energia solare viene agevolmente immagazzinata tramite celle fotovoltaiche per effetto fotovoltaico appunto, che si prestano facilmente per essere scalate in piccole dimensioni, un esempio è quello delle calcolatrici tascabili, oppure quello delle lampade segnapasso da giardino,...
- In presenza di gradienti termici si possono utilizzare generatori termoelettrici. Si possono anche ottenere potenziali utili per un uso diretto, semplicemente disponendo in serie numerosi generatori. Le potenze prelevate tramite questo processo sono in genere nell'ordine dei mW e si può anche sfruttare il calore umano e animale.

### *1.1.2 Applicazioni ed interesse commerciale*

I dispositivi in grado di raccogliere e convertire l'energia ambientale in energia elettrica hanno suscitato molto interesse sia nel settore militare che commerciale.

Le applicazioni future potrebbero includere l'alimentazione di dispositivi autonomi ad alta potenza di uscita (o matrici di tali dispositivi), sufficientemente robusti per sopportare l'esposizione a lungo termine in ambienti ostili quali località isolate o di difficile accesso, per servire come affidabili centraline elettriche a servizio di grossi sistemi, oppure dispositivi dal consumo molto basso, come ad esempio, sensori la cui trasmissione dati avviene via radio e quindi privi di una connessione elettrica su cui si possa fare fluire una corrente di alimentazione.

# 1.2 la Tecnologia RFID

RFID è l'acronimo inglese di Radio Frequency Identification.
La tecnologia RFID di Identificazione Automatica è basata sulla propagazione nell'aria di onde elettromagnetiche, e consente la rilevazione automatica (hand free), massiva ed a distanza di oggetti, persone e animali sia statici che in movimento.

Un TAG RFID è genericamente composto da un trasponder, ovvero un ricetrasmettitore in generale composto da una o più antenne, col compito di ricevere segnali RF(o HF/VHF/UHF in base alla tipologia di TAG), che poi arrivano ad un modulatore back-scatter, che cambia la polarità dell'impedenza alla sorgente per permettere la ricezione completa alla sorgente.
I TAGs o trasponders, possono essere classificabili in quattro categorie ( in termini di alimentazione) :

- attivi
- passivi
- semiattivi
- semipassivi

In generale un TAG si dice attivo se possiede "a bordo" un'alimentazione, sia per circuiteria di memoria (perchè possono essere solo read, o anche read/write) che per circuiteria di ricetrasmissione.
Possiedono, a differenza dei passivi, una capacità in termini di potenza maggiore, e raggiungono distanze più elevate, nell'ordine dei 10/15 metri in UHF.

Un'ulteriore classificazione è possibile farla in termini di banda di utilizzo; vi sono infatti diversi TAGs operanti a differenti frequenze, velocità di comunicazione e con diversi gradi di vulnerabilità a rumore, disturbi ed interferenze.

Esistono TAGs operanti in bande differenti, che sono:

- TAGs LF
- TAGs RF
- TAGs HF
- TAGs VHF

# 2. Descrizione del Sistema

Il Sistema che si vuole andare a gestire tramite il blocco WuR + uC è rappresentato nell' "Illustrazione 1", nella quale si è evidenziata la sottoparte di sola competenza di microcontrollore e Wake Up Radio (WuR), raffigurata tramite sistema a blocchi generico, nell'"Illustrazione 2" poco sotto.

La WuR si occuperà in primo luogo di demodulare la portante UHF modulata in OOK o ASK con un segnale che emula il protocollo di comunicazione seriale asincrona UART.

Una volta che il micro riceve un indirizzo valido, invia il segnale PWM verso il modulatore di back-scattering, per farlo commutare e di conseguenza variare la polarità dell'impedenza vista dalla sorgente verso l'antenna.

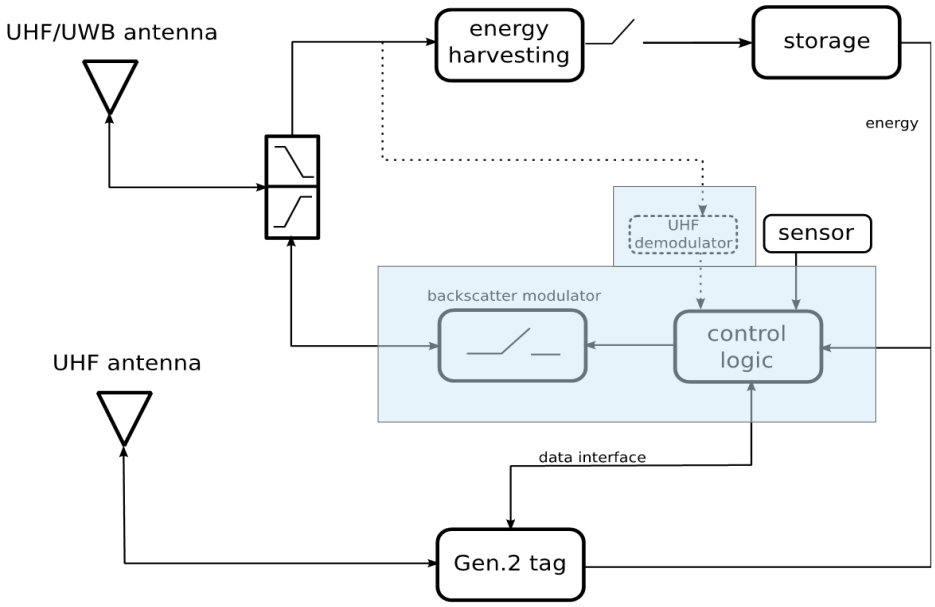


*Illustrazione 1: Schema di un TAG RFID*

Il micro rappresenta appunto nello schema visto sopra in "Illustrazione 1", la Logica di Controllo necessaria per il funzionamento complessivo del TAG [6].

È di fondamentale importanza, in quanto senza di essa non sarebbe possibile realizzare il Sistema pensato.

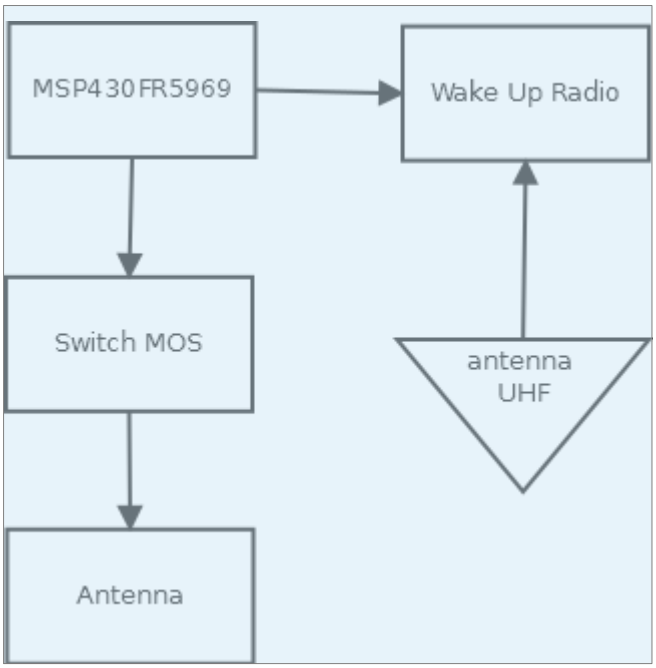

*Illustrazione 2: schema semplificato del TAG RFID per quanto riguarda il funzionamento da imputare al microcontrollore ULP di Controllo*

Nell' "Illustrazione 2" è rappresentato uno schema più semplificato.

I compiti del microcontrollore sono:

- risveglio micro da parte di WuR
- configurazione periferiche
- lettura dell'indirizzo proveniente da WuR, verificando se è un indirizzo valido
- se valido, attivazione segnale PWM in uscita da parte di Timer_A1
- conteggio tramite Timer_B0 della durata del segnale PWM di 3ms
- passati i 3ms dall'attivazione dei Timer il micro deve passare in modalità LPM4(a basso consumo)
- la WuR una volta commutato il backscatter, toglie alimentazione al micro che si spegne

Il processo deve durare una decina di millisecondi da quando il micro viene acceso; questo perchè si vogliono leggere nell'arco di un secondo almeno venti TAGS.

Inoltre è indispensabile che la ricezione UART sia il più robusta possibile, in modo da non rischiare di perdere dati perchè non ricevuti correttamente.

Il segnale PWM dell'onda quadra mostrava requisiti meno stringenti, nel senso che si poteva pensare di abbassare la frequenza e quindi di aumentare il periodo dell'onda quadra generata sul pin per diminuire il dispendio energetico.

Per questo nei capitoli successivi verranno mostrati diversi iter effettuati, per testare la variazione in consumo a seconda dei diversi procedimenti considerati e testati per raggiungere anche questo requisito di progetto.

## 2.1 Wake Up Radio

Ad oggi le applicazioni tramite microcontrollori sono limitate fortemente dall'utilizzo della batteria; il consumo energetico determina in questo caso in maniera diretta la durata dell'applicazione, che determina una scarsa facilità di allocazione dei chip all'interno di luoghi difficilmente raggiungibili, e porta a compromessi aggiuntivi.

Il connubio tra microcontrollore e questo dispositivo che verrà in seguito brevemente spiegato, è una soluzione ottimale ai problemi sopraelencati, in quanto l'utilizzo di una Wake Up Radio per applicativi a microcontrollori ne aumenta drasticamente la portabilità, la durata, e ne abbassa fortemente il consumo energetico quando sono coinvolte comunicazioni radio.

Questo dispositivo sfrutta l'energia spuria delle telecomunicazioni a radiofrequenze, o comunque è in grado di immagazzinare energia propria da radiazioni elettromagnetiche provenienti da comunicazioni radio, in questo caso UHF, consentendo di spegnere i convenzionali ricevitori radio che rapprensentano generalmente una delle fonti di consumo principali.

Una volta che viene trasmessa nell'etere una portante UHF, la Wake Up Radio attraverso l'antenna di ricezione ne incanala la potenza incidente, per poi demodularla tramite un rilevatore ad inviluppo e utilizzarla per risvegliare il microcontrollore spento, già pronto per effettuare le operazioni di sua competenza.

In questo modo viene completamente eliminata la batteria, e il consumo totale del Sistema costituito dai due dispositivi è dato dalla richiesta energetica da parte della Wake Up Radio, e dalla singola richiesta energetica dell'applicazione eseguita dal micro al risveglio.

Ad oggi è possibile realizzare una Wake Up Radio, tramite diverse modalità ed architetture differenti.

In generale è composta da una rete a diodi per la demodulazione OOK della portante ricevuta, un comparatore, blocco che si prende appunto l'incarico di svegliare il microcontrollore (tramite il quale viene completata la ricezione degli indirizzi TAG con l'ausilio del modulo UART) quando viene ricevuta la portante, e una CPU interna che gestisce le operazioni di tipo logico.

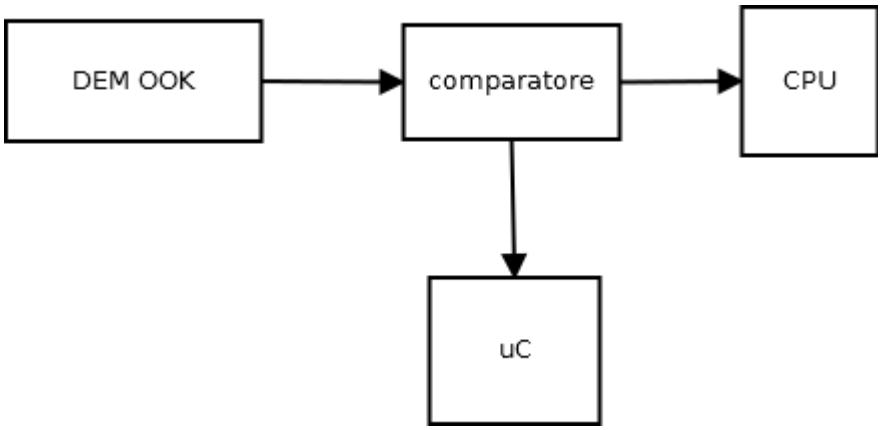


*Illustrazione 3: Schema riassuntivo dei blocchi costitutivi della Wake Up Radio*

# 3. Microcontrollore ULP di riferimento: MSP430FR5969

Il microcontrollore scelto, è un microcontrollore della Texas Instruments, ovvero l'MSP430FR5969.

È un microcontrollore classificato come ULP (Ultra Low Power), che dispone di diverse modalità di utilizzo, dalla più dispendiosa modalità di Run (unica modalità in cui viene utilizzata la CPU) fino a 7 modalità low power, LPM0, LPM1, LPM2, LPM3, LPM4, LPM3.5 ed LPM4.5.

La configurazione e l'utilizzo mirato di queste modalità , dà modo al programmatore di attuare scelte che vanno ad incidere molto sul dispendio energetico, sulla velocità del micro stesso ed anche sulla precisione.

In altre parole, più si va verso le modalità LPMx.5, più la disponibilità di periferiche attive si fa scarsa, più diminuisce il dispendio energetico, riducendo potenza dinamica in maniera ponderata, al fine di ridurre consumo complessivo.

I pro dell'utilizzo di questo microcontrollore sono i consumi bassi in modalità low power SLEEP, la disponibilità sul componente delle periferiche necessarie per l'implementazione del firmware pensato, come Protocollo di Trasmissione Seriale UART ad esempio, e soprattutto la sua disponibilità immediata in laboratorio.

Esso è il vero e proprio cuore del progetto, e oggetto di studio in questa tesi.

Il suo utilizzo è stato quindi da subito considerato coerente con le richieste di progetto, quindi per organizzare un sistema di comunicazione con la WuR, possibilmente asincrono (senza necessità di sincronizzazione col ricevitore), a basso consumo e che rispetti determinate tempistiche per garantire la ricezione di pacchetti di comunicazione in un lasso di tempo limitato.

## 3.1 Caratteristiche principali

Il costruttore dichiara consumi nominali molto bassi, soprattutto nelle modalità a bassa potenza più basse.

Di seguito sono stati riportati gli screenshot delle tabelle prese da datasheet, raffiguranti i valori in corrente dichiarati dal costruttore.

Si può notare come all'abbassarsi degli stati, si ottengano correnti sempre più basse, fino a raggiungere dei consumi tipici di 200nA in LPM4, a partire dai 210uA in Active Mode.

### 5.4 Active Mode Supply Current Into $V_{CC}$ Excluding External Current

over recommended operating free-air temperature (unless otherwise noted)(1) (2)

| PARAMETER | EXECUTION MEMORY | $V_{CC}$ | FREQUENCY ($f_{MCLK} = f_{SMCLK}$) | | | | | | | | | | UNIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 MHz 0 wait states (NWAITSx = 0) | | 4 MHz 0 wait states (NWAITSx = 0) | | 8 MHz 0 wait states (NWAITSx = 0) | | 12 MHz 1 wait states (NWAITSx = 1) | | 16 MHz 1 wait states (NWAITSx = 1) | | |
| | | | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | |
| $I_{AM, FRAM\_UNI}$ (Unified memory)(3) | FRAM | 3.0 V | 210 | | 640 | | 1220 | | 1475 | | 1845 | | µA |
| $I_{AM, FRAM}$(0%)(4) (5) | FRAM 0% cache hit ratio | 3.0 V | 370 | | 1280 | | 2510 | | 2080 | | 2650 | | µA |
| $I_{AM, FRAM}$(50%)(4) (5) | FRAM 50% cache hit ratio | 3.0 V | 240 | | 745 | | 1440 | | 1575 | | 1990 | | µA |
| $I_{AM, FRAM}$(66%)(4) (5) | FRAM 66% cache hit ratio | 3.0 V | 200 | | 560 | | 1070 | | 1300 | | 1620 | | µA |
| $I_{AM, FRAM}$(75%)(4) (5) | FRAM 75% cache hit ratio | 3.0 V | 170 | 255 | 480 | | 890 | 1085 | 1155 | 1310 | 1420 | 1620 | µA |
| $I_{AM, FRAM}$(100%(4) (5) | FRAM 100% cache hit ratio | 3.0 V | 110 | | 235 | | 420 | | 640 | | 730 | | µA |
| $I_{AM, RAM}$ (6) | RAM | 3.0 V | 130 | | 320 | | 585 | | 890 | | 1070 | | µA |
| $I_{AM, RAM\ only}$ (7) (5) | RAM | 3.0 V | 100 | 180 | 290 | | 555 | | 860 | | 1040 | 1300 | µA |

(1) All inputs are tied to 0 V or to $V_{CC}$. Outputs do not source or sink any current.
(2) Characterized with program executing typical data processing.
$f_{ACLK}$ = 32768 Hz, $f_{MCLK} = f_{SMCLK} = f_{DCO}$ at specified frequency, except for 12 MHz. For 12 MHz, $f_{DCO}$= 24 MHz and $f_{MCLK} = f_{SMCLK} = f_{DCO}/2$.
At MCLK frequencies above 8 MHz, the FRAM requires wait states. When wait states are required, the effective MCLK frequency ($f_{MCLK,eff}$) decreases. The effective MCLK frequency also depends on the cache hit ratio. SMCLK is not affected by the number of wait states or the cache hit ratio.
The following equation can be used to compute $f_{MCLK,eff}$:
$f_{MCLK,eff} = f_{MCLK}$ / [wait states × (1 - cache hit ratio) + 1]
For example, with 1 wait state and 75% cache hit ratio $f_{MCKL,eff} = f_{MCLK}$ / [1 × (1 - 0.75) + 1] = $f_{MCLK}$ / 1.25.
(3) Represents typical program execution. Program and data reside entirely in FRAM. All execution is from FRAM.
(4) Program resides in FRAM. Data resides in SRAM. Average current dissipation varies with cache hit-to-miss ratio as specified. Cache hit ratio represents number cache accesses divided by the total number of FRAM accesses. For example, a 75% ratio implies three of every four accesses is from cache, and the remaining are FRAM accesses.
(5) See Figure 5-1 for typical curves. Each characteristic equation shown in the graph is computed using the least squares method for best linear fit using the typical data shown in Section 5.4.
(6) Program and data reside entirely in RAM. All execution is from RAM.
(7) Program and data reside entirely in RAM. All execution is from RAM. FRAM is off.

