ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA MAGISTRALE DI INGEGNERIA INFORMATICA

TESI DI LAUREA

In

Metodologie Di Progettazione Hardware-Software M

“Progettazione ed implementazione di un Sistema On Chip per applicazioni audio"

CANDIDATO

Mattia Sinigaglia
0000883967

RELATORE

Prof. Davide Rossi

CORRELATORI

Prof. Luca Benini
Prof. Francesco Conti
Angelo Garofalo
Luca Valente

Anno Accademico 2020/2021

Sessione I

Indice

Abbreviazioni

ADC - Analog to Digital Converter

AI - Artificial Intelligence

AXI - Advanced eXtensible Interface

CMOS - Complementary Metal-Oxide Semiconductor

CTS - Clock Tree Synthesis

DAC - Digital to Analog Converter

DRC - Design Rules Constraints

DSP - Digital Signal Processing

DUT - Design Under Test

EDA - Electronic Design Automation

FFT - Fast Fourier Transform

FS - Frequency Sampling

FSM - Finite State Machine

GPIO - General Purpose Input Output

HAL - Hardware Abstraction Layer

HDL - Hardware Description Language

I2S - Inter-IC Sound

IC - Integrated Circuit

IoAuT - Internet of Audio Things

IoMusT - Internet of Musical Things

IoT - Internet of Things

ISA - Instruction Set Architecture

JTAG - Joint Test Action Group

LVS - Layout Versus Schematic

MCU - Micro Controller Unit

MIMD - Multiple Instructions Multiple Data

MOS - Metal-Oxide Semiconductor

MSB - Most Significant Bit

PCB - Printed Circuit Board

PCM - Pulse Code Modulation

PDM - Pulse Density Modulation

PLL - Phase Locked Loop

PULP - Parallel Ultra Low Power

RTL - Register Transfer Level

SDC - Synopsis Design Constraints

SDK - System Development Kit

SPMD - Single Program Multiple Data

TCDM - Tightly Coupled-Data-Memories

TDM - Time Division Multiplexing

ULP - Ultra Low Power

VIP - Verification IP

VR - Virtual Reality

1. Introduzione

L'Internet-of-Things ha favorito una grande crescita del numero di nodi connessi alla rete e ha ulteriormente consentito lo sviluppo di una vasta gamma di applicazioni. Tale incremento ha portato ad un notevole aumento del flusso dati sulla rete che mette l'infrastruttura duramente alla prova. In aggiunta, problemi di privacy e consumi legati alla trasmissione di grandi quantità di dati ha portato alla necessità di filtrare i dati direttamente sui nodi della rete tramite algoritmi di elaborazione e compressione. In questo modo il carico energetico diminuisce notevolmente rispetto al consumo richiesto dalla trasmissione. La possibilità di effettuare filtering ed elaborazione direttamente sui nodi contribuisce ad alleggerire il carico computazionale degli utilizzatori che invece ricevono dei dati pre-processati. Gli attuali dispositivi IoT integrano molteplici sensori e sono costruiti intorno a un microcontrollore che viene utilizzato principalmente per il controllo e l'elaborazione base. I microcontrollori sono molto efficienti per effettuare della computazione che non richiede grandi performance, non sono invece adatti ad eseguire algoritmi complessi che lavorano sull'acquisizione parallela di grossi flussi di dati. Un approccio che consente un aumento delle performance di un microcontrollore ed anche un aumento di efficienza energetica è quello di dotare il microcontrollore di Digital Signal Processing (DSP) che consente di effettuare una elaborazione a livello di segnale. Anche se i DSP consentono di raggiungere alte prestazioni, questi non sono flessibili come i microcontrollori e quindi la loro programmazione non è così semplice. Un'efficienza maggiore può essere ottenuta tramite l'ausilio di acceleratori dedicati che sono integrati direttamente nell'architettura del microcontrollore e che possono essere programmati via software.

Lo scopo de progetto è stato quello di contribuire alla realizzazione di un microcontrollore progettato per applicazioni audio con bassissimi consumi. Il microcontrollore integra un acceleratore FFT che effettua la trasformata di Fourier su diversi segnali audio acquisiti dalla periferica I2S che è una periferica dedicata alla comunicazione con interfacce audio digitali.

Nello specifico, è stato implementato nella periferica I2S il protocollo DSP con TDM per consentire la connessione di molteplici dispositivi sulla stessa linea dati. Il risultato ottenuto è stato quello di riuscire a comunicare contemporaneamente con 16 dispositivi di input e di output fornendo l'elaborazione effettuata dall'acceleratore FFT sui dati acquisiti.

Il microcontrollore, basato su PULP, prende il nome di Echoes come tributo ai Pink Floyd perché specifico per applicazioni audio ed è dotato di un largo set di periferiche che gli consentono di comunicare con il mondo esterno.

L'elaborato è suddiviso in due parti, la prima introduce all'edge processing ed ai protocolli audio digitali, la seconda invece descrive le fasi di integrazione del nuovo protocollo DSP nella periferica I2S in PULP, la progettazione e l'implementazione fisica del chip Echoes

Background

La costante evoluzione tecnologica ha consentito lo sviluppo di sistemi on chip sempre più specifici e performanti. In questa sezione viene percorsa l'evoluzione e l'importanza dell'edge computing per un target applicativo orientato all'audio processing.

2. Edge Computing

Per sistema embedded si intende un qualsiasi dispositivo che include una CPU programmabile ma che non è di per sé general purpose. Questo sistema realizza principalmente una specifica funzione che viene eseguita ciclicamente:

- Acquisizione
- Elaborazione
- Produzione di un feedback

Tutti i sistemi che hanno questo comportamento fanno parte di quello che viene definito cyber physical loop (*figura 1*), cioè acquisiscono dati provenienti dall'ambiente circostante utilizzando sensori e trasduttori che si occupano di convertire i dati in forma digitale per poter essere elaborati.

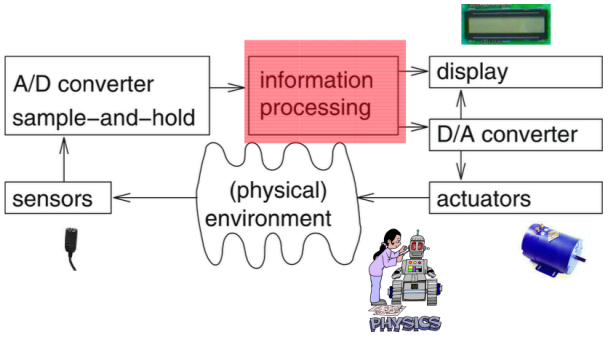


Figura 1: Cyber Physical Loop

Per svolgere l'azione di controllo in modo corretto bisogna tenere in considerazione diversi fattori definiti come vincoli non funzionali:

Tempo: Vincolo temporale nell'elaborazione e produzione di un feedback

Consumo: Quantitativo di energia richiesta per svolgere la sua funzione

Robustezza: Resistenza all'interferenza del mondo esterno

Ad oggi, quando si parla di embedded system si fa riferimento all'Internet of Things (IoT) che però differisce dai sistemi embedded tradizionali in quanto, lavorano sì secondo il cyber physical loop, ma hanno anche la possibilità di comunicare tra di loro. La particolarità dei sistemi IoT è quindi di essere un embedded system in cui l'interazione con altri sistemi gioca un ruolo fondamentale.