*Illustrazione 4: Corrente di Run dello stato di "Active Mode"*

Nell' "Illustrazione 4" possiamo vedere i valori della corrente in Active Mode, rappresentata a seconda delle percentuali di cache hit ratio e della memoria dalla quale la CPU va a prendere le informazioni necessarie al suo funzionamento.

Viene anche rappresentata in funzione delle frequenze di funzionamento della CPU, selezionabili via software.

### 5.6 Low-Power Mode (LPM0, LPM1) Supply Currents Into $V_{CC}$ Excluding External Current

over recommended operating free-air temperature (unless otherwise noted)(1) (2)

| PARAMETER | $V_{CC}$ | FREQUENCY ($f_{SMCLK}$) | | | | | | | | | | UNIT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 MHz | | 4 MHz | | 8 MHz | | 12 MHz | | 16 MHz | | |
| | | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | |
| $I_{LPM0}$ | 2.2 V | 70 | | 95 | | 150 | | 250 | | 215 | | µA |
| | 3.0 V | 80 | 115 | 105 | | 160 | | 260 | | 225 | 260 | |
| $I_{LPM1}$ | 2.2 V | 35 | | 60 | | 115 | | 215 | | 180 | | µA |
| | 3.0 V | 35 | 60 | 60 | | 115 | | 215 | | 180 | 205 | |

(1) All inputs are tied to 0 V or to $V_{CC}$. Outputs do not source or sink any current.
(2) Current for watchdog timer clocked by SMCLK included.
$f_{ACLK}$ = 32768 Hz, $f_{MCLK}$ = 0 MHz, $f_{SMCLK}$ = $f_{DCO}$ at specified frequency - except for 12 MHz: here $f_{DCO}$=24MHz and $f_{SMCLK}$ = $f_{DCO}$/2.

*Illustrazione 5: Correnti negli stati LPM0 e LPM1*

Nell "Illustrazione 5" abbiamo le correnti nei due stati prossimi all'active mode, ovvero LPM0 ed LPM1; sono gli stati entro i quali le periferiche di clock sono ancora tutte disponibili, ad eccezion fatta per il bus del Main Clock (diretto alla CPU), attivo solo in Active Mode.

### 5.7 Low-Power Mode (LPM2, LPM3, LPM4) Supply Currents (Into $V_{CC}$) Excluding External Current

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (1)

| PARAMETER | | $V_{CC}$ | -40 °C | | 25 °C | | 60 °C | | 85°C | | UNIT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | |
| $I_{LPM2,XT12}$ | Low-power mode 2, 12-pF crystal(2) (3) (4) | 2.2 V | 0.5 | | 0.9 | | 2.2 | | 6.1 | | µA |
| | | 3.0 V | 0.5 | | 0.9 | 1.8 | 2.2 | | 6.1 | 17 | |
| $I_{LPM2,XT3.7}$ | Low-power mode 2, 3.7-pF cyrstal(2) (5) (4) | 2.2 V | 0.5 | | 0.9 | | 2.2 | | 6.0 | | µA |
| | | 3.0 V | 0.5 | | 0.9 | | 2.2 | | 6.0 | | |
| $I_{LPM2,VLO}$ | Low-power mode 2, VLO, includes SVS(6) | 2.2 V | 0.3 | | 0.7 | | 1.9 | | 5.8 | | µA |
| | | 3.0 V | 0.3 | | 0.7 | 1.6 | 1.9 | | 5.8 | 16.7 | |
| $I_{LPM3,XT12}$ | Low-power mode 3, 12-pF crystal, excludes SVS(2) (3) (7) | 2.2 V | 0.5 | | 0.6 | | 0.9 | | 1.85 | | µA |
| | | 3.0 V | 0.5 | | 0.6 | 0.9 | 0.9 | | 1.85 | 4.9 | |
| $I_{LPM3,XT3.7}$ | Low-power mode 3, 3.7-pF cyrstal, excludes SVS(2) (5) (8) (also refer to Figure 5-2) | 2.2 V | 0.4 | | 0.5 | | 0.8 | | 1.7 | | µA |
| | | 3.0 V | 0.4 | | 0.5 | | 0.8 | | 1.7 | | |
| $I_{LPM3,VLO}$ | Low-power mode 3, VLO, excludes SVS(9) | 2.2 V | 0.3 | | 0.4 | | 0.7 | | 1.6 | | µA |
| | | 3.0 V | 0.3 | | 0.4 | 0.7 | 0.7 | | 1.6 | 4.7 | |
| $I_{LPM4,SVS}$ | Low-power mode 4, includes SVS(10) (also refer to Figure 5-3) | 2.2 V | 0.4 | | 0.5 | | 0.8 | | 1.7 | | µA |
| | | 3.0 V | 0.4 | | 0.5 | 0.8 | 0.8 | | 1.7 | 4.8 | |
| $I_{LPM4}$ | Low-power mode 4, excludes SVS(11) | 2.2 V | 0.2 | | 0.3 | | 0.6 | | 1.5 | | µA |
| | | 3.0 V | 0.2 | | 0.3 | 0.6 | 0.6 | | 1.5 | 4.6 | |

*Illustrazione 6: Correnti negli stati LPM2, LPM3 e LPM4*

Nell' "Illustrazione 6" abbiamo i valori di corrente dei diversi stati che vanno da LPM2 ad LPM4, ovvero gli stati dai quali non è più disponibile oltre al Main Clock, anche il cosiddetto Sub Main Clock.

Negli stati considerati in figura possono essere attivi infatti solo l'Auxiliary Clock (VLO oppure MODOSC) e l'External Clock, al quale si può collegare un oscillatore al quarzo molto preciso e stabile ad alta frequenza, ma del quale va implementata la circuiteria hardware da aggiungere alla scheda, oppure si può anche collegare un oscillatore al quarzo già presente sulla scheda, dal risparmio energetico evidente, in quanto molto lento (32KHz) ma con una tolleranza del 20%, quindi poco preciso ed affidabile per applicazioni di tipo fast response.

**Low-Power Mode (LPM2, LPM3, LPM4) Supply Currents (Into $V_{CC}$) Excluding External Current *(continued)***

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (1)

| PARAMETER | | $V_{CC}$ | -40 °C | | 25 °C | | 60 °C | | 85°C | | UNIT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | |
| $I_{IDLE,GroupA}$ | Additional idle current if one or more modules from Group A (refer to Table 6-2) are activated in LPM3 or LPM4. | 3.0V | | | 0.02 | | | | 0.33 | 1.3 | µA |
| $I_{IDLE,GroupB}$ | Additional idle current if one or more modules from Group B (refer to Table 6-2) are activated in LPM3 or LPM4 | 3.0V | | | 0.015 | | | | 0.25 | 1.0 | µA |

**5.8 Low-Power Mode (LPM3.5, LPM4.5) Supply Currents (Into $V_{CC}$) Excluding External Current**

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)(1)

| PARAMETER | | $V_{CC}$ | -40 °C | | 25 °C | | 60 °C | | 85°C | | UNIT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | TYP | MAX | TYP | MAX | TYP | MAX | TYP | MAX | |
| $I_{LPM3.5,XT12}$ | Low-power mode 3.5, 12-pF crystal, includes SVS(2)(3)(4) | 2.2 V | 0.4 | | 0.45 | | 0.5 | | 0.7 | | µA |
| | | 3.0 V | 0.4 | | 0.45 | 0.7 | 0.5 | | 0.7 | 1.2 | |
| $I_{LPM3.5,XT3.7}$ | Low-power mode 3.5, 3.7-pF cyrstal, excludes SVS(2)(5)(6) (also refer to Figure 5-4) | 2.2 V | 0.2 | | 0.25 | | 0.3 | | 0.45 | | µA |
| | | 3.0 V | 0.2 | | 0.25 | | 0.3 | | 0.5 | | |
| $I_{LPM4.5,SVS}$ | Low-power mode 4.5, includes SVS(7) (also refer to Figure 5-5) | 2.2 V | 0.2 | | 0.2 | | 0.2 | | 0.3 | | µA |
| | | 3.0 V | 0.2 | | 0.2 | 0.4 | 0.2 | | 0.3 | 0.55 | |
| $I_{LPM4.5}$ | Low-power mode 4.5, excludes SVS(8) (also refer to Figure 5-5) | 2.2 V | 0.02 | | 0.02 | | 0.02 | | 0.08 | | µA |
| | | 3.0 V | 0.02 | | 0.02 | | 0.02 | | 0.08 | 0.35 | |

(1) All inputs are tied to 0 V or to $V_{CC}$. Outputs do not source or sink any current.
(2) Not applicable for devices with HF crystal oscillator only.
(3) Characterized with a Micro Crystal MS1V-T1K crystal with a load capacitance of 12.5 pF. The internal and external load capacitance are chosen to closely match the required 12.5-pF load.
(4) **Low-power mode 3.5, 12-pF crystal, includes SVS** test conditions:
Current for RTC clocked by XT1 is included. Current for brownout and SVS are included (SVSHE = 1). Core regulator is disabled.
PMMREGOFF = 1, CPUOFF = 1, SCG0 = 1 SCG1 = 1, OSCOFF = 1 (LPMx.5),

*Illustrazione 7(1): Correnti addizionali nelle transizioni da LPM4/LPM3 a stati con clock non attivi in questi ultimi*

*Illustrazione 7(2): Correnti negli stati LPM 3.5 ed LPM 4.5*

Nelle due figure in "Illustrazione 7", troviamo i valori tipici e massimi della corrente impiegata per i passaggi dagli stati low power a stati di active o richiesti tramite wake up del micro, dovuti al tempo che intercorre nel passaggio tra uno stato ed un altro ed iconsumi tipici e nominali della corrente al variare della temperatura, negli stati di dormienza più profonda.

Questi stati hanno un consumo estremamente ridotto, dovuto anche al fatto di avere una disponibilità di periferiche accese molto limitata.
Non hanno ritenzione di memoria, ed inoltre hanno tempistiche di risveglio molto più lunghe rispetto agli altri stati di dormienza meno spinti, come si può notare dalla figura seguente dei tempi di wake up.

### 5.12.4 Wake-Up Characteristics

**Table 5-9. Wakeup Times From Low-Power Modes and Reset**

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

| PARAMETER | | TEST CONDITIONS | $V_{CC}$ | MIN | TYP | MAX | UNIT |
|---|---|---|---|---|---|---|---|
| $t_{WAKE-UP\ FRAM}$ | (Additional) wakeup time to activate the FRAM in AM if previously disabled by the FRAM controller or from an LPM if immediate activation is selected for wake up | | | | 6 | 10 | µs |
| $t_{WAKE-UP\ LPM0}$ | Wakeup time from LPM0 to active mode(1) | | 2.2 V, 3.0 V | | | 400 ns + 1.5/$f_{DCO}$ | |
| $t_{WAKE-UP\ LPM1}$ | Wakeup time from LPM1 to active mode(1) | | 2.2 V, 3.0 V | | 6 | | µs |
| $t_{WAKE-UP\ LPM2}$ | Wakeup time from LPM2 to active mode(1) | | 2.2 V, 3.0 V | | 6 | | µs |
| $t_{WAKE-UP\ LPM3}$ | Wakeup time from LPM3 to active mode(1) | | 2.2 V, 3.0 V | | 7 | 10 | µs |
| $t_{WAKE-UP\ LPM4}$ | Wakeup time from LPM4 to active mode(1) | | 2.2 V, 3.0 V | | 7 | 10 | µs |
| $t_{WAKE-UP\ LPM3.5}$ | Wakeup time from LPM3.5 to active mode(2) | | 2.2 V, 3.0 V | | 250 | 350 | µs |
| $t_{WAKE-UP\ LPM4.5}$ | Wakeup time from LPM4.5 to active mode(2) | SVSHE = 1 | 2.2 V, 3.0 V | | 250 | 350 | µs |
| | | SVSHE = 0 | 2.2 V, 3.0 V | | 1 | 1.5 | ms |
| $t_{WAKE-UP-RST}$ | Wakeup time from a $\overline{RST}$ pin triggered reset to active mode(2) | | 2.2 V, 3.0 V | | 250 | 350 | µs |
| $t_{WAKE-UP-BOR}$ | Wakeup time from power-up to active mode (2) | | 2.2 V, 3.0 V | | 1 | 1.5 | ms |

(1) The wakeup time is measured from the edge of an external wakeup signal (for example, port interrupt or wakeup event) to the first externally observable MCLK clock edge. MCLK is sourced by the DCO and the MCLK divider is set to divide-by-1 (DIVMx = 000b, $f_{MCLK} = f_{DCO}$). This time includes the activation of the FRAM during wake up.
(2) The wakeup time is measured from the edge of an external wakeup signal (for example, port interrupt or wakeup event) until the first instruction of the user program is executed.

*Illustrazione 8: Tempistiche di risveglio dai diversi stati Low Power*

### 5.10 Typical Characteristics, Current Consumption per Module(1)

| MODULE | TEST CONDITIONS | REFERENCE CLOCK | MIN | TYP | MAX | UNIT |
|---|---|---|---|---|---|---|
| Timer_A | | Module input clock | | 3 | | µA/MHz |
| Timer_B | | Module input clock | | 5 | | µA/MHz |
| eUSCI_A | UART mode | Module input clock | | 5.5 | | µA/MHz |
| eUSCI_A | SPI mode | Module input clock | | 3.5 | | µA/MHz |
| eUSCI_B | SPI mode | Module input clock | | 3.5 | | µA/MHz |
| eUSCI_B | $I^2C$ mode, 100 kbaud | Module input clock | | 3.5 | | µA/MHz |
| RTC_B | | 32 kHz | | 100 | | nA |
| MPY | Only from start to end of operation | MCLK | | 25 | | µA/MHz |
| AES | Only from start to end of operation | MCLK | | 21 | | µA/MHz |
| CRC | Only from start to end of operation | MCLK | | 2.5 | | µA/MHz |

*Illustrazione 9: Corrente dinamica imputata alle periferiche utilizzate*

Nell' "Illustrazione 9" vediamo i consumi di corrente tipici delle diverse periferiche; come si può notare, i valori rappresentati sono caratteristici di una stima dinamica della corrente dovuta ai singoli blocchi presi singolarmente.

**Table 6-2. Peripheral Groups**

| Group A | Group B |
|---|---|
| Timer TA1 | Timer TA0 |
| Timer TA2 | Timer TA3 |
| Timer TB0 | Comparator |
| eUSCI_A0 | ADC12_B |
| eUSCI_A1 | REF_A |
| eUSCI_B0 | |

*Illustrazione 10: Classificazione Gruppi A e B di appartenenza delle periferiche*

Nella tabella sopra invece, si può notare la disposizione delle diverse periferiche del micro, nei due macrogruppi di classificazione di consumi, ideata dalla Texas Instruments.

## 3.2 Modalità Low Power

L'MSP430 possiede un active mode e sette modalità low-power selezionabili via software.

Un evento da interrupt può svegliare il dispositivo da un Low Power Mode qualsiasi che sia al massimo l'LPM4, servire la richiesta di Interrupt, e quindi la Routine interessata, e ritornare indietro alla modalità low power di partenza.

Gli stati low power LPM3.5 ed LPM4.5 disabilitano l'alimentazione del core per minimizzare il consumo di potenza, a discapito però della ritenzione di memoria.