Embedded System + Networking = Internet of Things

Lo scaling tecnologico dei transistor e la costante evoluzione delle tecnologie radio hanno permesso l'esplosione dei sistemi embedded sul mercato. L'evoluzione degli standard di comunicazione ha effetti non solo sulla quantità di dati che si riescono a trasmettere in un certo range, ma anche sul consumo energetico richiesto per il loro funzionamento.

Un altro fattore che ha contribuito all'espansione dei sistemi embedded è l'intelligenza artificiale (IA) che con il tempo è migrata dal Cloud verso i dispositivi sia per ragioni di mancata efficienza di comunicazione sia ragioni di privacy. Il vantaggio di questa migrazione (Processing su Cloud → Processing su Edge) è dato dal fatto che i dati acquisiti dai sensori subiscono un pre-processing direttamente nel nodo e solo dopo vengono inviati al cloud, trasmettendo quindi solo informazioni di più alto livello e con maggiore contenuto informativo. Questi dispositivi, che integrano capacità di IA negli embedded system, vengono definiti extreme edge AI oppure Tiny ML.

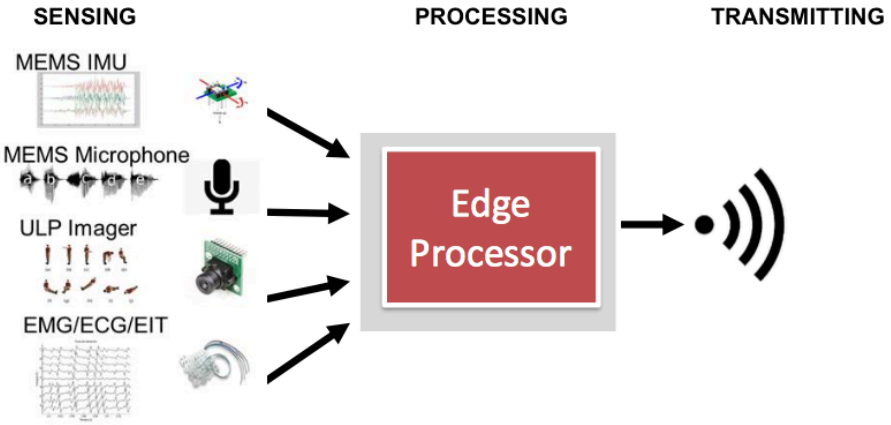


Figura 2: Edge Processing

La sfida a cui sono sottoposti i progettisti di sistemi embedded è quella di raggiungere un giusto compromesso tra flessibilità ed efficienza in termini di tempo di esecuzione, peso, costi e consumi.

Nei sistemi digitali il consumo di potenza è suddiviso in tre classi:

- Dynamic Power
- Leakage Power
- Short-circuit power

Nella progettazione di Micro Controller Unit (MCU) si prendono principalmente in considerazione Dynamic power P_{dyn} e Leakage power P_{leak} definite rispettivamente come:

$$\left.\begin{aligned} P_{dyn} &= ACV^2 f \\ P_{leak} &= VI_{leak} \end{aligned}\right\} \; P_{tot} = P_{dyn} + P_{leak}$$

Nella potenza dinamica P_{dyn}, la componente *A* rappresenta la switching activity dei gates, *C* rappresenta la capacità totale dei gate, *V* è la tensione operativa e *f* è la frequenza di clock (che si ricorda essere dipendente da *V*). La potenza di leakage P_{leak} è la potenza consumata dai Metal-Oxide Semiconductor (MOS) in stato di OFF a causa della corrente di polarizzazione inversa; questa diventa dominante a basso voltaggio e a basse frequenze mentre cresce esponenzialmente con l'aumentare della temperatura e con un voltage threshold basso (transistor più veloci consumano maggiore corrente di leakage). Ciò induce delle limitazioni sulla riduzione della potenza totale e in particolar modo pone dei limiti alla potenza minima e alla massima efficienza del sistema. Nella tecnologia Complementary Metal-Oxide Semiconductor (CMOS) la potenza dinamica è dominante in stato ACTIVE e dipende in modo quadratico dalla tensione *V*.

Per gli sviluppatori embedded la riduzione della potenza viene ottenuta tramite la riduzione del voltaggio operativo *V*, della frequenza *f* e della switching activity A ottenuta spegnendo moduli non necessari.

La saturazione della legge di Moore (*"La complessità di un circuito digitale, misurata tramite il numero di transistor per chip, raddoppia ogni 18 mesi (e quadruplica ogni 3 anni)"* compromette o comunque rallenta la produzione e la disponibilità di architetture sempre più performanti. Sulla base di questa considerazione è necessario adottare diverse soluzioni per ottenere l'efficienza di cui si ha bisogno in termini di consumi e performance.

La riduzione del single transistor ha consentito l'aumento delle prestazioni in termini di complessità dei circuiti integrati e in termini di velocità di esecuzione, ma se si aumenta la frequenza di elaborazione senza diminuire il power supply si ha come effetto che il power consumption aumenta linearmente ($P_{dyn} = ACV^2 f$). Di fatto, riducendo la tecnologia (per integrare molteplici funzionalità nello stesso core) e aumentando la frequenza operativa f per andare più veloci si deve dissipare molta più potenza in un'area sempre più piccola. Questo comporta un aumento vertiginoso della power density che giunge ad un punto in cui è impossibile dissipare la potenza prodotta perché il power consumption è ormai troppo elevato rispetto alla superficie di silicio. Il problema principale è dovuto al power wall che impedisce la riduzione del voltage supply in quanto porterebbe ad un decadimento delle prestazioni (V ed f sono correlate).

Come evidenziato nella *figura 3*, l'aumento delle performance indotto dall'aumento della frequenza operativa risulta inferiore rispetto al conseguente aumento del power consumption. Se si riduce la frequenza operativa invece si ha sia un aumento di performance sia una diminuzione del power consumption.

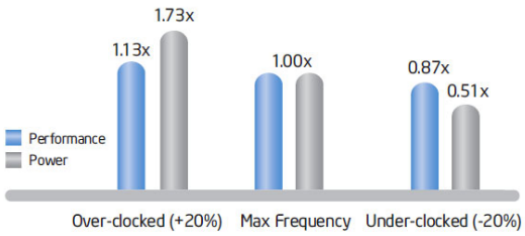


Figura 3: Confronto performance e power consumption su Single-Core

Sulla base delle osservazioni evidenziate, un modo per ottenere alte prestazioni è quello di utilizzare un approccio multicore: invece di aumentare la frequenza di un single core si rende quest'ultimo power efficient per poi integrare molteplici core che eseguono insieme.