**Table 6-1. Operating Modes**

| Mode | AM | | LPM0 | LPM1 | LPM2 | LPM3 | LPM4 | LPM3.5 | LPM4.5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Active | Active, FRAM Off[1] | CPU Off[2] | CPU Off | Standby | Standby | Off | RTC only | Shutdown with SVS | Shutdown without SVS |
| Maximum System Clock | 16 MHz | | 16 MHz | 16 MHz | 50 kHz | 50 kHz | 0[3] | 50 kHz | 0[3] | |
| Typical Current Consumption, $T_A$ = 25°C | 103 µA/MHz | 65 µA/MHz | 70 µA at 1 MHz | 35 µA at 1 MHz | 0.7 µA | 0.4 µA | 0.3 µA | 0.25 µA | 0.2 µA | 0.02 µA |
| Typical Wake-up Time | N/A | | instant | 6 µs | 6 µs | 7 µs | 7 µs | 250 µs | 250 µs | 1000 µs |
| Wake-Up Events | N/A | | all | all | LF I/O Comp | LF I/O Comp | I/O Comp | RTC I/O | I/O | |
| CPU | on | | off | off | off | off | off | reset | reset | |
| FRAM | on | off[1] | standby (or off[1]) | off | off | off | off | off | off | |
| High-Frequency Peripherals | available | | available | available | off | off | off | reset | reset | |
| Low-Frequency Peripherals | available | | available | available | available | available[4] | off | RTC | reset | |
| Unclocked Peripherals[5] | available | | available | available | available | available[4] | available[4] | reset | reset | |
| MCLK | on | | off | off | off | off | off | off | off | |
| SMCLK | optional[6] | | optional[6] | optional[6] | off | off | off | off | off | |
| ACLK | on | | on | on | on | on | off | off | off | |
| Full Retention | yes | | yes | yes | yes | yes | yes | no | no | |
| SVS | always | | always | always | optional[7] | optional[7] | optional[7] | optional[7] | on[8] | off[9] |
| Brownout | always | | always | always | always | always | always | always | always | |

*Illustrazione 11: Modi Operativi di funzionamento del Microcontrollore*

## 3.3 Modulo del Clock di Sistema

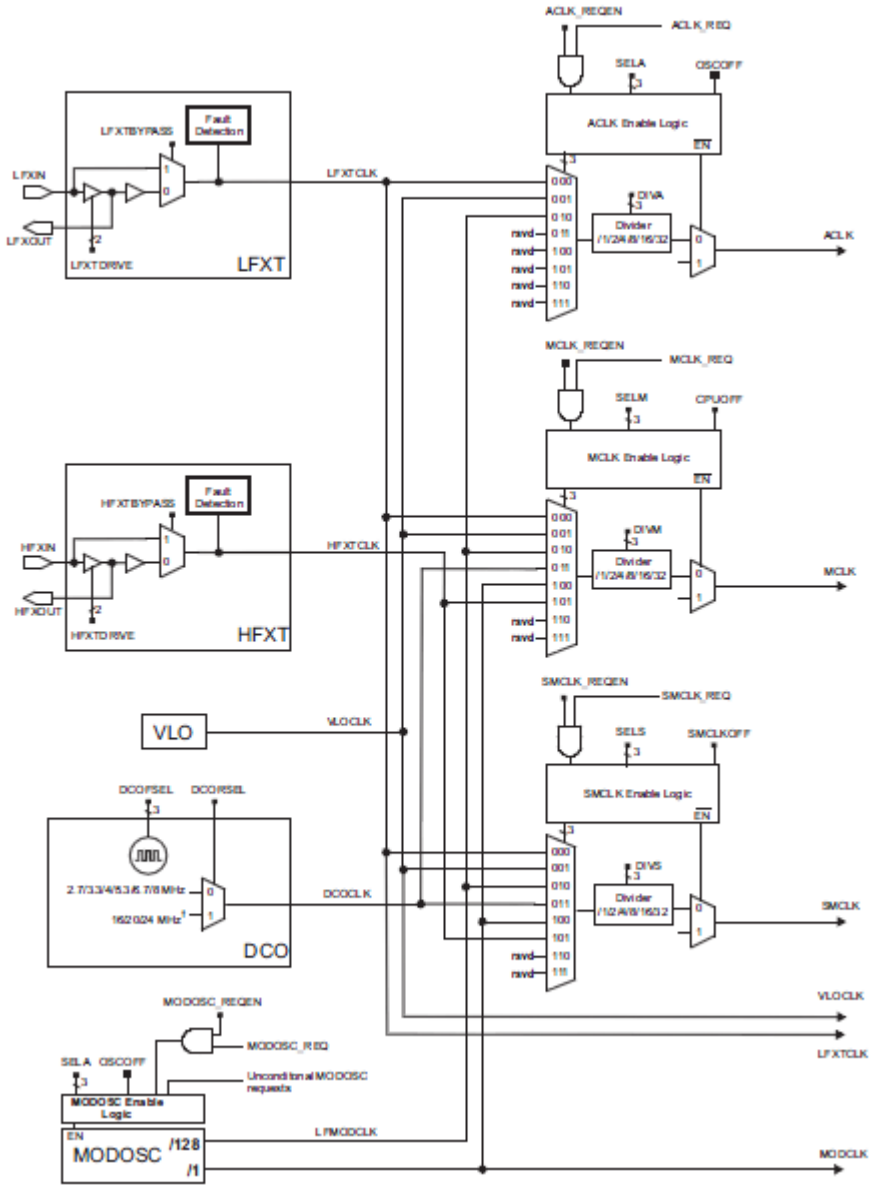


*Illustrazione 12: diagramma a blocchi del modulo del clock di Sistema*

Il modulo del clock di sistema supporta sistemi a basso costo e basso consumo energetico.

Utilizzando tre segnali di clock, l'utente può selezionare il miglior bilancio per la prestazione del consumo di potenza da parte del micro.

Il modulo di clock può essere configurato per operare senza componenti esterni, con uno o due cristalli esterni o con risonatori, sotto il completo controllo dell'utente.

Questo modulo include le seguenti sorgenti di clock:

- LFXTCLK: oscillatore a bassa frequenza che può essere utilizzato anche avvalendosi di un cristallo presente sulla board di 32KHz, oppure con cristalli standard, risonatori o sorgenti di clock esterne nel range dei 50KHz o meno.
- VLOCLK: è un oscillatore interno di bassissimo consumo e con una frequenza tipica di 10KHz.
- DCOCLK: è un DCO (acronimo di Digital Controlled Oscillator) molto preciso, con frequenze selezionabili via software.
- MODCLK: oscillatore a bassa potenza con una frequenza tipica nell'ordine dei 5MHz. LFMODCLK è il MODCLK diviso per 128.
- HFXTCLK: oscillatore ad alta frequenza utilizzabile con cristalli standard ad HF nel range di frequenze da 4 a 24 Mhz.

Dal modulo di clock sono poi disponibili quattro bus di clock, in cui collocarci i clock desiderati, e sono:

- ACLK (acronimo di Auxiliary Clock): è un bus selezionabile via software come bus di LFXTCLK, VLOCLK oppure LFMODCLK. ACLK può essere poi ulteriormente diviso di 1,2,4,8,16 o 32, tramite divisori interni al micro selezionabili anch'essi via software.
- MCLK (acronimo di Master Clock): questo bus è selezionabile via software come bus di qualsiasi sorgente di clock presente sul micro. Supporta le sorgenti a più alta frequenza, e quindi anche il DCO, fondamentale per applicazioni accurate e di estrema precisione e rapidità. Come l'ACLK può essere ulteriormente diviso una volta selezionatasi la sua frequenza di lavoro, per 1,2,4,8,16,32.
- SMCLK (acronimo di Sub-System Master Clock): selezionabile via software, come MCLK può supportare ogni tipo di clock presente sul micro o esterno, ad alta e bassa frequenza.
- MODCLK e VLO possono poi essere utilizzati singolarmente come sorgenti di clock di periferiche, selezionabile via software.

## 3.4 eUSCI (Enhanced Serial Communication Interface)

La periferica eUSCI supporta tre tipi di comunicazione seriale, ovvero SPI, I2C e UART.

Nel progetto si è scelto di adottare una comunicazione seriale asincrona, che non necessitasse sincronia di clock per evitare di doverla ricavare dalla portante UHF, quindi si è ricaduti sulla comunicazione seriale asincrona ottenuta tramite periferica UART.

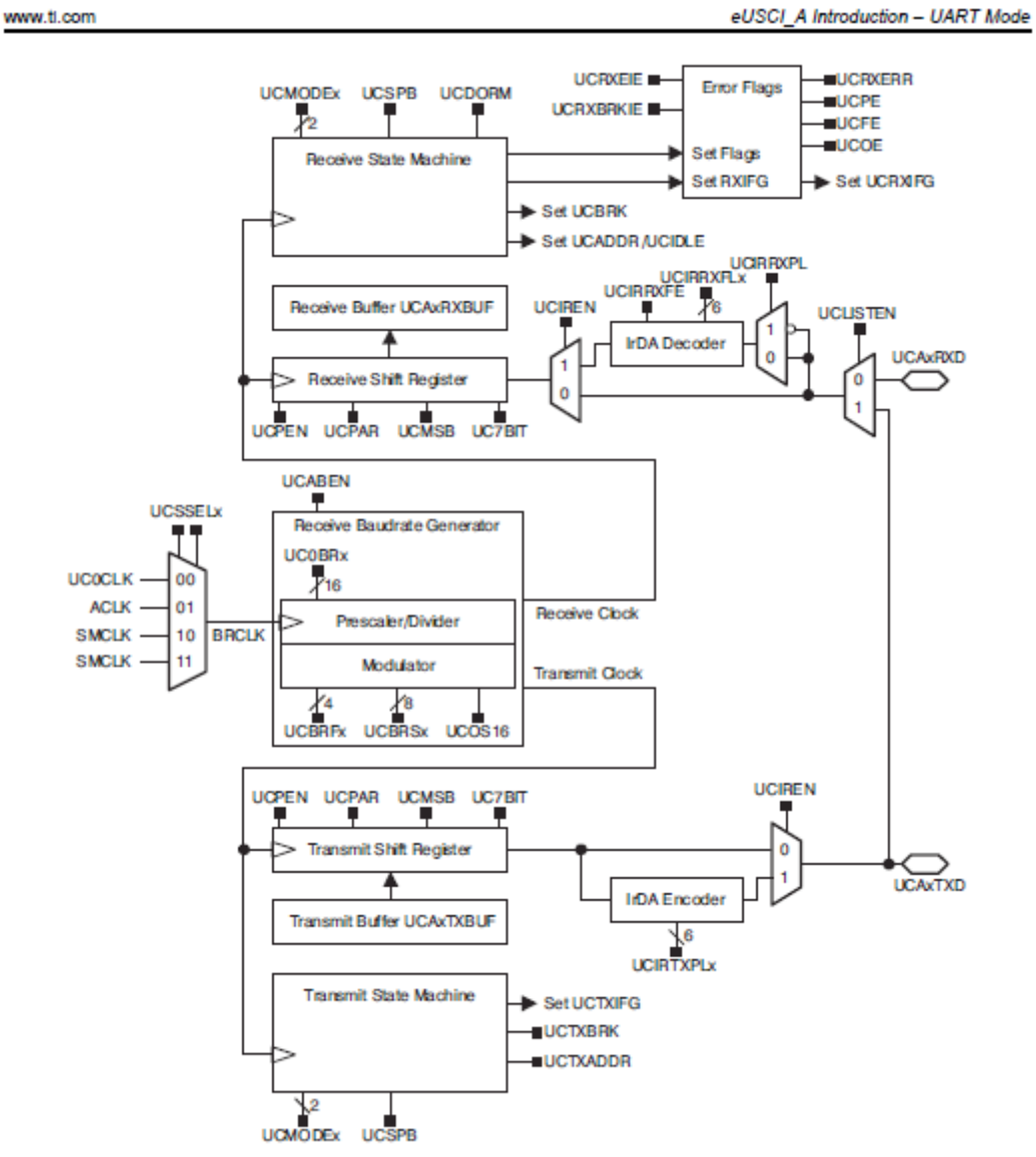


**Figure 21-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0)**

*Illustrazione 13: Diagramma a blocchi dell' eUSCI_A configurata in UART Mode*

## Introduzione modulo eUSCI_A

Nella modalità asincrona, i moduli eUSCI_Ax connettono il dispositivo ad un sistema esterno attraverso 2pin, UCARXD e UCATXD.

La modalità UART è selezionata quando il bit UCSYNC viene pulito.

Questa modalità include diverse funzionalità:

- la configurazione della trama può essere da 7 o 8 bit, con parità pari, parità dispari o senza parità.
- Trasmissione e Ricezione possiedono due registri a scorrimento indipendenti tra loro
- Una separazione tra registri buffer lato trasmissione e ricezione
- possibilità di configurare i dati da trasmettere e ricevere come LSB-first o MSB-first
- Baud Rate programmabile con modulazione, per supportare baud rate frazionali
- registri di stato per rilevamento errori e soppressione
- registri di stato per rilevamento indirizzi
- interrupt indipendenti per ricezione, trasmissione, ricezione start bit, completamento trasmissione
- Ricezione Start-Edge automatica per risveglio automatico da LPMx modalità (non è supportato il risveglio da modalità LPMx.5)

**Table 21-7. eUSCI_A UART Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | UCAxCTLW0 | eUSCI_Ax Control Word 0 | Read/write | Word | 0001h | Section 21.4.1 |
| 01h | UCAxCTL0[1] | eUSCI_Ax Control 0 | Read/write | Byte | 00h | |
| 00h | UCAxCTL1 | eUSCI_Ax Control 1 | Read/write | Byte | 01h | |
| 02h | UCAxCTLW1 | eUSCI_Ax Control Word 1 | Read/write | Word | 0003h | Section 21.4.2 |
| 06h | UCAxBRW | eUSCI_Ax Baud Rate Control Word | Read/write | Word | 0000h | Section 21.4.3 |
| 06h | UCAxBR0[1] | eUSCI_Ax Baud Rate Control 0 | Read/write | Byte | 00h | |
| 07h | UCAxBR1 | eUSCI_Ax Baud Rate Control 1 | Read/write | Byte | 00h | |
| 08h | UCAxMCTLW | eUSCI_Ax Modulation Control Word | Read/write | Word | 00h | Section 21.4.4 |
| 0Ah | UCAxSTATW | eUSCI_Ax Status | Read/write | Word | 00h | Section 21.4.5 |
| 0Ch | UCAxRXBUF | eUSCI_Ax Receive Buffer | Read/write | Word | 00h | Section 21.4.6 |
| 0Eh | UCAxTXBUF | eUSCI_Ax Transmit Buffer | Read/write | Word | 00h | Section 21.4.7 |
| 10h | UCAxABCTL | eUSCI_Ax Auto Baud Rate Control | Read/write | Word | 00h | Section 21.4.8 |
| 12h | UCAxIRCTL | eUSCI_Ax IrDA Control | Read/write | Word | 0000h | Section 21.4.9 |
| 12h | UCAxIRTCTL | eUSCI_Ax IrDA Transmit Control | Read/write | Byte | 00h | |
| 13h | UCAxIRRCTL | eUSCI_Ax IrDA Receive Control | Read/write | Byte | 00h | |
| 1Ah | UCAxIE | eUSCI_Ax Interrupt Enable | Read/write | Word | 00h | Section 21.4.10 |
| 1Ch | UCAxIFG | eUSCI_Ax Interrupt Flag | Read/write | Word | 02h | Section 21.4.11 |
| 1Eh | UCAxIV | eUSCI_Ax Interrupt Vector | Read | Word | 0000h | Section 21.4.12 |

*Illustrazione 14: Registri dell'eUSCI_A*

Nell' "Illustrazione 14" sono indicati i registri di configurazione per applicazioni UART e i loro offset di default.

Il registro UCAxCTLW0 serve al settaggio del tipo di pacchetto ricevuto o trasmesso, all'abilitazione della parità, al rifiuto di caratteri errati, al settaggio di modalità funzionamento UART, alla configurazione opzionale della modalità sincrona,...

Il registro UCAxCTLW1 serve nel caso si desideri effettuare una "deglitch suppression" non utilizzata nel progetto.

L'UCAxBRW, insieme all' UCAxMCTLW, gestisce la generazione interna della baud rate della periferica; in particolare c'è un bit da impostare ad uno nell'UCAxMCTLW, ovvero l'UCOS16 che serve ad abilitare la modalità di OverSampling a 16bit.

Altri registri sono utilizzabili per settare una modalità di generazione baud rate automatica, in base alla natura dei pacchetti in ricezione.

UCAxIE contiene le abilitazioni degli interrupt disponibili nella seriale.

UCRXBUFx è il buffer interno, nel quale viene copiato il dato in ricezione e se viene letto, avviene automaticamente il clean dell'UCRXIFG (flag di interrupt del ricevitore seriale).

UCTXBUFx è un buffer analogo a quello sopra, ma inerente al lato trasmissione: se viene trasmesso un dato viene copiato prima in UCTXBUFx, quando viene riscritto avviene automaticamente il clean dell'UCTXIFG.

## UART Baud Rate Generation

L'eUSCI_A, ovvero il modulo utilizzato come protocollo di comunicazione Uart, permette la generazione di Baud Rate standard, attraverso sorgenti di clock non standard.

È possibile fare ciò attraverso due modalità differenti, caratterizzati dal valore associato al bit UCOS16, che seleziona appunto una modalità di generazione di Baud Rate o l'altra.

Le due modalità sono:

- Low Frequency Baud Rate Generation
- Oversampling Baud Rate Generation

### Low Frequency Baud Rate Generation

Questa modalità viene selezionata impostando UCOS16 = 0.

questa modalità permette la generazione di Baud Rate attraverso sorgenti di clock a bassa frequenza, come ad esempio un VLO a 9600Hz, fino a 32KHz.

Utilizzando un input a bassa frequenza, il consumo dinamico dell'intero modulo viene ridotto.

Utilizzare un input ad alta frequenza, causa inoltre una riduzione della finestra entro la quale vengono campionati 3 campioni di stima, per la generazione, cosa che riduce l'efficacia del

campionamento appunto.

Nella modalità a Bassa Frequenza, il modulo utilizza un modulatore ed un prescaler per generare la temporizzazione di bit.

In questa modalità inoltre la massima frequenza di utilizzo per la Baud Rate è un terzo della frequenza utilizzata in input al modulo.

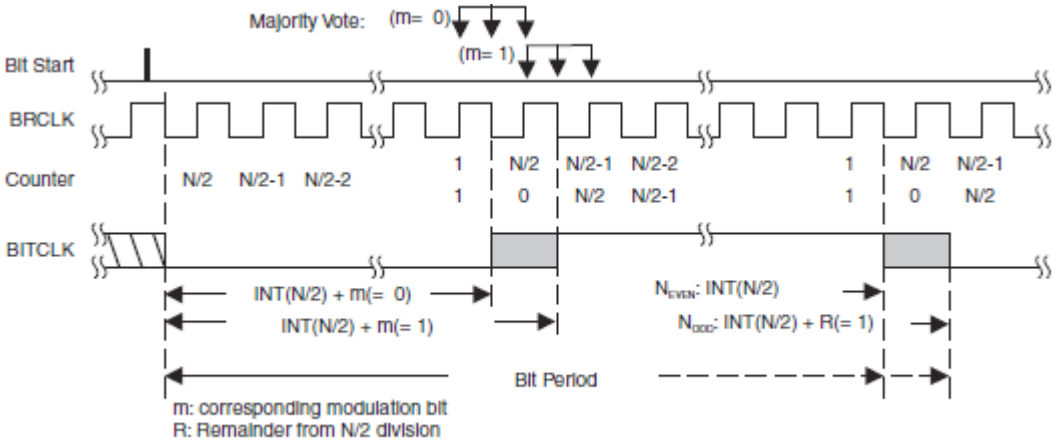


**Figure 21-10. BITCLK Baud-Rate Timing With UCOS16 = 0**

*Illustrazione 15: Temporizzazione della Baud Rate nella modalità a Bassa Frequenza*

Nell' "Illustrazione 15" si può vedere come funziona il Bit Timing nella modalità spiegata.

Per ogni bit ricevuto, viene determinato un "voto a maggioranza", ovvero vengono presi 3 campioni, con una finestra selezionata dalla modulazione settata attraverso il registro UCBRSx, che si può settare riferendosi alla User Guide dell'MSP430FR5969, al fine di calcolare il valore di bit.

Questi campioni vengono presi ogni N/2 – 1/2, N/2, ed N/2 + 1/2 del periodo di BRCLK, dove N è il numero di BRCLKs per periodo.

## Oversampling Baud Rate Generation

Questa modalità è selezionata mettendo ad 1 UCOS16.

Questa modalità supporta un campionamento della trama di bit dell'UART, anche con frequenze di input elevate.
In questo caso viene attuato un "voto a maggioranza" su 16 bit di sovracampionamento, quindi il campionamento del Bit Timing è sempre corrispondente a 1/16 del periodo di clock.

Questa modalità utilizza un modulatore e un prescaler per realizzare una trama di bit detta BITCLK16 che è 16 volte più veloce della classica BITCLK, la quale poi entra in un'altro modulatore con divisore in cascata, dalla quale si ottiene poi all'uscita la BITCLK.

Chiaramente, in questa modalità, la massima frequenza di utilizzo per la Baud Rate equivale a 1/16

della frequenza di input al modulo.

La modulazione per BITCLK è basata sul settaggio di UCBRSx.

Table 21-3. BITCLK16 Modulation Pattern

| UCBRFx | Number of BITCLK16 Clocks After Last Falling BITCLK Edge | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 00h | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 01h | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 02h | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 03h | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 04h | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 05h | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 06h | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 07h | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 08h | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 09h | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0Ah | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0Bh | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0Ch | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0Dh | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0Eh | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0Fh | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*Illustrazione 16: Pattern di Modulazione per BITCLK*

## Settaggio della Baud Rate

Per una data sorgente di BRCLK, la Baud Rate usata determina il fattore di divisione N necessario per la generazione della Baud Rate:

$N = F_{BRCLK} / \text{baud-rate}$

Il fattore di divisione è spesso un numero non intero, per questo motivo almeno un divisore ed un modulatore vengono utilizzati per rendere questo numero più vicino possibile alla divisione esatta.

Se N è maggiore o uguale a 16, la Texas Instruments raccomanda l'utilizzo di SovraCampionamento, attraverso il settaggio ad 1 del bit UCOS16.

L'impostazione corrisponde alle seguenti operazioni nei due casi:

1. calcolo di N
2. se N<16 --> OS16=0; UCBRx = INT(N)
3. se N>=16 --> OS16=1; UCBRx = INT(N/16); UCBRFx = {INT[(N/16) – INT(N/16)] * 16}
4. UCBRSx si può selezionare calcolando N – INT(N) e utilizzando il codice elencato nella tabella seguentemente mostrata.

Table 21-4. UCBRSx Settings for Fractional Portion of N = $f_{BRCLK}$/Baud Rate

| Fractional Portion of N | UCBRSx(1) | | Fractional Portion of N | UCBRSx(1) |
|---|---|---|---|---|
| 0.0000 | 0x00 | | 0.5002 | 0xAA |
| 0.0529 | 0x01 | | 0.5715 | 0x6B |
| 0.0715 | 0x02 | | 0.6003 | 0xAD |
| 0.0835 | 0x04 | | 0.6254 | 0xB5 |
| 0.1001 | 0x08 | | 0.6432 | 0xB6 |
| 0.1252 | 0x10 | | 0.6667 | 0xD6 |
| 0.1430 | 0x20 | | 0.7001 | 0xB7 |
| 0.1670 | 0x11 | | 0.7147 | 0xBB |
| 0.2147 | 0x21 | | 0.7503 | 0xDD |
| 0.2224 | 0x22 | | 0.7861 | 0xED |
| 0.2503 | 0x44 | | 0.8004 | 0xEE |
| 0.3000 | 0x25 | | 0.8333 | 0xBF |
| 0.3335 | 0x49 | | 0.8464 | 0xDF |
| 0.3575 | 0x4A | | 0.8572 | 0xEF |
| 0.3753 | 0x52 | | 0.8751 | 0xF7 |
| 0.4003 | 0x92 | | 0.9004 | 0xFB |
| 0.4286 | 0x53 | | 0.9170 | 0xFD |
| 0.4378 | 0x55 | | 0.9288 | 0xFE |

*Illustrazione 17: Settaggio raccomandato di UCBRSx da datasheet*

Nel nostro caso la frequenza di bit è stata impostata a 9600Hz, quindi il divisore e il modulatore hanno i seguenti valori:

N = 1000000/9600;

UCBRx = 6;

UCBRSx = 0x20;

è fondamentale la corretta temporizzazione dell'UART, in quanto non vi è Handshake col segnale portante, quindi il tempo di bit trasmesso e quello di ricezione devono coincidere.

Nelle tabelle che seguono si può vedere una configurazione complessiva dei settaggi raccomandati da Texas Instruments, con annessi anche errori percentuali di margine destro e sinistro.

| BRCLK | Baud Rate | UCOS16 | UCBRx | UCBRFx | UCBRSx | TX Error (%) | | RX Error (%) | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | neg | pos | neg | pos |
| 32768 | 1200 | 1 | 1 | 11 | 0x25 | -2.29 | 2.25 | -2.56 | 5.35 |
| 32768 | 2400 | 0 | 13 | - | 0xB6 | -3.12 | 3.91 | -5.52 | 8.84 |
| 32768 | 4800 | 0 | 6 | - | 0xEE | -7.62 | 8.98 | -21 | 10.25 |
| 32768 | 9600 | 0 | 3 | - | 0x92 | -17.19 | 16.02 | -23.24 | 37.3 |
| 1000000 | 9600 | 1 | 6 | 8 | 0x20 | -0.48 | 0.64 | -1.04 | 1.04 |
| 1000000 | 19200 | 1 | 3 | 4 | 0x2 | -0.8 | 0.96 | -1.84 | 1.84 |
| 1000000 | 38400 | 1 | 1 | 10 | 0x0 | 0 | 1.76 | 0 | 3.44 |
| 1000000 | 57600 | 0 | 17 | - | 0x4A | -2.72 | 2.56 | -3.76 | 7.28 |
| 1000000 | 115200 | 0 | 8 | - | 0xD6 | -7.36 | 5.6 | -17.04 | 6.96 |
| 1048576 | 9600 | 1 | 6 | 13 | 0x22 | -0.46 | 0.42 | -0.48 | 1.23 |
| 1048576 | 19200 | 1 | 3 | 6 | 0xAD | -0.88 | 0.83 | -2.36 | 1.18 |
| 1048576 | 38400 | 1 | 1 | 11 | 0x25 | -2.29 | 2.25 | -2.56 | 5.35 |
| 1048576 | 57600 | 0 | 18 | - | 0x11 | -2 | 3.37 | -5.31 | 5.55 |
| 1048576 | 115200 | 0 | 9 | - | 0xD8 | -5.37 | 4.49 | -5.93 | 14.92 |
| 4000000 | 9600 | 1 | 26 | 0 | 0xB6 | -0.08 | 0.16 | -0.28 | 0.2 |
| 4000000 | 19200 | 1 | 13 | 0 | 0x84 | -0.32 | 0.32 | -0.64 | 0.48 |
| 4000000 | 38400 | 1 | 6 | 8 | 0x20 | -0.48 | 0.64 | -1.04 | 1.04 |
| 4000000 | 57600 | 1 | 4 | 5 | 0x55 | -0.8 | 0.64 | -1.12 | 1.76 |
| 4000000 | 115200 | 1 | 2 | 2 | 0xBB | -1.44 | 1.28 | -3.92 | 1.68 |
| 4000000 | 230400 | 0 | 17 | - | 0x4A | -2.72 | 2.56 | -3.76 | 7.28 |
| 4194304 | 9600 | 1 | 27 | 4 | 0xFB | -0.11 | 0.1 | -0.33 | 0 |
| 4194304 | 19200 | 1 | 13 | 10 | 0x55 | -0.21 | 0.21 | -0.55 | 0.33 |
| 4194304 | 38400 | 1 | 6 | 13 | 0x22 | -0.46 | 0.42 | -0.48 | 1.23 |
| 4194304 | 57600 | 1 | 4 | 8 | 0xEE | -0.75 | 0.74 | -2 | 0.87 |
| 4194304 | 115200 | 1 | 2 | 4 | 0x92 | -1.62 | 1.37 | -3.56 | 2.06 |
| 4194304 | 230400 | 0 | 18 | - | 0x11 | -2 | 3.37 | -5.31 | 5.55 |
| 8000000 | 9600 | 1 | 52 | 1 | 0x49 | -0.08 | 0.04 | -0.1 | 0.14 |
| 8000000 | 19200 | 1 | 26 | 0 | 0xB6 | -0.08 | 0.16 | -0.28 | 0.2 |
| 8000000 | 38400 | 1 | 13 | 0 | 0x84 | -0.32 | 0.32 | -0.64 | 0.48 |
| 8000000 | 57600 | 1 | 8 | 10 | 0xF7 | -0.32 | 0.32 | -1 | 0.36 |
| 8000000 | 115200 | 1 | 4 | 5 | 0x55 | -0.8 | 0.64 | -1.12 | 1.76 |
| 8000000 | 230400 | 1 | 2 | 2 | 0xBB | -1.44 | 1.28 | -3.92 | 1.68 |
| 8000000 | 460800 | 0 | 17 | - | 0x4A | -2.72 | 2.56 | -3.76 | 7.28 |
| 8388608 | 9600 | 1 | 54 | 9 | 0xEE | -0.06 | 0.06 | -0.11 | 0.13 |
| 8388608 | 19200 | 1 | 27 | 4 | 0xFB | -0.11 | 0.1 | -0.33 | 0 |
| 8388608 | 38400 | 1 | 13 | 10 | 0x55 | -0.21 | 0.21 | -0.55 | 0.33 |
| 8388608 | 57600 | 1 | 9 | 1 | 0xB5 | -0.31 | 0.31 | -0.53 | 0.78 |
| 8388608 | 115200 | 1 | 4 | 8 | 0xEE | -0.75 | 0.74 | -2 | 0.87 |
| 8388608 | 230400 | 1 | 2 | 4 | 0x92 | -1.62 | 1.37 | -3.56 | 2.06 |
| 8388608 | 460800 | 0 | 18 | - | 0x11 | -2 | 3.37 | -5.31 | 5.55 |
| 12000000 | 9600 | 1 | 78 | 2 | 0x0 | 0 | 0 | 0 | 0.04 |

*Illustrazione 18: Configurazioni di Clock consigliate da datasheet con a fianco errori percentuali in ricezione e trasmissione già calcolati(1)*

| BRCLK | Baud Rate | UCOS16 | UCBRx | UCBRFx | UCBRSx | TX Error (%) | | RX Error (%) | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | neg | pos | neg | pos |
| 12000000 | 19200 | 1 | 39 | 1 | 0x0 | 0 | 0 | 0 | 0.16 |
| 12000000 | 38400 | 1 | 19 | 8 | 0x65 | -0.16 | 0.16 | -0.4 | 0.24 |
| 12000000 | 57600 | 1 | 13 | 0 | 0x25 | -0.16 | 0.32 | -0.48 | 0.48 |
| 12000000 | 115200 | 1 | 6 | 8 | 0x20 | -0.48 | 0.64 | -1.04 | 1.04 |
| 12000000 | 230400 | 1 | 3 | 4 | 0x2 | -0.8 | 0.96 | -1.84 | 1.84 |
| 12000000 | 460800 | 1 | 1 | 10 | 0x0 | 0 | 1.76 | 0 | 3.44 |
| 16000000 | 9600 | 1 | 104 | 2 | 0xD6 | -0.04 | 0.02 | -0.09 | 0.03 |
| 16000000 | 19200 | 1 | 52 | 1 | 0x49 | -0.08 | 0.04 | -0.1 | 0.14 |
| 16000000 | 38400 | 1 | 26 | 0 | 0xB6 | -0.08 | 0.16 | -0.28 | 0.2 |
| 16000000 | 57600 | 1 | 17 | 5 | 0xDD | -0.16 | 0.2 | -0.3 | 0.38 |
| 16000000 | 115200 | 1 | 8 | 10 | 0xF7 | -0.32 | 0.32 | -1 | 0.36 |
| 16000000 | 230400 | 1 | 4 | 5 | 0x55 | -0.8 | 0.64 | -1.12 | 1.76 |
| 16000000 | 460800 | 1 | 2 | 2 | 0xBB | -1.44 | 1.28 | -3.92 | 1.68 |
| 16777216 | 9600 | 1 | 109 | 3 | 0xB5 | -0.03 | 0.02 | -0.05 | 0.06 |
| 16777216 | 19200 | 1 | 54 | 9 | 0xEE | -0.06 | 0.06 | -0.11 | 0.13 |
| 16777216 | 38400 | 1 | 27 | 4 | 0xFB | -0.11 | 0.1 | -0.33 | 0 |
| 16777216 | 57600 | 1 | 18 | 3 | 0x44 | -0.16 | 0.15 | -0.2 | 0.45 |
| 16777216 | 115200 | 1 | 9 | 1 | 0xB5 | -0.31 | 0.31 | -0.53 | 0.78 |
| 16777216 | 230400 | 1 | 4 | 8 | 0xEE | -0.75 | 0.74 | -2 | 0.87 |
| 16777216 | 460800 | 1 | 2 | 4 | 0x92 | -1.62 | 1.37 | -3.56 | 2.06 |
| 20000000 | 9600 | 1 | 130 | 3 | 0x25 | -0.02 | 0.03 | 0 | 0.07 |
| 20000000 | 19200 | 1 | 65 | 1 | 0xD6 | -0.06 | 0.03 | -0.1 | 0.1 |
| 20000000 | 38400 | 1 | 32 | 8 | 0xEE | -0.1 | 0.13 | -0.27 | 0.14 |
| 20000000 | 57600 | 1 | 21 | 11 | 0x22 | -0.16 | 0.13 | -0.16 | 0.38 |
| 20000000 | 115200 | 1 | 10 | 13 | 0xAD | -0.29 | 0.26 | -0.45 | 0.66 |
| 20000000 | 230400 | 1 | 5 | 6 | 0xEE | -0.67 | 0.51 | -1.71 | 0.62 |
| 20000000 | 460800 | 1 | 2 | 11 | 0x92 | -1.38 | 0.99 | -1.84 | 2.8 |

*Illustrazione 19: Configurazioni di Clock consigliate da datasheet con a fianco errori percentuali in ricezione e trasmissione già calcolati(2)*

## Utilizzo del modulo eUSCI_A nella modalità UART attraverso LPModes

Il modulo prevede un'attivazione di clock automatica per essere utilizzato con Modalità Low Power.

Nel caso in cui il modulo abbia disattivata la sorgente di clock, perchè stanziante in una modalità Low Power che non prevede l'attivazione del clock necessario, all'accensione del modulo la sorgente di clock relativa viene attivata automaticamente tramite richiesta programmata nella macchina a stati del modulo.

La tempistica per l'attivazione non è sempre la stessa.

Per quanto riguarda richieste di DCO, la tempistica equivale al tempo di risveglio dagli stati Low Power in cui ci si trova al momento dell'avvenuta richiesta da parte del modulo.

Anche per questo motivo l'applicativo ha imposto l'utilizzo del DCO come sorgente di UART e Timers, a discapito di un maggiore dispendio energetico, ma garantendo le tempistiche di progetto.

## 3.5 Timers (Timer_A)

Il timerA è un contatore da 16 bit con 7registri di cattura/comparazione che può supportare anche contemporaneamente. Può inoltre supportare uscite PWM e temporizzazioni ad intervalli.