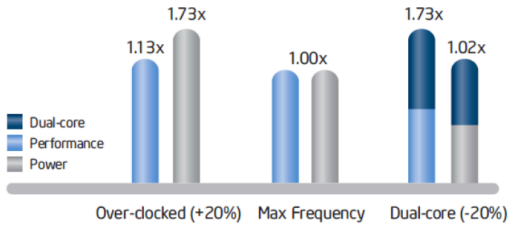


Figura 4: Confronto performance e power consumption su Multi-Core

Nel caso di un dual core con 2 single core power efficient si ottiene un raddoppio delle prestazioni, e se si confrontano le prestazioni del single core con overclok e il multicore con underclock si evince che il multicore oltre ad avere performance operative maggiori ha anche un power consumption inferiore che lo rende migliore in tutti gli aspetti.

Per salvaguardare e migliorare il consumo energetico, i microcontrollori sono dotati di un'architettura capace di ridurre la frequenza del clock, spegnere moduli non necessari e/o cambiare modalità operativa di power. Per fare tutto questo sono dotati di diverse power mode che

differiscono per ogni produttore (RUN, LOW POWER, SLEEP, STANDBY, STOP, …) alle quali sono associati livelli di tensione *V* diversi.

Le applicazioni embedded che usano microcontrollori sono in generale legate al mondo fisico ed effettuano operazioni di monitoraggio e/o attuazione, ciò implica che le applicazioni seguono schemi con diverse fasi.

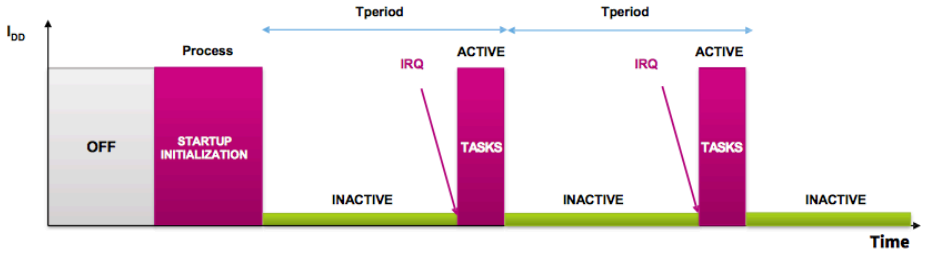


Figura 5: Fasi applicazione MCU

Principalmente ci sono due modi in cui le applicazioni basate su MCU sono eseguite:

- Duty cycling

  Alcuni task devono essere eseguiti in modo periodico, perciò si dedica una frazione di tempo al task in cui il microcontrollore è in RUN e il restante del tempo è in SLEEP.

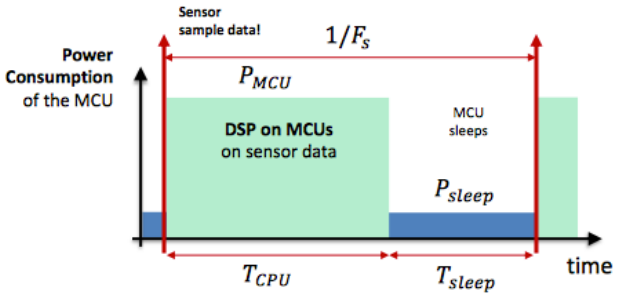


Figura 6: Duty Cycling

- Interrupt

  Il microcontrollore è quasi sempre in SLEEP e viene risvegliato ad intervalli irregolari a causa di un evento esterno o interno

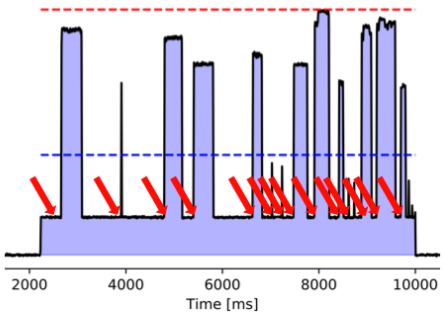


Figura 7: Interrupt

Spendendo meno tempo in active mode si ha un minor consumo energetico.

L'efficienza energetica di ogni dispositivo digitale è caratterizzata dalla combinazione delle modalità operative che influiscono direttamente su P_{dyn} e P_{leak}. Se mettiamo in relazione le due grandezze come in *figura 8* e consideriamo un solo core in tensione nominale 1.2V notiamo che si riesce a

raggiungere relativamente un'alta frequenza operativa, ma l'efficienza energetica è limitata perché la P_{dyn} è dominante. Se si diminuisce la tensione nominale si ottiene un miglioramento dell'efficienza energetica ma un consistente peggioramento delle performance perché diventa dominante la P_{leak}. Supponendo di avere 10 core, dal punto di vista delle performance è possibile notare che le curve aumentano di un fattore 10. Quel che accade è che operando in low voltage si ottiene un incremento di performance rispetto al caso single core e inoltre si ha che l'efficienza energetica rimane alta. Per questa ragione si usa la parallelizzazione hardware, che con un uso attento delle metodologie di parallelizzazione del software si riescono ad ottenere notevoli performance.

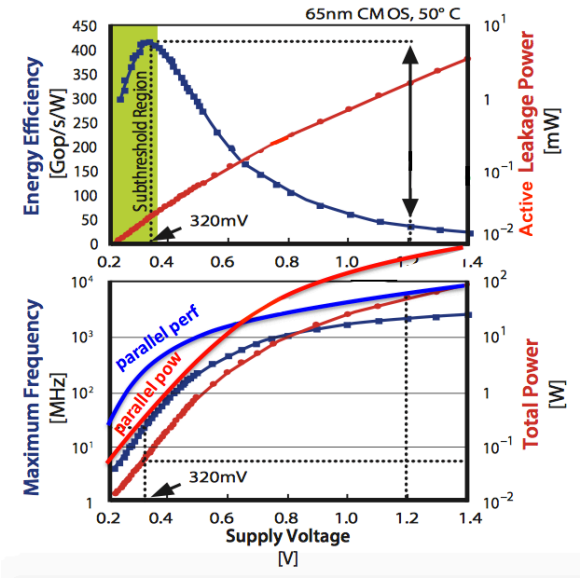


Figura 8: Dynamic Power e Leakage Power

Per un'applicazione è possibile utilizzare differenti tipologie di architetture in relazione al consumo e al livello di performance che ne conseguono: più è specifica l'architettura per compiere una particolare funzione e più ne aumenta l'efficienza energetica. L'adozione di un'architettura flessibile implica una minore efficienza energetica e allo stesso tempo una architettura specifica implica un costo maggiore.