Le funzionalità e quindi i possibili utilizzi del timer A sono le seguenti:

- timer asincrono da 16 bit con 7registri cattura/comparazione e 4modalità di operatività differenti
- sorgenti di clock selezionabili e configurabili via software
- fino a 7registri di cattura/comparazione
- uscite configurabili come uscite PWM (Pulse Width Modulation)
- Ingressi asincroni e uscite latch
- dispone inoltre di registri di tipo Interrupt Vector per una rapida decodifica di tutti gli interrupt di competenza del Timer_A

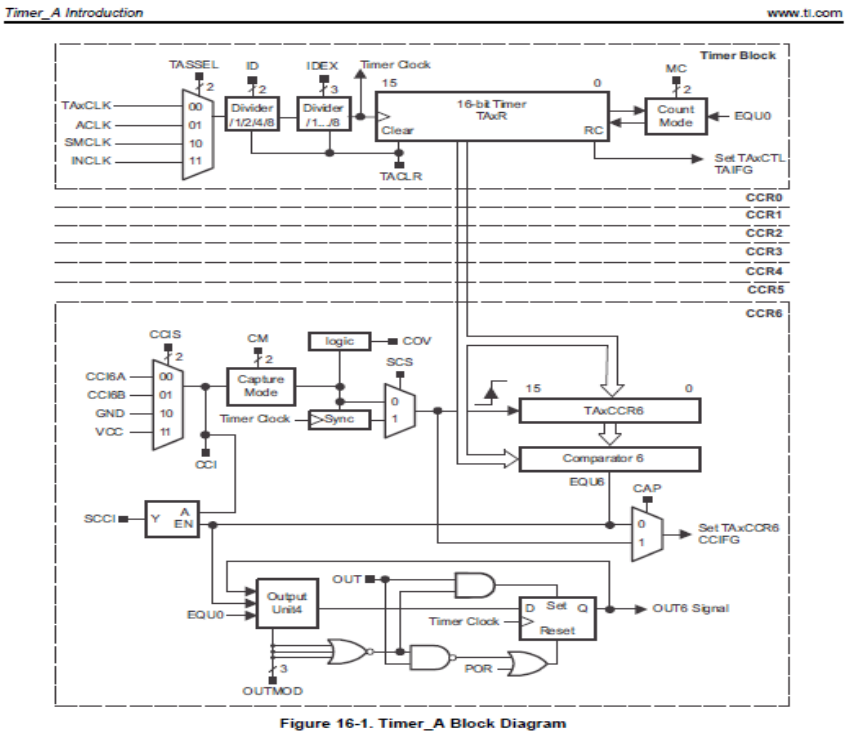

Figure 16-1. Timer_A Block Diagram

*Illustrazione 20: Diagramma a Blocchi del Timer_A*

Nel nostro caso il registro TA1CCR0 viene utilizzato per impostare la fine soglia del PWM e quindi la conseguente frequenza del PWM; mentre tramite il registro TA1CCR1 si è impostato il duty cycle esattamente come la metà della ripetizione dell'onda, quindi al 50%.

I registri inerenti alla configurazione del Timer_A ed utilizzati, sono:

- TaxCTL: Per configurare sorgente di clock del timer, modalità di funzionamento, start conteggio, stop conteggio,...
- TaxCTLLn: registro per configurazione dei registri di cattura/comparazione interni al sistema a blocchi del Timer
- TaxR: registro in cui viene salvato il numero di conteggio del Timer passo dopo passo

- TaxIV: registro in cui sono presenti le celle contenenti i valori dell'Interrupt Vector dedicato al timer_A

Le possibili modalità di funzionamento del Timer_A, configurabili via software, sono:

1. Stop Mode: il timer è configurato, ma spento in attesa di uno start per l'inizio del conteggio
2. Up Mode: il timer si avvia da quando questo bit viene settato nel registro di competenza, e attua un conteggio di tipo Up, fino ad azzerarsi ogni n-1 conteggi.
3. Continuous Mode: il timer si avvia, ma in modalità Continua; ciò significa che inizia a contare fino a n-1 conteggi come nel caso Up, ma una volta raggiunta la soglia di conteggio preimpostata, nel registro di conteggio il timer continua a salvare le n cifre contate da quando è iniziato il conteggio.
4. Up/Down Mode: all'avvio il timer raggiunge gli n-1 conteggi per arrivare alla soglia, ma a quel punto, invece di azzerarsi e ripartire da 0, riparte dalla cifra n-1esima fino a ritornare a 0, per poi ripartire a contare fino a n-1. In questo modo è realizzabile per l'appunto un contatore Up/Down, come si può intuire dal nome.

Nel nostro caso la frequenza del segnale PWM è stata impostata a 100KHz attraverso il registro TACCR0 per soddisfare le specifiche di progetto.

## 3.6 Le porte di GPIO

Le GPIO sono dotate di 8 porte da 8bit l'una, associabili anche come 4porte da 16bit, chiamate convenzionalmente PORTA, PORTB, PORTC, PORTD.

Questa associazione vale incondizionatamente per tutte le porte di GPIO, eccezion fatta per gli interrupt, che mantengono la nomenclatura numerata.

Oltre a queste 4 porte vi è anche un'altra porta da 16bit, non splittabile come le altre in due, che è la porta di JTAG.

Per quanto riguarda le operazioni con byte e word, è possibile modificare i lower bits senza modificare gli upper, e viceversa è possibile anche il contrario; ciò significa appunto, che se modifico dei bit della portA, modificandoli però con la nomenclatura di port1, a meno chè non modifichi anche la port2, quest'ultima non subirà variazioni.

Ci sono 5 registri, che danno la possibilità di configurare via software tutte le porte di GPIO, e sono:

- PxIN: tramite questo registro è possibile caratterizzare i valori logici delle porte dichiarate come Ingressi. Se il bit viene settato a 0, l'ingresso sarà a livello logico basso, e viceversa nel caso contrario.
- PxOUT: tramite questo registro si vanno a configurare i valori logici delle porte dichiarate come Uscite. Se il bit viene settato a 0, l'uscita sarà a livello logico basso, e viceversa nel caso contrario.
- PxDIR: tramite questo registro si configura la vera e propria direzione delle porte, ovvero se il bit associato alla porta viene settato a 1, la porta sarà in direzione Output, mentre nel caso contario, come da default, se viene fatto il clear del bit associato alla porta, o comunque in generale viene settato a 0 quest'ultimo, la porta a cui farà riferimento quel bit sarà settata

come Ingresso.

- PxREN: questo registro serve ad abilitare o disabilitare delle resistenze interne al micro per configurare delle resistenze di pull down o pull up, qual'ora servissero, senza dover mettere mano alcuna via hardware.

Table 10-1. I/O Configuration

Table 10-2. I/O Function Selection

| PxSEL1 | PxSEL0 | I/O Function |
|---|---|---|
| 0 | 0 | General purpose I/O is selected |
| 0 | 1 | Primary module function is selected |
| 1 | 0 | Secondary module function is selected |
| 1 | 1 | Tertiary module function is selected |

*Illustrazione 22: Descrizione del funzionamento del registro PxSEL, deterministico della funzione delle I/O*

- PxSEL: quest'ultimo registro serve a selezionare il modulo o periferica a cui viene associata una porta, ovvero: se non viene toccato, o comunque rimane a 0, la porta a cui ci si riferisce sarà di tipo GPIO, come da default. In caso contrario, verrà settata la porta come funzionalità alternativa, ad esempio come Timer, o come UART in ricezione, oppure come pin esterno a cui collegare un quarzo per esempio.

Nell'"Illustrazione 23" si può vedere come sono disposte nei vari pin del micro (40pin) tutte le periferiche e i moduli.

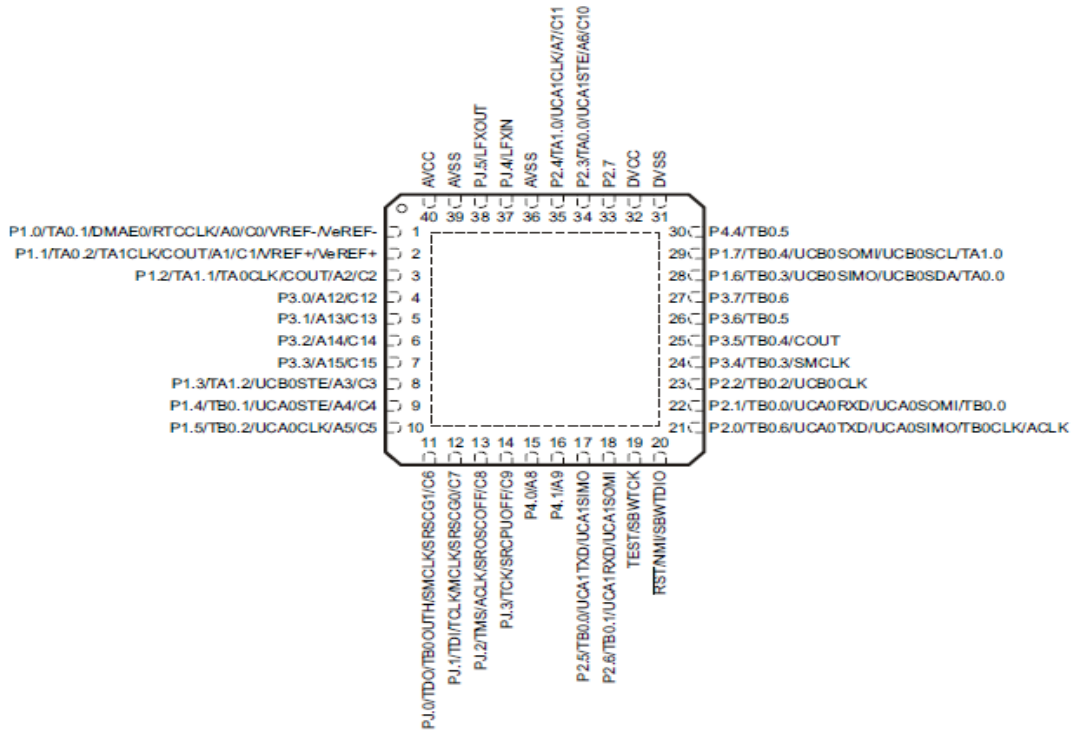


*Illustrazione 23: Diagramma dei Pin dell'MSP430FR5969*

Una nota importante è quella di tenere sempre in considerazione anche tutte le porte inutilizzate ai fini applicativi, in quanto se non venisse fatto, si potrebbe incorrere in consumi maggiori rispetto a quelli aspettati, perchè le porte se programmate in modo da non essere ad un valore logico fissato, possono, se si ritrovano in una condizione di meta stabilità, subire delle fluttuazioni sui pin ed in generale consumare della potenza dinamica dovuta alla loro pseudoaccensione.

Per evitare consumi aggiuntivi in fase di misura infatti, è necessario fissare le porte di GPIO ad un valore logico prestabilito e fissato.
Per questo motivo, nei codici presentati di seguito, verranno configurate tutte le porte, anche quelle inutilizzate, per ovviare a questi problemi.

# 4. Firmware

La programmazione dell'MSP430FR5969 è stata possibile tramite l'utilizzo dell'IDE Code Composer Studio 6.1.2 della Texas Instruments.
Di seguito si può vedere il diagramma di flusso rappresentativo dello schema logico per implementare la procedura di Main.


*Illustrazione 24: Diagramma di flusso della procedura di Main*

Nel Main si sono inizializzati registri di I/O e periferiche,si sono abilitati gli interrupt necessari al risveglio del micro e si è portato il dispositivo in LPM4, la fase di dormienza più bassa per il dispositivo nella quale il wake up da evento è ancora possibile.

## 4.1 INIZIALIZZAZIONE REGISTRI di I/O

Per poter scegliere al meglio le sorgenti di clock da utilizzare, e quindi soddisfare le specifiche di consumo a cui sottostare, si sono effettuati diversi applicativi con MAIN invariato, ad eccezion fatta per la configurazione dei clock del dispositivo.
Inoltre sono state verificate diverse configurazioni di Baud Rate per la ricezione seriale, per provare a stimare una configurazione efficace e coerente con i requisiti.

```
10
11 int main(void)
12 {
13
14
15     WDTCTL = WDTPW | WDTHOLD;                 // Stop WDT
16
17     // Configure GPIO
18
19     // Disable the GPIO power-on default high-impedance mode to activate
20     // previously configured port settings
21     PM5CTL0 &= ~LOCKLPM5;
22
23
24     //DEFINIZIONE DIREZIONE PORTE
25
26     P1DIR |= BIT2;              // p1.2(timerA1) come output,altri input
27     P2DIR &= ~BIT1;             // pin PORTA2 come input:p2.1(rxd),altri input
28     P3DIR = 0x00;               // pin porta3 tutti come input
29     P4DIR = 0x00;               // pin porta4 tutti come input
30     PJDIR = 0x00;               // pin PORTAJ tutti come input
31
32     //ABILITAZIONE RESISTENZE PORTE
33     P1REN = 0x00;               //resistenze disabilitate
34     P2REN = 0x00;               //resistenze disabilitate
35     P3REN = 0x00;               //resistenze disabilitate
36     P4REN = 0x00;               //resistenze disabilitate
37     PJREN = 0x00;               //resistenze disabilitate
38
39     //SETTAGGIO VALORI LOGICI DEFAULT PORTE
40
41     P1OUT = 0x00;           //valore logico fissato a 0
42     P2OUT = 0x00;           //valore logico fissato a 0
43     P3OUT = 0x00;           //valore logico fissato a 0
44     P4OUT = 0x00;           //valore logico fissato a 0
45     PJOUT = 0x00;           //valore logico fissato a 0
```

*Illustrazione 25: Settaggio porte di I/O*

Nell'"Illustrazione 25" sopra, si può vedere la configurazione utilizzata per le porte, al fine di configurare come ingresso il pin RXD dell' UART, come uscita il TimerA1, responsabile del PWM sul pin P1.2 e TimerB0 come ingresso, per il conteggio del Tpwm, da mantenere nei 3ms accordati.
Tutte le porte inutilizzate sono state configurate come input con livello logico basso, configurazione che è risultata essere la più adeguata dal punto di vista dei consumi dei pin inutilizzati.

```
//SELEZIONE PERIFERICHE (UART PER P2.1-->RXD)
P2SEL1 |= BIT1;
P2SEL0 &= ~(BIT1);                                          //(UART) per P2.1(rxd)

//SETTAGGIO PORTA PER TIMERA0
P1SEL0 |= BIT2;
P1SEL1 &= ~(BIT2);                                          //Modulo primario(TIMERA1.1) --> su p1.2

//SETTAGGIO PORTA PER TIMERB0
P2SEL0 |= BIT5;
P2SEL1 &= ~(BIT5);                                  //Modulo primario(TIMERB0.0) --> su p2.5
```

*Illustrazione 26: Settaggio delle periferiche utilizzate come moduli non primari di I/O*

Nell' "Illustrazione 26" si può vedere la configurazione per selezionare le periferiche richieste dall'applicativo nelle porte in cui sono a disposizione, al posto delle I/O di default.
Il registro PxSEL0/1 come detto precedentemente serve a configurare i vari pin come moduli primario/secondario/terziario.
Questi ultimi sono sempre impostati come primari se non viene specificata la configurazione dei registri di selezione, e quindi come I/O.

## 4.2 Configurazione Modulo di Clock

Per il modulo di Clock si è scelto una frequenza che potesse anche garantire divisori tali, da non compromettere il conteggio tramite Timer, quindi identificando sottomultipli più interi possibili, in modo da non generare uno sfasamento temporale significativo.

L'applicazione lavora in tempistiche nell'ordine dei ms, quindi questo dato è abbastanza importante, in quanto essendo appunto le tempistiche molto ridotte, un errore di conteggio può ripercuotersi sul resto dell'applicazione in maniera evidente.

Un'altro parametro caratterizzante la scelta del clock è stata la precisione che potevano garantire i diversi clock; come spiegato nel capitolo precedente infatti, sul dispositivo sono presenti VLO, un clock a bassissimo consumo, ma poco preciso, DCO un oscillatore di clock digitale con altissima precisione, e che può raggiungere frequenze elevate a discapito di un consumo maggiore; è possibile inoltre utilizzare il quarzo interno al micro tramite la sorgente LFXTCLK, che lavora a 32KHz con consumi bassissimi e una discreta precisione.

Sono state considerate queste 3 sorgenti di Clock per andare a configurare il Clock Sorgente, che è stato poi utilizzato per temporizzare il modulo UART, nel quale, tramite divisori programmabili via software, è stato possibile testare diverse soluzioni di Baud Rate, con sorgenti di clock differenti delle quali si è testato l'implementabilità e l'adeguatezza all'applicativo desiderativo.

```
// Configure Clock Setup
CSCTL0 = 0XA500;                                                    //password predefinita Unlock CS reg
CSCTL1 = DCOFSEL_6;                                                 // Set DCO = 1MHz
CSCTL2 = SELA_VLOCLK | SELS__DCOCLK | SELM__DCOCLK;                 // Set ACLK=VLO SMCLK=DCO
CSCTL3 = DIVA__32 | DIVS__1 | DIVM__1 ;                             // Set all dividers M/1;A/1;S/32;
CSCTL0_H = 0;                                                       // Lock CS registers
```

*Illustrazione 27: Configurazione clock sources 1: ACLK = VLO; SMCLK = DCO; MCLK = DCO*

Il registro CSCTL0 serve a sbloccare i registri di configurazione per il Setup del Clock; la prima riga di codice in figura mostra come effettuare l'operazione di Unlock appunto.

CSCTL1 serve a specificare quale range di frequenza viene selezionato, nel caso in figura Frequenza del DCO è impostata a 1MHz.

CSCTL2 serve a impostare quali sorgenti di clock immettere in ACLK(Auxiliary Clock), SMCLK (Sub-Master Clock) e in MCLK (Master Clock); nel caso in figura si è selezionato ACLK = VLO, SMCLK = DCO, MCLK = DCO.

Il registro CSCTL2 serve all'impostazione dei divisori a valle delle sorgenti di clock selezionate col registro precedente; tramite questo registro è possibile quindi lavorare con sottomultipli della frequenza specificata in CSCTL1; Nel caso di SMCLK e MCLK viene impostato divisore a 1, mentre nel caso di ACLK viene impostato a 32, per ridurne il consumo dinamico all'attivazione in quanto inutilizzato.

Questa è la configurazione che si è rivelata essere la migliore, inquanto il DCO presenta un'elevata precisione, dato fondamentale per generare in seguito una Baud Rate stabile e precisa, per una ricezione sicura dell'indirizzo da ricevere tramite WuR.

Il consumo energetico rispetto alle altre configurazioni è risultato limitato o comunque un fattore di rilevanza secondario, sicchè nel caso del VLO come sorgente di SMCLK, la ricezione non è stata neppure possibile, in quanto il Clock si è rivelato essere troppo instabile per lavorare con tempistiche così strette per garantire una Baud Rate abbastanza precisa.