Lo scopo è quindi quello di usare architetture adatte alle funzionalità di cui si hanno bisogno ma che abbiano un minimo di flessibilità, per questa ragione i Sistemi on Chip sono costituiti da moduli interni progettati per compiere funzioni specifiche.

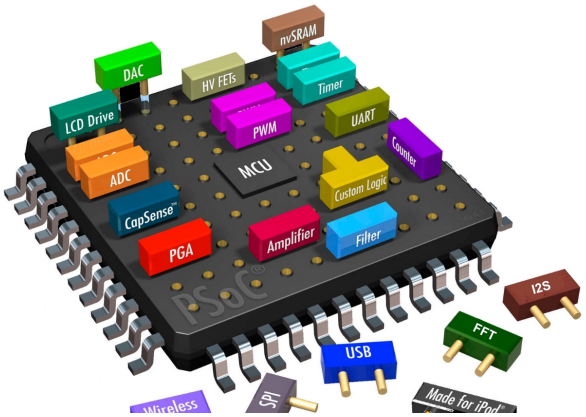


Figura 9: System On Chip

2.1 Applicazioni audio in ambito IoT

L'IoT, come abbiamo visto, offre la possibilità di mettere in comunicazione una vasta gamma di dispositivi. Inizialmente l'IoT aveva come obiettivo il controllo da remoto di diverse tipologie di dispositivi e solo negli ultimi anni ha iniziato a diffondersi rapidamente anche nel dominio della musica, dell'audio e dei contenuti multimediali.

Come è definito in [1], in questo dominio applicativo si definiscono due categorie di IoT:

- Internet of Musical Things (IoMusT)
- Internet of Audio Things (IoAuT)

IoMusT si occupa principalmente di applicazioni musicali come per esempio nelle produzioni musicali e nei concerti: la caratteristica principale è che gli utilizzatori finali sono le persone e quindi lo sviluppo di applicazioni IoMusT richiede un'elevata qualità del servizio in quanto l'esperienza utente si basa sulla qualità audio che deve essere costituita da perdite e latenze impercettibili.

IoAuT invece, si focalizza sul più generale campo delle applicazioni audio e si riferisce alle reti di dispositivi informatici incorporati in oggetti fisici (Audio Things) dedicati alla produzione, ricezione, analisi e comprensione dell'audio in ambienti distribuiti. Le Audio Things, come i nodi delle reti di sensori acustici wireless, sono collegati da un'infrastruttura che permette la comunicazione multidirezionale, sia a livello locale che a distanza. Solitamente è usato per applicazioni industriali e servizi e a differenza di IoMusT, i dati audio acquisiti dalle reti IoAuT non sono necessariamente consumati dalle persone, ma piuttosto dalle macchine per misurare e analizzare per esempio rumori ambientali nell'ambito dell'ecoacustica o per il monitoraggio del traffico, nel settore agricolo per il rilevamento del taglio illegale di alberi in aree boschive difficili da raggiungere oppure per acquisire dati audio in diversi punti dello spazio per studiare e migliorare la navigazione del campo sonoro in applicazioni di Virtual Reality (VR).

Dal punto di vista tecnologico le componenti principali di un sistema IoAuT sono:

- Audio Things

  Un Audio Things è un dispositivo capace di rilevare, acquisire, elaborare e scambiare informazioni legate all'audio e quindi agisce da mittente o destinatario e che può essere utilizzato per produrre contenuti audio o per analizzare fenomeni associati ad eventi uditivi.

- Connettività

  L'infrastruttura di connessione (locale o remota) tra dispositivi supporta una comunicazione bidirezionale cablata e wireless. L'interconnessione avviene con tecnologie hardware e software che regolano la comunicazione tramite protocolli di rete.

- Applicazione

  Sulla connettività e il data processing possono essere costruiti vari tipi di applicazioni e servizi, rivolti a utenti diversi in base allo scopo dell'Audio Thing.

In questa tipologia di sistemi embedded è evidente che i dispositivi che assumono un ruolo fondamentale nell'architettura sono i microfoni che effettuano l'operazione di sensing e dei microcontrollori che ricevono le informazioni, le elaborano e le inviano al successivo nodo AuT tramite i protocolli di comunicazione di cui sono dotati.

2.2 Protocolli interfacce audio digitali

Come è ampiamente descritto in [2], l'aumentare dello scaling tecnologico e il conseguente aumento della densità dei circuiti digitali rende sempre più difficile l'integrazione di circuiti analogici ad alte prestazioni nello stesso pezzo di silicio. Per ovviare a questo problema, i progettisti di sistemi audio stanno spostando le porzioni analogiche della catena del segnale audio verso i trasduttori di ingresso/uscita collegando digitalmente tutto quello che si trova nel mezzo. Questo fa in modo che le interfacce digitali tra i circuiti integrati assumono un ruolo fondamentale: i Digital Signal Processing (DSP) hanno sempre avuto connessioni digitali ed ora le stesse interfacce di comunicazione vengono incluse anche nei trasduttori e negli amplificatori che solitamente avevano solo interfacce di tipo analogico.

La catena di un segnale audio tradizionale ha connessioni di segnale analogico tra microfoni, preamplificatori, Analog to Digital Converter (ADC), Digital to Analog Converter (DAC), amplificatori di uscita e altoparlanti.

La *figura 10* mostra un esempio di una catena di segnale audio tradizionale.

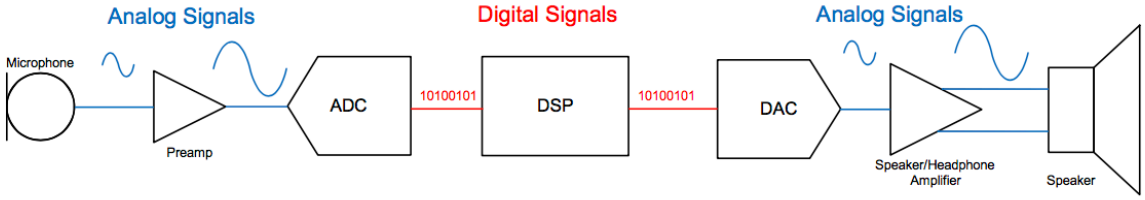


Figura 10: Catena segnale audio tradizionale

I progettisti di circuiti audio integrati stanno integrando ADC, DAC e modulatori direttamente nei trasduttori situati alle estremità della catena del segnale, eliminando così la necessità di inviare qualsiasi segnale audio analogico sul Printed Circuit Board (PCB), e riducendo il numero di dispositivi nella catena del segnale.

La *figura 11* mostra un esempio di una catena di segnale audio completamente digitale.