Per quanto riguarda il test con quarzo da 32KHz, una volta effettuata la prova, si è deciso di scartare quest'opzione visto che per attivarsi, essendo il quarzo interno al micro a bassa frequenza e lento in fase di risveglio, non si è rivelato adeguato all'applicativo.

Essendo inoltre l'applicazione orientata al wake up da OFF, il quarzo ad ogni risveglio necessitava di 135 ms in cui risiedeva nello stato di RUN, modalità nella quale il micro dà il massimo consumo di tipo dinamico.


*Illustrazione 28: Analisi tramite Energy Trace ++ della simulazione con configurazione di ACLK = LFXTCLK*

```
// Configure Clock Setup
CSCTL0 = 0XA500;                                  //password predefinita Unlock CS reg
CSCTL1 = DCOFSEL_3;                               // Set DCO = 1MHz
CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;   // Set ACLK=XT1 SMCLK=DCO
CSCTL3 = 0x0030;                                  // Set all dividers Master/1;Aux/1;SubMain/8 = 128K
CSCTL4 &= ~LFXTOFF;                               // Enable LFXT1
  do
  {
    CSCTL5 &= ~LFXTOFFG;                          // Clear XT1 fault flag
    SFRIFG1 &= ~OFIFG;
  } while (SFRIFG1&OFIFG);                        // Test oscillator fault flag
CSCTL0_H = 0;                                     // Lock CS registers
```

*Illustrazione 29: configurazione di Clock ACLK = LFXTCLK*

Nel codice riportato nell'"Illustrazione 29", si può osservare la configurazione che si è scelta di applicare al fine di sfruttare un clock con consumo molto più basso, ma più lento, ovvero il LFXTCLK.

I registri utilizzati per il settaggio e la selezione del range di frequenza, ovvero i divisori e le sorgenti di clock, sono gli stessi spiegati più sopra per la configurazione utilizzata.

Si nota che utilizzando un quarzo come sorgente di clock, è necessario l'utilizzo di una procedura di test sul fault flag dell'oscillatore, indispensabile al fine di assicurare un'oscillazione pura, sinusoidale e un autosostenimento perpetuo.

Per via di questo test necessario effettuato tramite un ciclo do...while, ad ogni accensione il quarzo necessita come fatto vedere nella figura sopra, di un tempo minimo di settling in fase di RUN, che ne ha caratterizzato la non idoneità alle specifiche.

Tramite questa configurazione infatti, ad ogni accensione è necessaria un'attesa di 135 ms solo per l'assestamento del Quarzo stesso,che comporta l'uscita dalle specifiche di tempo riguardanti Tchip = 5ms.

## 4.3 Configurazione Modulo UART

Nella prima configurazione si mostra come configurare il modulo UART affinchè dal clock del VLO venga generata correttamente la Baud Rate richiesta, ovvero di 9600Hz.

```
// Configure USCI_A0 for UART mode(con ACLK come clock INPUT)
UCA0CTLW0 |= UCSWRST;                       // (SET di UCSWRST) --> Put eUSCI in reset;
//inizializzazione dei registri UART con UCSWRST=1
UCA0CTL1 |= UCSSEL__ACLK;                   // CLK = ACLK
UCA0MCTLW |= UCBRS0;
UCA0BR0 = 1;                                //INT+dec = fVLO(9.6*1024)/baudRate(9400) = 1,021 --> INT = 1; dec = 0,0
UCA0BR1 = 0x00;                             // DEFAULT --> INUTILIZZATO
UCA0CTL1 &= ~UCSWRST;                      // release from reset; clear del settaggio dell'UART
UCA0IE |= UCRXIE;                          // Enable USCI_A0 RX interrupt
```

*Illustrazione 30: Configurazione UART con ACLK = VLO = Source Uart = 9600Hz*

Nella seconda configurazione viene fatta la stessa cosa, quindi stessa Baud Rate, ma con il clock in ingresso all'UART proveniente dal SMCLK, configurato per avere in input DCO a 1MHz.

```
// Configure USCI_A0 for UART mode(con SMCLK come clock INPUT)
UCA0CTLW0 |= UCSWRST;                                   // (SET di UCSWRST) --> Put eUSCI in reset;
//inizializzazione dei registri UART con UCSWRST=1
UCA0CTL1 |= UCSSEL__SMCLK;                              // CLK = SMCLK
UCA0MCTLW = 0x2000;                                     // {1000000/(16*9600) - INT[1000000/(16*9600
                                                        // UCBRSx value = 0x20 (See UserGuide)
UCA0BR0 = 6;                                            // 1000000/(16*9600) = 6.510
UCA0BR1 = 0x00;                                         // DEFAULT --> INUTILIZZATO
UCA0MCTLW |= UCOS16 | UCBRF_8;                          // UC016=1(abilitazione oversampling)
//UCBRF=8 settaggio first modulation: come da USER GUIDE
UCA0CTL1 &= ~UCSWRST;                                   // release Reset; termine settaggio UART
UCA0IE = UCRXIE;                                        // Enable USCI_A0 RX interrupt
```

*Illustrazione 31: Configurazione UART con SMCLK = Source Uart = 1MHz*

## 4.4 Configurazione Timers

```
//configurazione timer A1
TA1CCR0 = 10-1;                                  // PWM Period f=1*10^6/10 = 100*10^3 = 100KH
TA1CCTL1 = OUTMOD_7;                             // CCR1 reset/set
TA1CCR1 = 5;                                     // CCR1 PWM duty cycle
TA1CTL = TASSEL__SMCLK | MC__STOP | TACLR;       // SMCLK, stop mode, clear Timer

//Configurazione TimerB0
TB0CCTL0 = CCIE;                                 // Capture/Compare interrupt enabled
TB0CCR0 = 3000;                                  // setto quanti us deve contare(30us e poi e
TB0CTL = TBSSEL__SMCLK | MC__STOP | TBCLR;       // do in input SMCLK=1MHz; stoppo il counter
```

*Illustrazione 32: Configurazione Timer_A1 e Timer_B0*

I Timers programmati come nell' "Illustrazione 32", hanno lo scopo di:

- TimerA1 : creare un segnale PWM sul pin a cui è associato in uscita, al fine di ottenere un'onda quadra di 100KHz dalla frequenza di input di 1MHz, per la commutazione del BackScatter. Sono stati utilizzati i registri TA1CCR0, che serve per settare la soglia di conteggio calcolata come segue:

$F_{PWM} = F_{input} / N = 100KHz = 1MHz / (N)$

$N = 10$

Tramite il regitro TA1CCTL1 come configurato, si imposta un funzionamento del Timer nella modalità Reset/Set, che fa sì chead ogni commutazione di TA1CCR1, il Timer resetti l'uscita a 0 e a fine conteggio la risetti ad 1, in modo da generare un'onda quadra come desiderato.

Il registro TA1CTL serve al settaggio della sorgente di clock del modulo tramite TASSEL__SMCLK e alla configurazione del funzionamento del dispositivo; in questo caso viene stoppato in attesa di essere attivato nel momento appropriato ed eseguire il Clear del registro tramite il settaggio del bit TACLR.

Tramite il registro TA1CCR1 viene effettuato il settaggio della soglia nella quale avviene il Reset, e di conseguenza il duty cicle del segnale PWM generato.

- TimerB0 : una volta attivato, tiene il conteggio dei ms che passano dalla sua attivazione, contemporanea a quella del PWM in uscita. In questo modo si tiene traccia del tempo trascorso e si limita la tempistica di attivazione del PWM, garantendo 3ms necessari da specifiche di progetto.

Tramite il registro TB0CCTL0=CCIE vengono attivati gli interrupt dei registri Capture/Compare, per garantire l'attivazione dell'interrupt a fine conteggio del Timer B0.

Viene poi impostata la fine soglia di conteggio del Timer, ovvero il numero dei us dopo i quali deve avvenire la cattura del tempo, che fa scattare un'interrupt dentro il quale i registri vengono azzerati e il dispositivo ritorna in stato di dormienza (LPM4).

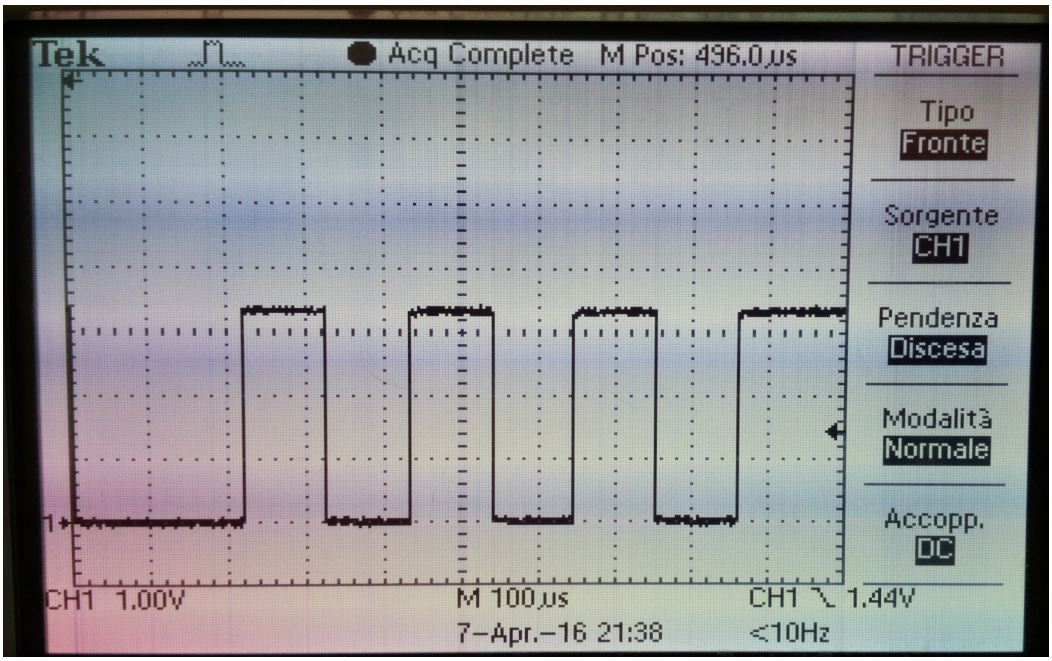


*Illustrazione 33: Tbit dell'onda quadra corrispondente alla frequenza di commutazione prefissata di 100KHz e con duty cycle 50%*

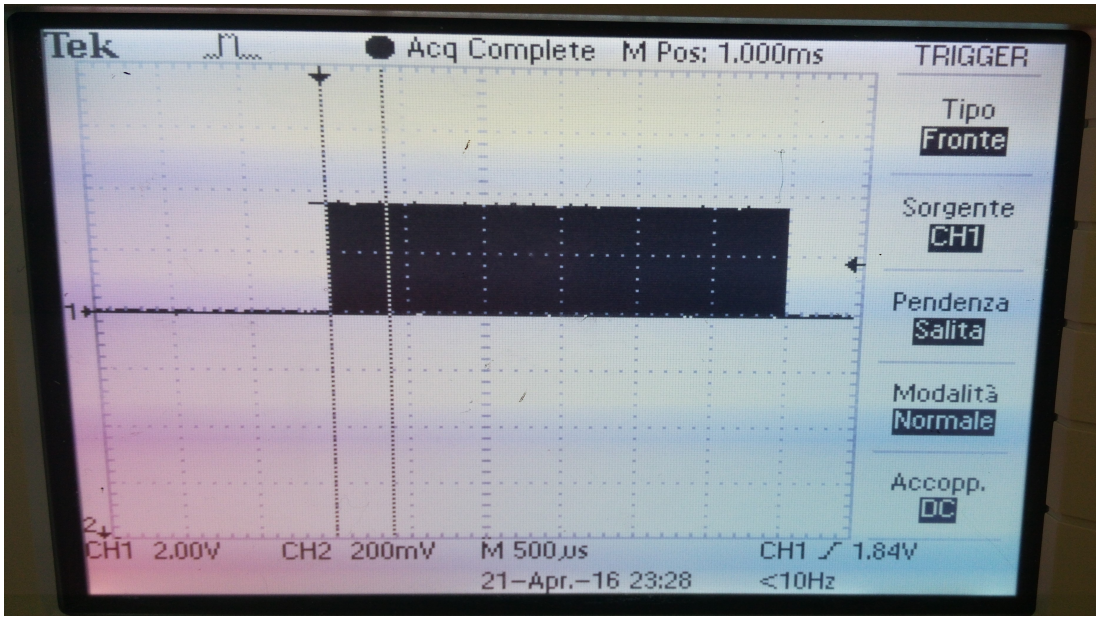


*Illustrazione 34: Onda quadra PWM della durata di 3ms di Tchip come richiesto da specifiche di progetto*

## 4.6 Routine di Interrupt

In questo paragrafo viene mostrato come sono state organizzate le Routine di Interrupt.

Sono prima raffigurati i diagrammi di flusso rappresentativi delle Routines, e poi i veri e propri codici sorgenti spiegati in seguito.

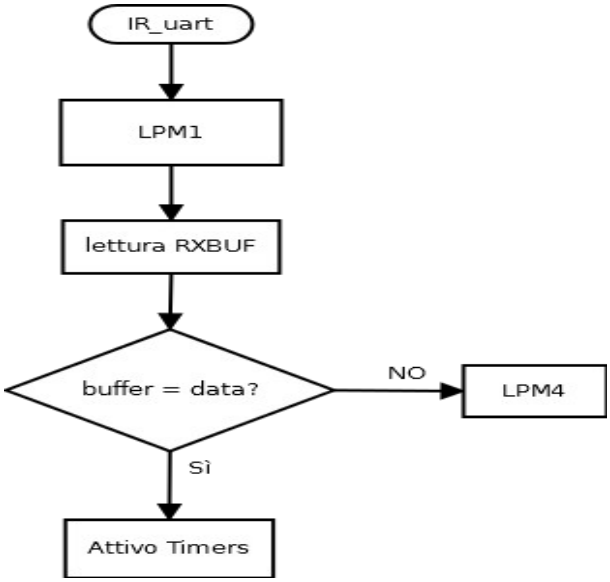


*Illustrazione 35: Diagramma di flusso IR Uart*

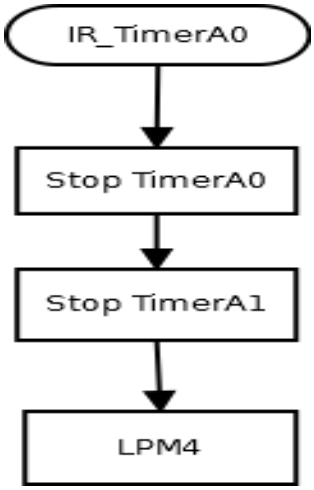


*Illustrazione 36: diagramma di flusso IR timerA0*

```
107 //ISR DELL'UART per la ricezione Uart
108 #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
109 #pragma vector=USCI_A0_VECTOR
110 __interrupt void USCI_A0_ISR(void)
111 #elif defined(__GNUC__)
112 void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
113 #else
114 #error Compiler not supported!
115 #endif
116 {
117   switch(__even_in_range(UCA0IV, USCI_UART_UCTXCPTIFG))
118   {
119     case USCI_NONE: break;
120     case USCI_UART_UCRXIFG:
121         buffer = UCA0RXBUF;
122                 if (buffer == DataRxDesiderato1){
123                     TA1CTL |= MC__UP;               //accendo timer A1
124                     TB0CTL |= MC__CONTINUOUS;       //accendo timer B0
125                     UCA0IE &= ~UCRXIE;              //disabilito ricezione interrupt per l'UART
126
127                     __no_operation();
128                 }else{
129                 }
130         break;
131     case USCI_UART_UCTXIFG: break;
132     case USCI_UART_UCSTTIFG: break;
133     case USCI_UART_UCTXCPTIFG: break;
134   }
135 }
```

*Illustrazione 37: Routine di Interrupt implementata per la ricezione Uart*

Nella routine seguente, come spiegato tramite diagramma di flusso in "Illustrazione 37", viene ricevuto un wake up event della seriale qualora il buffer di ricezione UCA0RXBUF venisse riempito da un byte; a quel punto viene letto il dato seriale su una variabile, che viene poi comparata con l'indirizzo già salvato all'interno del codice.

Qualora le due variabili fossero uguali, vengono avviati i due Timers, uno inerente alla generazione del segnale PWM e l'altro al conteggio del tempo Ttimers; viene inoltre disabilitato l'interrupt seriale in modo che l'arrivo di un altro indirizzo durante l'ISR dei timers non intervenga sulla procedura in corso.

```
137 // ISR TIMERB_0 (riguardante il CCR0)
138 #if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
139 #pragma vector = TIMER0_B0_VECTOR
140 __interrupt void Timer0_B0_ISR (void)
141 #elif defined(__GNUC__)
142 void __attribute__ ((interrupt(TIMER0_B0_VECTOR))) Timer0_B0_ISR (void)
143 #else
144 #error Compiler not supported!
145 #endif
146 {
147     static unsigned int n = 0;
148     n++;
149     if(n > 0){
150         TB0CTL &=  ~MC__CONTINUOUS;
151         TA1CTL &=  ~MC__UP;
152         TA1CTL |= TACLR;
153         TB0CTL |= TBCLR;
154     }
155     //riabilito ricezione interrupt per l'UART
156     UCA0IE |= UCRXIE;
157           }
158
```

*Illustrazione 38: Routine di Interrupt di conteggio del TimerB0*

La routine del Timer B0, raffigurata in "Illustrazione 38", si attiva alla prima fine soglia del TimerB0; una volta attivata arresta i timers, ne azzera il conteggio e riattiva l'interrupt seriale per una nuova ricezione Uart.

Nel nostro caso il Timer B0 è stato impostato con una latenza di 3ms.

## 4.7 Firmware FPGA per Simulazione Trasmettitore UART

In questo paragrafo viene mostrato il codice VHDL della programmazione del simulatore lato trasmissione UART, attraverso un FPGA della famiglia Cyclone II utilizzata anche nel corso degli studi di Elettronica dei Sistemi Digitali.

```
 1  library IEEE;
 2     use IEEE.std_logic_1164.all;
 3     use IEEE.std_logic_arith.all;
 4     use IEEE.std_logic_unsigned.all;
 5
 6  entity UART_TXtesi is
 7
 8     port( CK : in std_logic;
 9           RES : in std_logic;
10
11           SERIALLINE : out std_logic;
12
13           START : in std_logic;
14           D : in std_logic_vector(7 downto 0);
15           READY : out std_logic
16         );
17  end UART_TXtesi;
```

*Illustrazione 39: Caricamento Librerie; instanziamento dell'entità UART_TXtesi; e dichiarazione segnali dell'entity*

```
20  architecture A of UART_TXtesi is
21
22  type my_state is (idle, start_bit, d7, d6, d5, d4, d3 ,d2 ,d1, d0, stop_bit);
23
24  -- costante che definisce quanti cicli di clock dura il tempo di bit
25  constant lastcycle_tbit : integer := 2560;
26
27  signal Q : std_logic_vector(7 downto 0);
28  signal tbit, next_tbit : unsigned(14 downto 0);
29  signal reset_tbit, freeze_input, change_bit : std_logic;
30
31  signal current_state, next_state : my_state;
32
33  attribute syn_keep : boolean;
34  attribute syn_keep of change_bit : signal is true;
35
36
37
```

*Illustrazione 40: definizione Architettura di sistema dell'entità e definizione segnali interni*

```
40      -- contatore del tempo di bit
41      process(CK)
42      begin
43         if CK'event and CK='1' then
44            if RES='1' or reset_tbit='1' then
45               tbit <= conv_unsigned(0,15);
46            else
47               tbit <= next_tbit;
48            end if;
49         end if;
50      end process;
51
52      next_tbit <= conv_unsigned(0,15) when change_bit='1' else tbit+conv_unsigned(1,15);
53
54      -- segnali di campionamento e cambio bit, derivati dal valore del contatore
55      change_bit <= '1' when tbit=conv_unsigned(lastcycle_tbit,15) else '0';  -- si cambia bit (ovviamente) alla fine del tempo di bit
56
57
```

*Illustrazione 41: creazione del contatore del tempo di bit; definizione segnali di campionamento*

```
57
58          -- registro d'ingresso.
59          process(CK)
60          begin
61             if CK'event and CK='1' then
62                if RES='1' then
63                   Q <= conv_std_logic_vector(0,8);
64                elsif freeze_input='1' then
65                   Q <= D;
66                end if;
67             end if;
68          end process;
69
70
71          -- registro della macchina a stati
72          process(CK,RES)
73          begin
74                if RES='1' then
75                   current_state <= idle;
76                elsif CK'event and CK = '1' then
77                   current_state <= next_state;
78                end if;
79          end process;
80
81
```

*Illustrazione 42: Registro per gestione dati e registro della macchina a stati*

```
82          -- parte combinatoria della macchina a stati
83          process(current_state, START, change_bit, Q)
84          begin
85
86             case current_state is
87
88                when idle => reset_tbit <= '1';
89                             freeze_input <= START;
90                             SERIALLINE <= '1';
91                             READY <= '0';
92
93                             if START='1' then next_state <= start_bit;
94                                          else next_state <= idle;
95                             end if;
96
97
98                when start_bit => reset_tbit<= '0';
99                                  freeze_input <= '0';
100                                 SERIALLINE <= '0';
101                                 READY <= '1';
102
103                                 if change_bit='1' then next_state <= d7;
104                                                   else next_state <= start_bit;
105                                 end if;
```

*Illustrazione 43: Macchina a Stati parte 1*

```
107             when d7 =>      reset_tbit<= '0';
108                             freeze_input <= '0';
109                             SERIALLINE <= Q(7);
110                             READY <= '0';
111
112                             if change_bit='1' then next_state <= d6;
113                                              else next_state <= d7;
114                             end if;
115
116             when d6 =>      reset_tbit<= '0';
117                             freeze_input <= '0';
118                             SERIALLINE <= Q(6);
119                             READY <= '0';
120
121                             if change_bit='1' then next_state <= d5;
122                                              else next_state <= d6;
123                             end if;
124
125             when d5 =>      reset_tbit<= '0';
126                             freeze_input <= '0';
127                             SERIALLINE <= Q(5);
128                             READY <= '0';
129
130
131                             if change_bit='1' then next_state <= d4;
132                                              else next_state <= d5;
133                             end if;
```

*Illustrazione 44: Macchina a Stati parte 2*

```
135         when d4 =>      reset_tbit<= '0';
136                         freeze_input <= '0';
137                         SERIALLINE <= Q(4);
138                         READY <= '0';
139
140
141                         if change_bit='1' then next_state <= d3;
142                                          else next_state <= d4;
143                         end if;
144
145         when d3 =>      reset_tbit<= '0';
146                         freeze_input <= '0';
147                         SERIALLINE <= Q(3);
148                         READY <= '0';
149
150                         if change_bit='1' then next_state <= d2;
151                                          else next_state <= d3;
152                         end if;
153
154         when d2 =>      reset_tbit<= '0';
155                         freeze_input <= '0';
156                         SERIALLINE <= Q(2);
157                         READY <= '0';
158
159                         if change_bit='1' then next_state <= d1;
160                                          else next_state <= d2;
161                         end if;
```

*Illustrazione 45: Macchina a Stati parte 3*

```
163             when d1 =>    reset_tbit<= '0';
164                           freeze_input <= '0';
165                           SERIALLINE <= Q(1);
166                           READY <= '0';
167
168                           if change_bit='1' then next_state <= d0;
169                                             else next_state <= d1;
170                           end if;
171
172             when d0 =>    reset_tbit<= '0';
173                           freeze_input <= '0';
174                           SERIALLINE <= Q(0);
175                           READY <= '0';
176
177                           if change_bit='1' then next_state <= stop_bit;
178                                             else next_state <= d0;
179                           end if;
180
181
```

*Illustrazione 46: Macchina a Stati parte 4*

```
182           when stop_bit => reset_tbit <= '0';
183                            freeze_input <= '0';
184                            SERIALLINE <= '1';
185                            READY <= change_bit;
186
187                            if change_bit='1' then next_state <= stop_bit;
188                                              else next_state <= stop_bit;
189                            end if;
190
191
192           when others =>   reset_tbit <= '1';
193                            freeze_input <= '0';
194                            SERIALLINE <= '1';
195                            READY <= '0';
196
197                            next_state <= idle;
198        end case;
199     end process;
200
201
202 end A;
```

*Illustrazione 47: Macchina a Stati parte 5*

Il codice VHDL utilizzato per programmare l'FPGA è servito a realizzare un simulatore di trasmissione UART di una trama da 8 bit.
La trama è impostabile tramite la configurazione hardware degli switch sulla scheda; sono infatti associati tramite il firmware programmato ai bit che vengono impostati su una GPIO collegata con il piedino RXD del modulo Uart dell'evaluation board del micro.
La trama inviata è impostata per avere un Tbit tale da mandare 8 bit con una Baud Rate da 9600Hz, come configurato sul micro.
Nell'"Illustrazione 48" si può notare come la ricezione avvenga correttamente; una volta avviato il micro in LPM4 infatti, la trama da 8bit viene mandata al micro tramite l'attivazione comandata da un pulsante opportunamente configurato sull'FPGA.

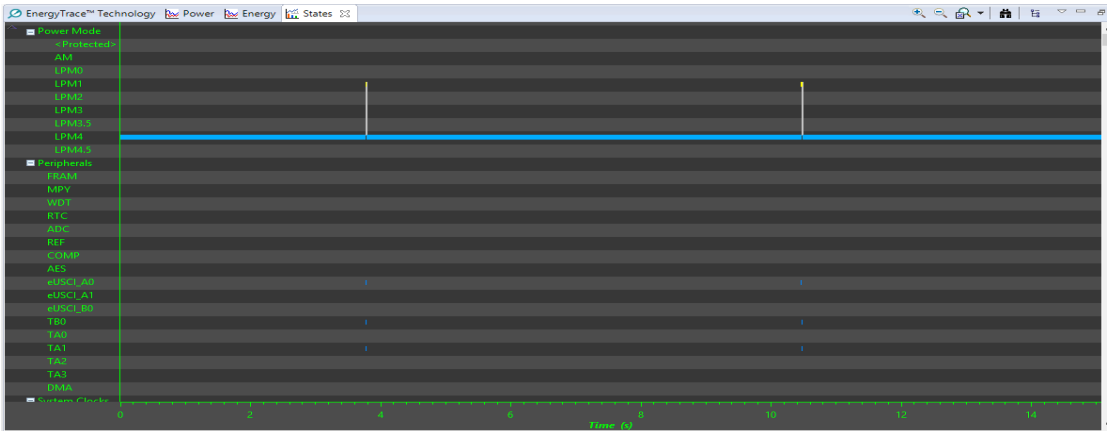

*Illustrazione 48: Ricezione Uart corretta e attivazione Timer corretta*

È stata testata anche la ricezione Uart tramite Uart clock Source = VLO = ACLK;
come indicato precedentemente nella tabella però, gli errori percentuali relativi a quella configurazione di clock, sono risultati fuori specifiche.
Infatti non è stato possibile implementare una ricezione che funzionasse regolarmente.
Da un punto di vista del codice, la trasmissione della parola da 8 bit è stata ottenuta tramite una macchina a stati con 10stati, uno per ogni bit più uno di riposo e uno di stop.

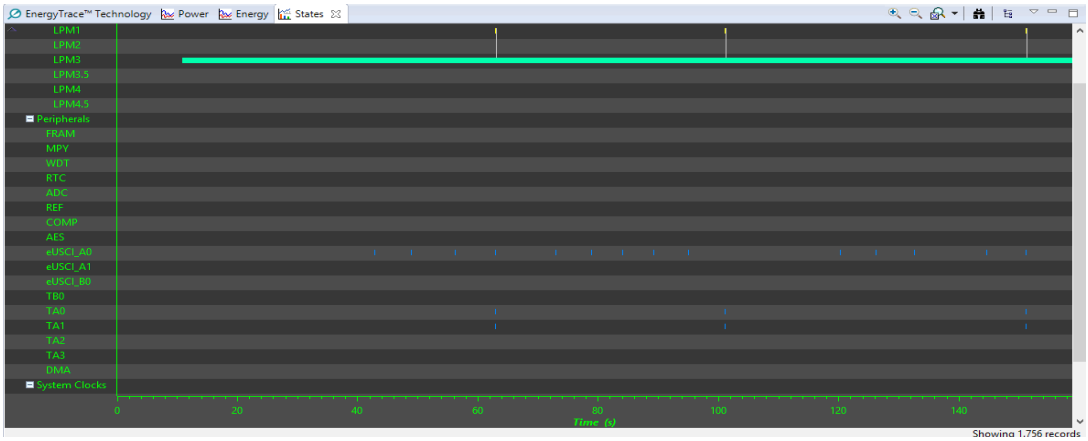

*Illustrazione 49: Ricezione Uart con VLO come clock source*

# 5. Risultati

Di seguito nella tabelle vengono riassunte le stime di consumo di potenza in varie condizioni di utilizzo e con diversi microcontrollori.

### *STIMA DEI CONSUMI MSP430FR5969*

| uC | Eactive (nJ) | Tactive (ns) | Pleak (uW) | Period(T) [ns] | Pavg(nW) | Vdd (V) | request/sec |
|---|---|---|---|---|---|---|---|
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 5,00E+07 | 13871,451 | 3,3 | 20 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 6,25E+007 | 11097,3588 | 3,3 | 16 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 1,00E+08 | 6936,2205 | 3,3 | 10 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 1,43E+008 | 4855,65135 | 3,3 | 7 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 2,00E+08 | 3468,60525 | 3,3 | 5 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 3,33E+008 | 2081,54925 | 3,3 | 3 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 5,00E+008 | 1388,0361 | 3,3 | 2 |
| ***MSP430FR5969*** | 693,528 | 5,00E+006 | 0,99 | 1,00E+09 | 694,51305 | 3,3 | 1 |

*Illustrazione 50: Stima dei parametri energetici primari al variare del numero di ricezioni al secondo nell'MSP430FR5969*

***LEGENDA:***

Eactive = energia spesa per l'applicazione dall'attivazione del micro al ritorno in LPM4 (per tutta la durata in LPM1)

Tactive = Periodo in LPM1

Pleak = potenza consumata in fase di dormienza, ovvero in LPM4

Period = periodo totale in cui il micro rimane acceso

Pavg = potenza media richiesta dal dispositivo per il suo funzionamento

| uC | Eactive (nJ) | Tactive (ns) | Pleak (uW) | Period(T) [ns] | Pavg(nW) | Vdd (V) | request/sec |
|---|---|---|---|---|---|---|---|
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 5,00E+07 | 9270,720 | 1,8 | 20 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 6,25E+007 | 7436,736 | 1,8 | 16 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 1,00E+08 | 4685,76 | 1,8 | 10 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 1,43E+008 | 3310,272 | 1,8 | 7 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 2,00E+08 | 2393,28 | 1,8 | 5 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 3,33E+008 | 1476,288 | 1,8 | 3 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 5,00E+008 | 1017,792 | 1,8 | 2 |
| *Apollo512-KBR* | 459 | 5,00E+006 | 100,8 | 1,00E+09 | 559,296 | 1,8 | 1 |

*Illustrazione 51: Stima dei parametri energetici primari al variare del numero di ricezioni al secondo nell'Apollo 512-KBR*

Da notare che è stato stimato un comportamento relativo al funzionamento tra LPM1 ed LPM4, basandosi sul numero di attivazioni richieste in un secondo dal micro.

Infatti il micro, una volta acceso, deve essere in grado di ricevere almeno 20 indirizzi al secondo, e per questo la stima è stata fatta su un numero di attivazioni crescenti da 1 a 20 attivazioni al secondo.

Sono stati stimati anche i consumi che si sarebbero avuti utilizzando come fase di Sleep le fasi LPM3.5 ed LPM4.5 invece che LPM4, per caratterizzare maggiormente la scelta di progetto della fase LPM4.

*Le relazioni utilizzate per il calcolo della stima sono le seguenti:*

$$E_{ACT} = [(I_{LPM1} + I_{idle} + I_{UART} + I_{TIMERS}) * T_{ON} * V_{DD}]$$

$$P_{LEAK} = (I_{LPM4} + I_{IDLE}) * V_{DD}$$

$$P_{AVG} = \{(E_{ACT} / T) + P_{LEAK} * [1 - (T_{ON} / T)]\}$$

Dalla tabella sopra si nota che ogni secondo, con un massimo di 20 attivazioni al secondo il consumo medio di potenza è di 13,871 uW.

A tal proposito è stato ricercato un micro alternativo che potesse garantire un consumo inferiore, per diminuire ancor più il dispendio energetico dovuto alla logica di controllo implementata appunto tramite microcontrollore.

| *dati da datasheet* | Ilpm1 (uA/MHz) | Ilpm4 (uA) | Ideepsleep (uA) | buck mode | Iidle [uA] | Iuart [uA] | Itimers [uA] |
|---|---|---|---|---|---|---|---|
| *Apollo 512-KBR* | 51 | 56 | 0,12 | on | / | / | / |
| MSP430(3.5) | 35 | 0,3 | 0,25 | / | 0,02 | 5,5 | 3 |
| MSP430(4.5) | 35 | 0,3 | 0,02 | / | 0,02 | 5,5 | 3 |
| | | *lpm4* | *lpm 3.5/4.5* | | | | |

| | Eact (nJ) | Pleak (uW) | Pleak (deepsleep) [uW] | Vdd(V) | Tchip (ms) | Tuart (ms) | Ttimers (ms) |
|---|---|---|---|---|---|---|---|
| *Apollo 512-KBR* | 459 | 100,8 | 0,216 | 1,8 | 5 | 2 | 3 |
| MSP430(3.5) | 673,728 | 0,99 | 0,825 | 3,3 | 5 | 2 | 3 |
| MSP430(4.5) | 673,728 | 0,99 | 0,066 | 3,3 | 5 | 2 | 3 |

*Illustrazione 52: Parametri utilizzati durante la stima dei consumi presi da Datasheet*

Di seguito viene mostrato un confronto tra il funzionamento del micro in LPM4, LPM3.5 e LPM4.5 come modalità di sleep, mantenendo invariata la fase di attivazione, gestita quindi tramite passaggio in stato Low Power LPM1.

Dai confronti nelle varia modalità si nota come il micro non presenti vantaggi nell'utilizzare stati low power meno dispendiosi (di Deep Sleep), rispetto allo stato Low-Power LPM4. Inoltre il wake up da LPM3.5 e LPM4.5 non è garantito.

L'applicativo è per questo motivo stato improntato sulla configurazione discussa più volte nel corso della tesi.