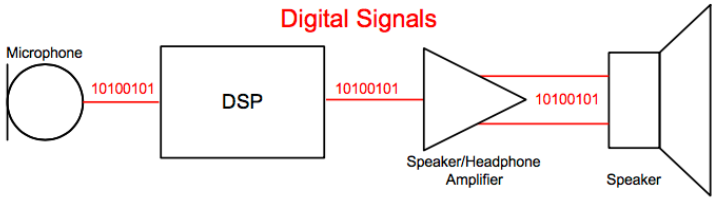


Figura 11: Catena segnale audio completamente digitale

Ci sono diversi standard per la trasmissione di un segnale audio digitale, tipicamente per circuiti integrati sullo stesso PCB si fa riferimento a Inter-IC Sound (I2S), Pulse Density Modulation (PDM), PCM (Pulse Code Modulation) o anche denominato DSP da NXP Semiconductors (differisce dal Digital Signal Processing: per non creare confusione da ora in avanti si farà riferimento a questo protocollo come DSP mode) e Time Division Multiplexing (TDM).

Per interconnettere PCB diversi tramite cablaggio esterno si fa riferimento agli standard S/PDIF e Ethernet AVB.

Il protocollo I2S è il formato audio digitale più utilizzato tra circuiti integrati, la maggior parte degli ADC, DAC e alcuni Micro Controller Units (MCUs) includono interfacce I2S. Questo protocollo trasferisce segnali audio modulati con Pulse Code Modulation (PCM).

Un’interfaccia I2S utilizza tre linee di segnale per il trasferimento dei dati:

- Word Select (WS)
- Serial Clock (SCK)
- Serial Data (SD)

I segnali WS e SCK possono essere generati dall'Integrated Circuit (IC) di ricezione, dall'IC di trasmissione o anche da un IC master separato a seconda dell'architettura. Un IC con una porta I2S spesso può essere configurato per agire in modalità master o slave. Questo standard non ha una topologia di interconnessione definita ma può assumere diverse forme a seconda delle necessità come mostrato in *figura 12*.

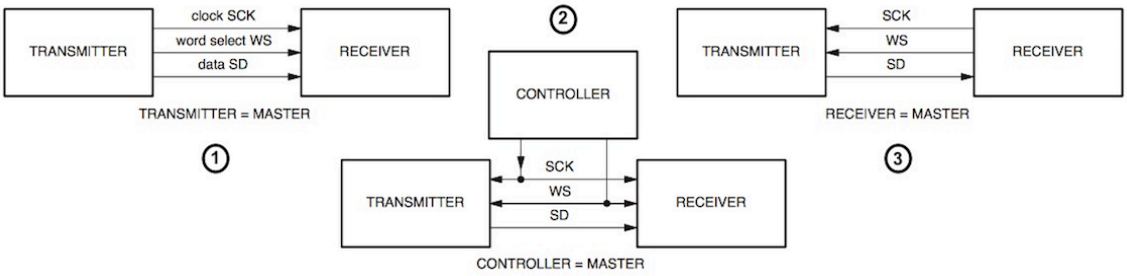


Figura 12: Possibili interconnessioni tra dispositivi con interfaccia I2S

Sebbene il formato I2S sia il più usato, esistono altre varianti di questa configurazione a 3 fili:

- Left Justified
- Right Justified
- DSP

Un altro formato che differisce da I2S è quello PDM, usato principalmente nelle applicazioni audio portatili, come smartphones, computer o tablet. Questo perché oltre a consentire una realizzazione di dispositivi con dimensioni limitate, ha il vantaggio che i segnali PDM non subiscono l’effetto crosstalk così come lo subiscono i segnali analogici quindi possono essere instradati intorno a circuiti rumorosi. A differenza di I2S, l’interfaccia PDM usa solo 2 linee di segnale, una per i dati e una per il segnale di sincronismo CLK che è tipicamente compreso tra 1MHz e 3,2MHz. Il clock generato dal sistema può essere usato da due sorgenti distinte che lavorano su fronti opposti come in *figura 13*. Un'architettura basata su PDM differisce ulteriormente da I2S e TDM in quanto il filtro di decimazione è nel circuito integrato di chi riceve, piuttosto che in quello di chi trasmette. L'uscita della sorgente è un dato modulato grezzo ad alta frequenza di campionamento, come l'uscita di un modulatore Sigma- Delta, piuttosto che un dato decimato, come in I2S. Un'architettura basata su PDM riduce la complessità nel dispositivo sorgente, e spesso fa uso di filtri di decimazione che sono già presenti negli ADC di un codec: questi filtri di decimazione possono essere implementati in modo più efficiente nelle geometrie di silicio più fini usate per un codec o un processore, rispetto che in quello che è usato per gli IC dei microfoni.

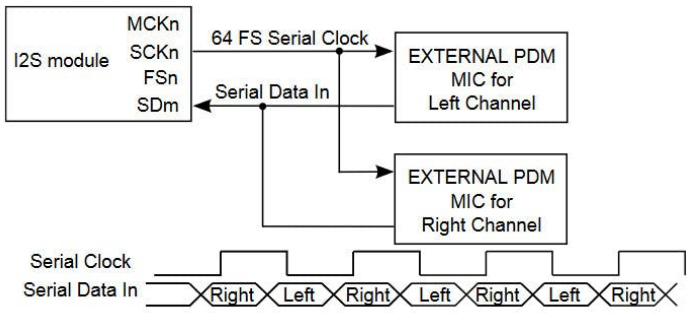


Figura 13 Interconnessione microfoni PDM

2.2.1 I2S

Il protocollo I2S è un'interfaccia seriale standard usata per interconnettere dispositivi audio digitali tramite l'utilizzo di 3 segnali principali: WS, SCK e SD. Un trasferimento dati I2S può includere uno o due canali (Sinistro/Destro) per segnali audio stereo.

Il segnale Word Select è utilizzato per selezionare il canale della sorgente ed ha una Frequency Sampling (FS) configurabile: quando il segnale WS= 0 la sorgente trasmette il canale sinistro, mentre quando WS=1 la sorgente trasmette il canale destro. Con una frequenza di campionamento FS = 8KHz si ha una frequenza di SCK = 512 KHz, mentre per una frequenza di campionamento FS = 192 KHz si ha una frequenza SCK = 12,288 MHz. La lunghezza della parola può essere di 16, 24 o 32 bits. Se ogni canale ha una lunghezza inferiore a 32 bit il frame viene solitamente incapsulato in 48 bits di dato ponendo a 0 i bits non utilizzati. Per canali aventi lunghezza pari a 16 bits invece l'intero frame è incapsulato in 32 bits di dato.

Molti IC non hanno molta flessibilità di configurazione, quindi, bisogna assicurarsi che i diversi dispositivi supportino le stesse lunghezze dei frame e di conseguenza le stesse frequenze di trasmissione. Il segnale SCK è il segnale di Serial Clock ed è il segnale di riferimento con cui viene inviato/letto ciascun bit di dato del segnale Serial Data SD.

La sua frequenza può essere derivata nel seguente modo:

$$fSCK = 2 * FS * nbits$$

Supponendo di voler trasmettere/ricevere un segnale audio stereo (2 canali L/R) con frequenza di campionamento FS = 48KHz e 32 bit di precisione:

$$fSCK = 2 * 48KHz * 32 = 3{,}072\ MHz$$

Come mostrato in *figura 14* il protocollo trasmette sulla linea SD a partire dal Most Significant Bit (MSB) con un ritardo iniziale pari a 1 periodo SCK a partire dalla prima variazione del segnale WS.

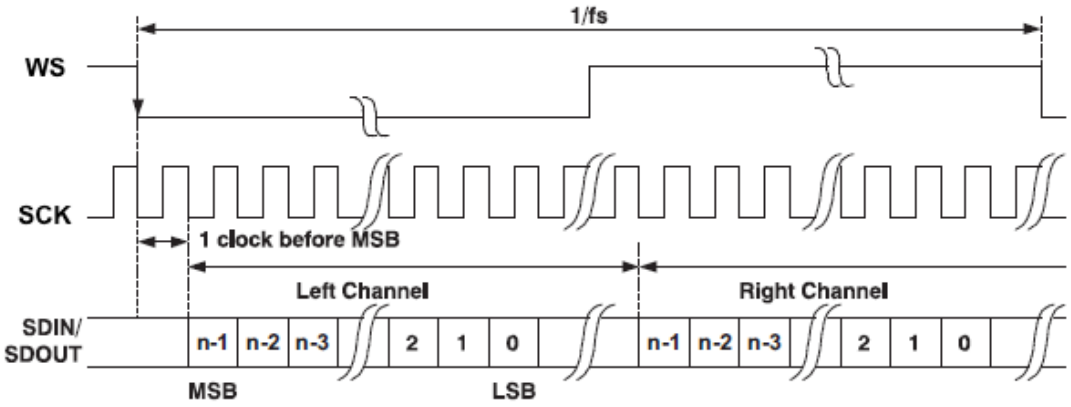


Figura 14: I2S Standard, TLV320AIC3107 datasheet [5]

2.2.2 I2S Left/Right Justified

A partire da I2S standard sono definite due varianti definite come Left Justified e Right Justified. La differenza sostanziale è che queste ultime “correggono” il ritardo iniziale della linea SD facendo in modo di allineare i frame sul lato sinistro o destro come mostrato in *figura 15* e *figura 16*

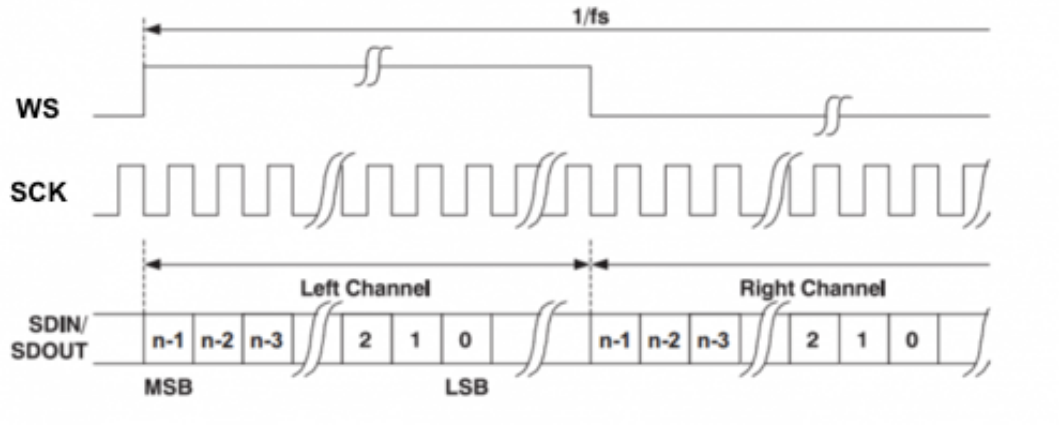


Figura 15: I2S Left Justified, TLV320AIC3107 datasheet [5]

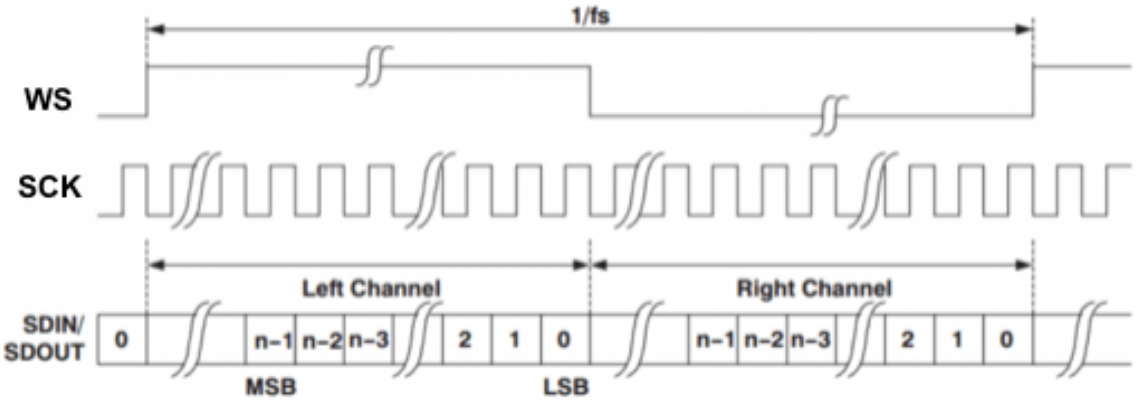


Figura 16: I2S Right Justified, TLV320AIC3107 datasheet [5]

2.2.3 DSP

L'interfaccia I2S può anche essere configurata come PCM, spesso definita come DSP. Questa modalità è fornita per consentire lo scambio dati con dispositivi esterni come moduli Bluetooth e differisce dalla modalità I2S in quanto il segnale WS non seleziona il canale L/R ma è un impulso che definisce l'inizio di un frame. La linea SD è pilotata in modo tale che i dati del canale sinistro siano immediatamente seguiti dai dati del canale destro con politica MSB first [3].

Il segnale WS può essere configurato per sincronizzarsi sul fronte di discesa (DSP MODE 0 – *figura 17*) o di salita (DSP MODE 1 - *figura 18*) del segnale SCK. Oltre a queste modalità, il DSP introduce altri 2 formati operativi definiti rispettivamente formato A e formato B che possono essere liberamente applicati alle DSP MODE sopra indicate:

- Il formato A (*figura 17*) è un formato in cui il frame inizia con un ritardo pari a un periodo SCK dopo la transizione WS (come nella modalità I2S).
- Il formato B (*figura 18*) invece è un formato in cui il frame inizia immediatamente dopo la transizione del segnale WS.