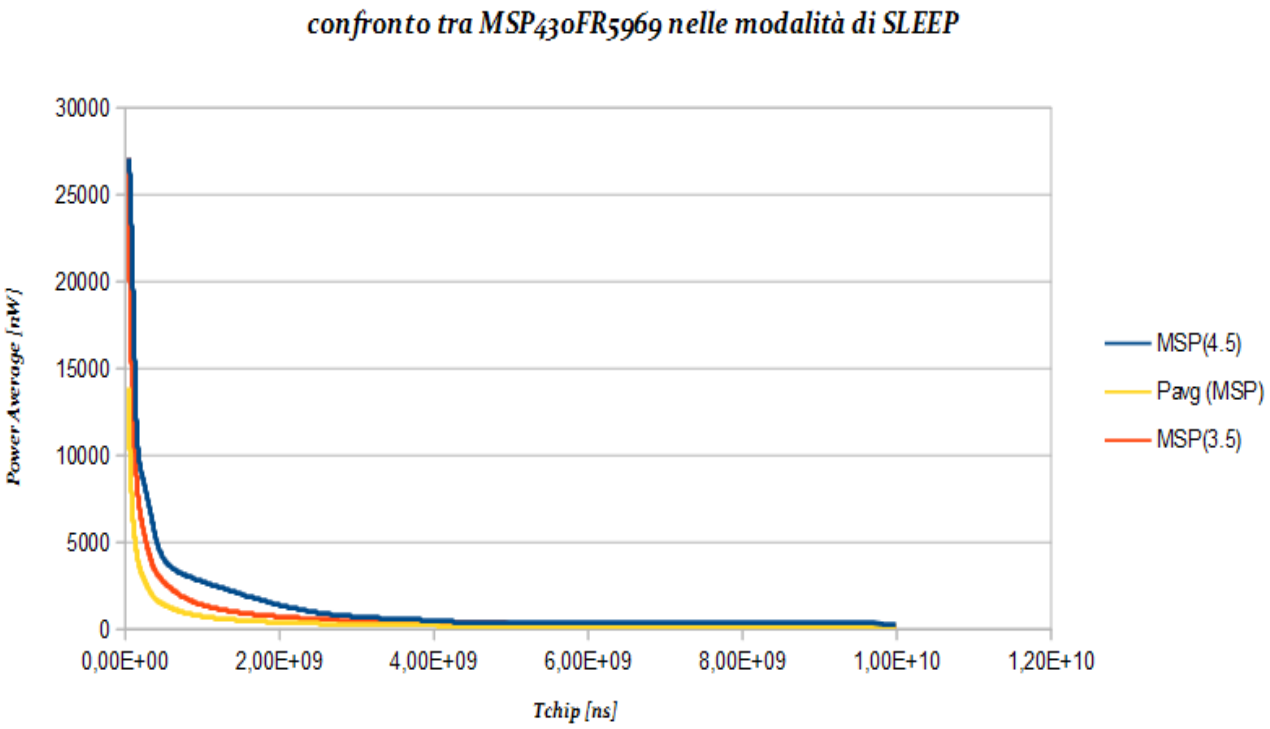


*Illustrazione 53: Grafico di Confronto tra le varie modalità di Sleep del micro*

### *5.1 Apollo KBR-512*

L'Apollo KBR-512 di Ambiq è un microcontrollore a 32 bit, sul quale è stata utilizzata la SPOT's technology di Ambiq, una tecnologia patentata ed innovativa che permette ai circuiti analogici interni al microcontrollore di lavorare a soglie di alimentazione ridotte rispetto agli standard di mercato attuali.

Solo attivando i regolatori DC/DC Buck interni al micro e riducendo la soglia di tensione di alimentazione, questo micro è in grado di vincere dal punto di vista del consumo contro tutti gli altri sul mercato ad oggi.

Inoltre esso è ad oggi il microcontrollore ARM-based con Real Time Calendar (RTC) col più basso consumo di potenza sul mercato.

Per semplicità non si è effettuata una descrizione dettagliata delle periferiche interne al micro interessato; sono presenti però i parametri da datasheet che hanno permesso di effettuare il confronto tra i due micro.

Conseguentemente è quindi stata raffigurata la differenza dal punto di vista dei consumi tra Apollo KBR-512 e MSP430FR5969 e l'analisi mostra un largo vantaggio da parte dell'Apollo KBR-512.

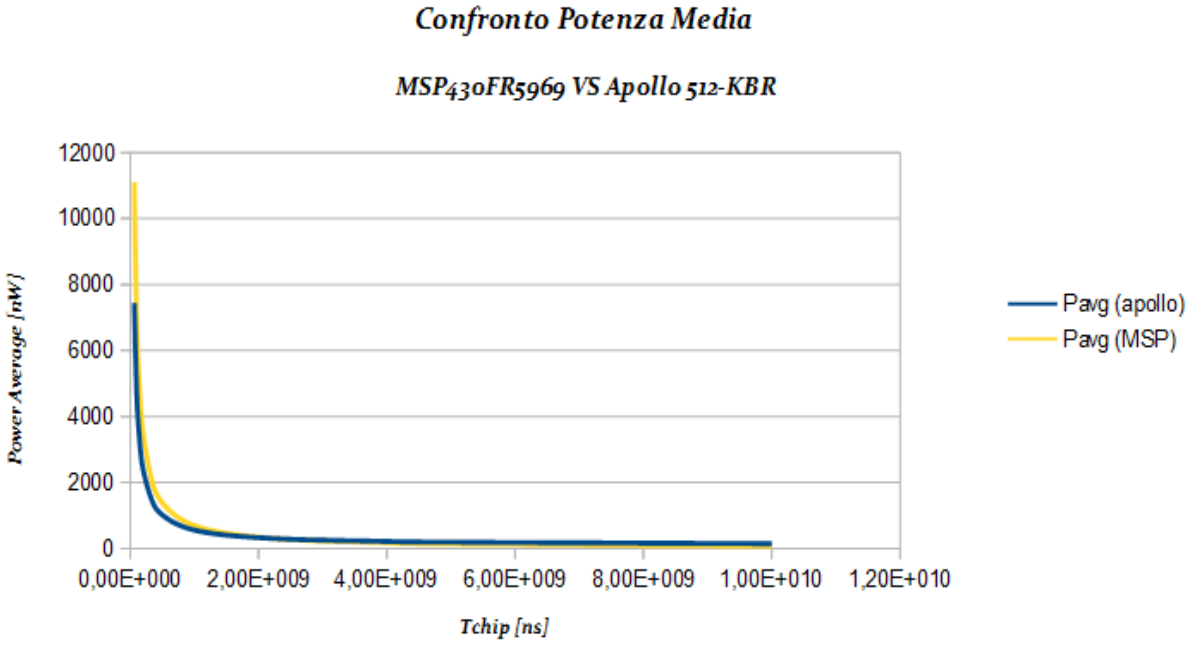


*Illustrazione 54: Confronto in termini energetici fra i due microcontrollori presi in considerazione con attivazioni al secondo che variano da 20 a 1 attivazione al secondo*

Al diminuire di Tchip, calcolato come Ttot / Nattivazioni, e quindi all'aumentare del Numero di Attivazioni del TAG, il consumo energetico dei due microcontrollori aumenta, ma è da evidenziare la riduzione dei consumi utilizzando l'Apollo 512-KBR.

Nell'"Illustrazione 54" mostrata sotto, si può apprezzare ancor più il miglioramento dal punto di vista del consumo energetico, mettendo in mostra l'andamento dei consumi dei due micro calcolato con attivazioni comprese tra 8 e 20 attivazioni al secondo.

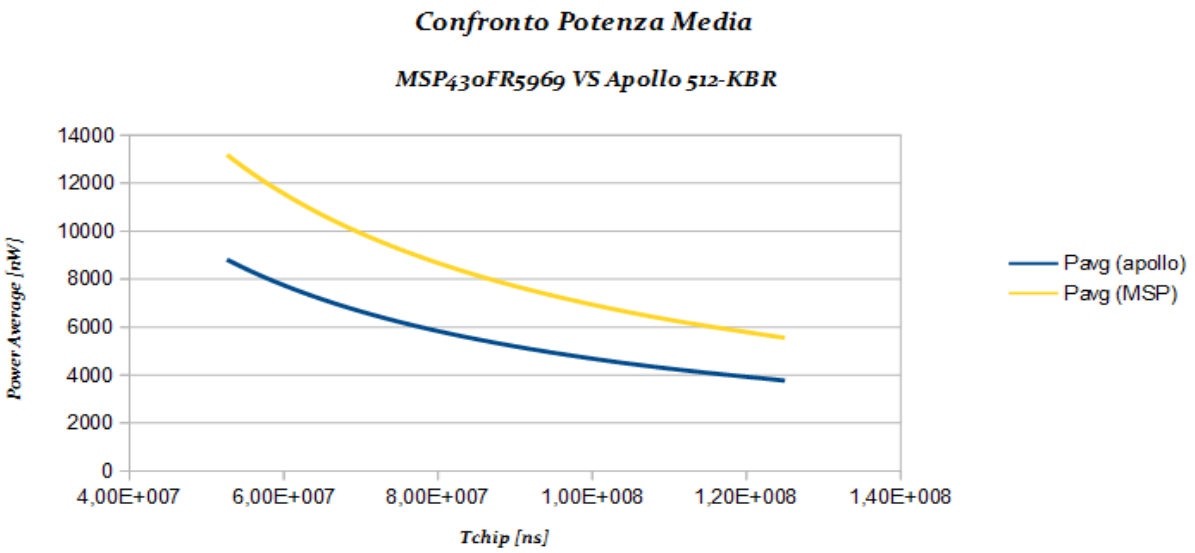


*Illustrazione 55: Confronto in termini energetici fra i due microcontrollori presi in considerazione con attivazioni al secondo che variano da 20 a 8 attivazione al secondo*

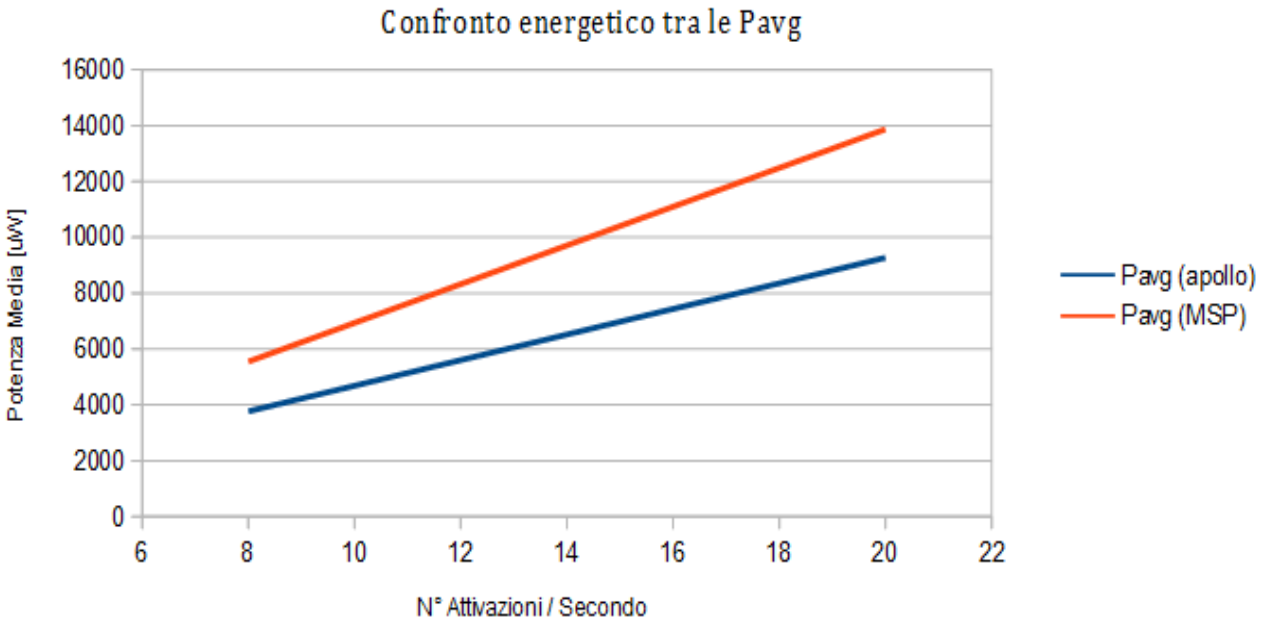


*Illustrazione 56: Confronto energetico (in termini di potenza media stimata) tra i due microcontrollori al variare di Nattivazioni/secondo*

# 6. Sviluppi Futuri

Al fine di una migliore caratterizzazione del progetto è stata fatta una stima di consumi del microcontrollore a livello parametrico, in modo da poterlo confrontare opportunamente con altri microcontrollori come alternativa di progetto, ed eventualmente effettuare un porting del codice su un dispositivo più performante.

La stima rappresenta un funzionamento senza spegnimento del micro, ovvero da quando il microcontrollore si accende, avviene la ricezione dell'indirizzo nel Tchip concordato di 5ms e conseguentemente il microcontrollore genera il segnale PWM per il Ttimer di 3ms, per poi portarsi nuovamente in LPM4 per attendere un nuovo indirizzo.

Come mostrato nel capitolo riguardante i risultati, il microcontrollore Apollo 512-KBR risulta ad oggi essere il migliore candidato a sostituire l'attuale MSP430FR5969 per il nostro applicativo.

# 7. Conclusioni

I requisiti di progetto fissati in ambito di tesi sono stati rispettati.

L'obiettivo era quello di implementare un TAG RFID completo, con particolare attenzione alla scelta dell'architettura con uP (microprocessore) e all'implementazione del firmware dedicato all'elaborazione dei dati ricevuti e al riconoscimento dell'indirizzo identificativo del TAG.

Inoltre, la gestione del commutatore in modalità back-scattering è stata garantita tramite un segnale PWM generato in seguito alla ricezione degli indirizzi TAG; a tal fine si è fissata una frequenza di 100KHz con duty cycle 50%, scelta come compromesso accettabile tra il consumo di potenza e le restrizioni dovute alla comunicazione a radiofrequenza.

Si è stimato il consumo del microcontrollore a livello parametrico, al fine di verificare il dispendio energetico al variare del numero di attivazioni al secondo, con un risultato di 13.87 uW di Potenza Media spesa attivando il chip 20 volte al secondo, con la configurazione che prevede i seguenti settaggi:

- Clock dell'UART = SMCLK = 1MHz
- Clock Timers = SMCLK = 1MHz
- Baud Rate = 9600bps
- Frequenza PWM = 100KHz con duty cycle 50%

I risultati riscontrati sono risultati compatibili con la quantità di energia disponibile nel sistema.

La comunicazione col ricevitore analogico è stata implementata attraverso una comunicazione seriale (UART).

Ai fini della validazione del sistema il ricevitore analogico è stato emulato attraverso un FPGA di cui si è sviluppato il codice specifico.

Confrontando poi il microcontrollore con l'Apollo KBR-512 si è garantita una soluzione alternativa per arrivare ad implementare un applicativo ancora più rapido e a basso consumo.

## *Bibliografia:*


[1]articolo riguardante alcuni approfondimenti sull'RFID
http://www.articolienews.com/approfondimenti/rfid-radio-frequency-identification/

[2] slides sui microcontrollori, Università la Sapienza di Roma
http://wwwusers.di.uniroma1.it/~spenza/pubs/pIPSN2016.pdf

[3] Slides prese dalle lezioni di Elettronica dei Sistemi Digitali, Aldo Romani
http://www-micro.deis.unibo.it/~romani/Dida01/lezioni/microcontrollori_v2.pdf

[4] datasheet MSP430FR5969 della Texas Instruments
http://www.eembc.org/download/down_bench_file.php?bs=2498&hs=319&file=msp430fr5969.pdf

[5] datasheet Apollo 512-KBR della Ambiq
http://www.eembc.org/download/down_bench_file.php?
bs=2501&hs=319&file=Apollo_MCU_Data_SheetDS0010V0p90.pdf

[6] N. Decarli *et al*., "The GRETA architecture for energy efficient radio identification and localization," *RFID Technology (EURFID), 2015 International EURASIP Workshop on*, Rosenheim, 2015, pp. 1-8.

# *Ringraziamenti*

*Oggi mi rendo conto che senza l'aiuto di persone che continuano ogni giorno sempre più a darmi la forza di andare avanti, di credere in me stesso, di camminare sempre e comunque a testa alta e fiero di ciò che sono diventato e di quello che sono, non sarei mai riuscito nell'intento di questa piccola impresa della mia vita.*
*Il raggiungimento di questo mio piccolo traguardo che segnerà per sempre la fine di un capitolo della mia vita e ne ha già avviato l'inizio di un altro, lo devo tutto alla mia famiglia, Mio Padre Nicola, Mia Madre Edi e le Mie splendide sorelle Sara, Ylenia e Cristina, insieme a tutto il resto dei familiari tra cui non mancano certo i miei cognati Paolo e Massimo e i miei fantastici nipotini Pietro e Linda.*
*Vi ringrazio di cuore per tutto quello che avete fatto per me e che farete in futuro.*
*Vi voglio un gran bene.*
*Dei ringraziamenti infiniti vanno inoltre ai compagni di università, di pranzo, di uscite, di Vita.*
*Ringrazio infatti la persona che è stata come un fratello per me da quando ci siamo incontrati, sempre pronto a darmi un sostegno importante e solido sotto ogni circostanza, Grazie Luca.*
*Ringrazio inoltre due fenomeni che ho avuto la fortuna di conoscere pian piano, di apprezzare ed ammirare per la loro tenacia, che mi hanno dato il loro appoggio anch'esso incondizionato e sincero, motivo di stimolo costante e veramente significativo per me: Grazie Mengac e Grazie Casa.*
*Ringrazio inoltre tutti i compagni di università, gli amici e le amiche con cui ho condiviso momenti belli e brutti di questa esperienza.*
*Ringrazio infine il relatore Aldo Romani e il correlatore Matteo Pizzotti, due persone disponibilissime e di una gentilezza unica, senza delle quali non avrei certamente potuto realizzare ciò che ho fatto in questa tesi.*

*Vanno a tutti voi i miei più sentiti ringraziamenti.*
*GRAZIE*