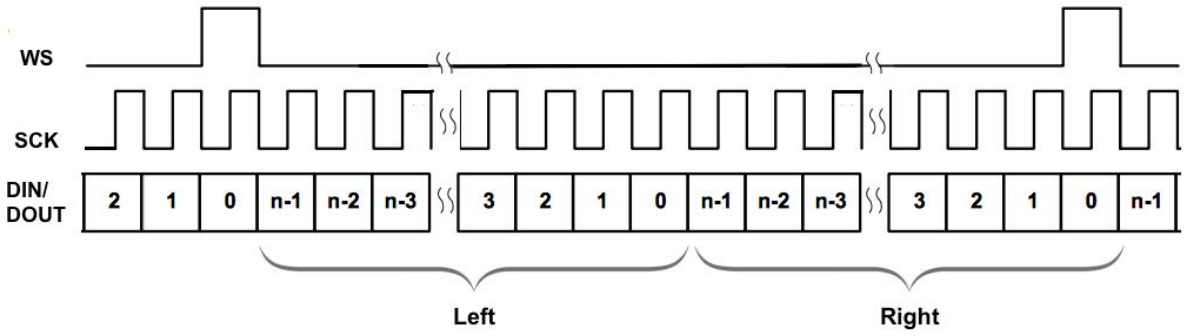


Figura 17: DSP mode 0 formato A, TLV320AIC3107 datasheet [5]

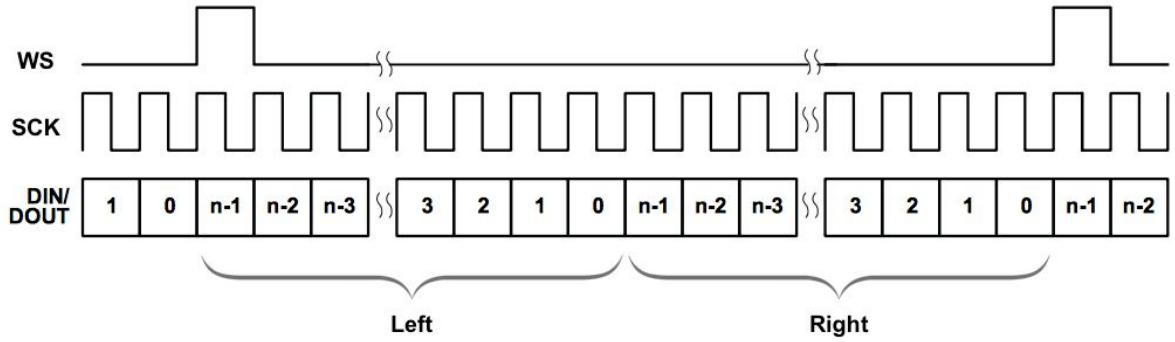


Figura 18: DSP mode 1 formato B, TLV320AIC3107 datasheet [5]

2.2.4 Time Division Multiplexing

Il Time Division Multiplexing (TDM) viene utilizzato quando si vogliono trasferire sulla stessa linea SD informazioni provenienti da *N* dispositivi.

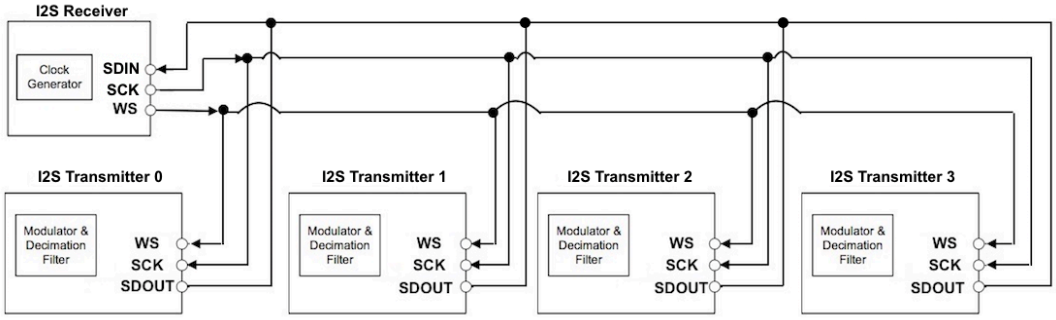


Figura 19: Interconnessione TDM

In questo modo, a ciascun dispositivo è assegnato uno slot temporale all'interno del quale viene associata l'informazione che trasmette. Supponendo di avere *N* dispositivi, ciascuno dispone per ciascun canale uno slot pari a $\frac{1}{Nslots}$ della dimensione di *nbits*. Il numero di slot temporali in cui è suddiviso ogni canale L/R è arrotondato per eccesso alla potenza di due più vicina ad N come mostrato in ***tabella 1***. Il numero N non è mai maggiore di 16 e dipende fortemente dall'implementazione fisica dell'interfaccia.

N Dispositivi	N slots
1-2	2
3-4	4
5-8	8
9-16	16

Tabella 1: Derivazione numero di slot nel frame

In *figura 20* è mostrato un esempio di TDM con un numero non definito N di CODEC in cascata. Come è possibile notare, ognuno dispone di uno slot per ciascun canale L/R.

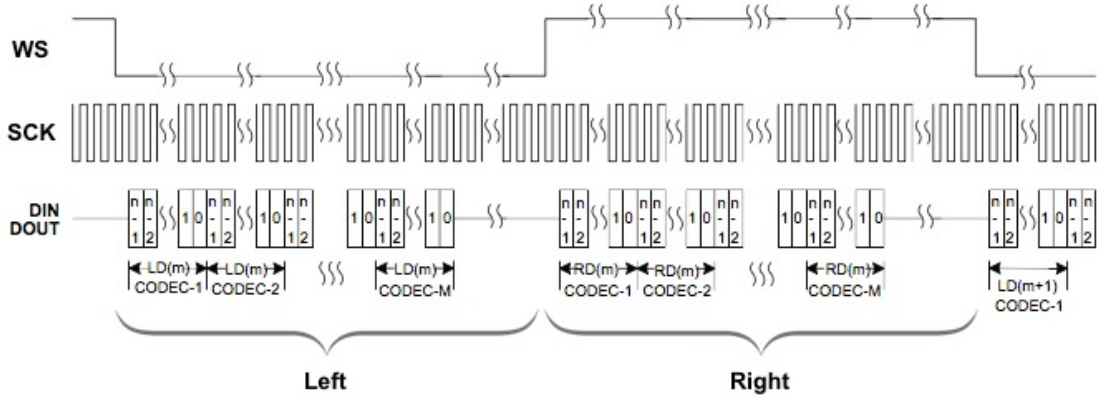


Figura 20: I2S TDM con N codec in cascata, TLV320AIC3107 datasheet [5]

Supponendo di voler multiplexare 8 dispositivi stereo (32 bits per canale), il frame avrà in totale 16 slots (8 slots per canale), come per I2S è possibile ricavare la frequenza SCK con TDM come:

$$fSCK = 2 * FS * nbits * nslots$$

$$fSCK = 2 * 48KHz * 32 * 8 = 24{,}57\ MHz$$

È evidente che aumentando il numero di dispositivi (2 – 4 – 8 – 16) la frequenza SCK raddoppia al raddoppiare del numero degli slots e può raggiungere l'ordine di 50 MHz se si hanno 16 dispositivi con FS=48KHz.

I dispositivi master che supportano la funzionalità TDM devono essere progettati per pilotare il bus dati solo durante lo slot che gli è stato assegnato, altrimenti devono porre la linea in tri-state per scollegarsi fisicamente dal bus. Questo comportamento è ottenuto programmando il dispositivo con un offset che stabilisce quanti periodi SCK intercorrono tra lo slot assegnato e la variazione del segnale WS.

La TDM non è uno standard per I2S, ciò significa che ogni produttore di IC può implementare la sua variante che può differire dalle altre rispetto alla polarità del clock, configurazione dei canali o sul tri-stating dei canali [2].

La maggior parte dei dispositivi audio che implementano la TDM sono progettati per utilizzare la modalità DSP, che è difatti conosciuta principalmente per l'implementazione della TDM [4]. Il segnale WS è un impulso e stabilisce l'inizio della trasmissione. Anche in questo caso, si suddivide l'intero frame in slots (vedi *tabella 1*) e la differenza sostanziale con TDM I2S è che ciascuno ha una profondità di bits pari a L+R (la modalità DSP trasmette i canali L e R in successione); ciò implica che la TDM DSP ha un numero di slots totali che sono esattamente la metà rispetto a TDM I2S.

Prendendo come riferimento il precedente esempio, si suddivide il frame in 8 slots con profondità di 64 bits (L+R) e la frequenza SCK è la medesima calcolata in I2S TDM. Il vantaggio sta nella semplificazione del protocollo e nella sincronizzazione dei dati tra master e slave in quanto la trasmissione/ricezione dei frame di un dispositivo non ha ritardi intermedi tra due canali. Per questa ragione la modalità DSP è quella che viene preferita per la TDM.

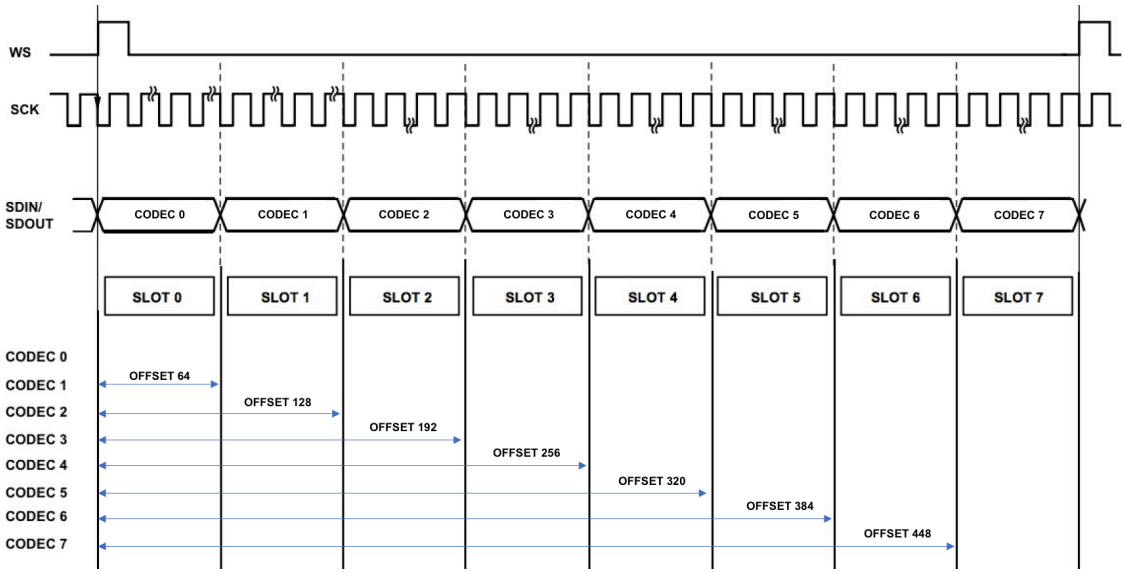


Figura 21: TDM DSP MODE 0 formato B

Come per la modalità DSP anche per la TDM possono essere implementate le DSP MODE nei formati A e B.

2.3 PULP Platform

PULP (Parallel Ultra-Low-Power) è una piattaforma di elaborazione multi-core open source sviluppata dalla collaborazione tra il gruppo di ricerca dell'Energy-Efficient Embedded Systems (EEES) dell'università di Bologna e l'Integrated Systems Laboratory (IIS) dell'ETH Zürich.

PULP è costituito da un'architettura Single Core (PULPissimo) e in aggiunta un parallel accelerator (Cluster) organizzati secondo il modello shared memory. Entrambi sono basati sul set di istruzioni open-source RISC-V con l'obiettivo di soddisfare le esigenze computazionali delle applicazioni IoT che richiedono un'elaborazione flessibile dei flussi di dati generati da più sensori come accelerometri, telecamere a bassa risoluzione, array di microfoni e monitor di segnali vitali. A differenza dei classici MCU single-core, l'architettura multi-core di PULP consente di soddisfare le esigenze computazionali di queste applicazioni senza superare il consumo tipico di pochi *mW*. Inoltre, PULP supporta OpenMP, OpenVC e OpenVX per consentire sviluppo, porting delle applicazioni e controllo delle performance [7].

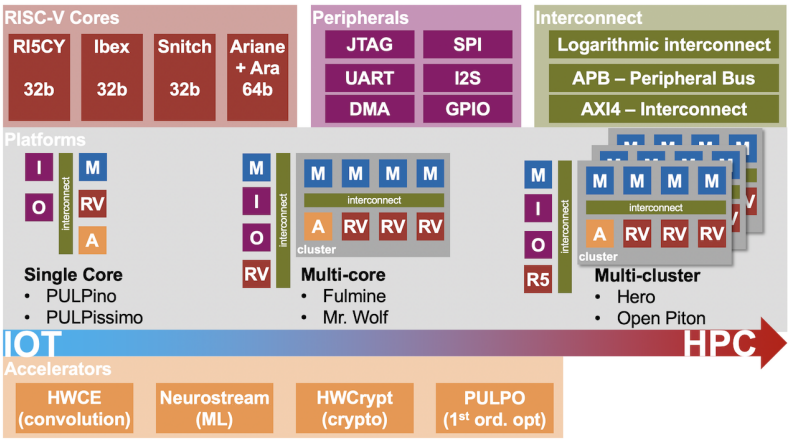


Figura 22: PULP Platform

2.3.1 PULPissimo

PULPissimo è un MCU single-core RISC-V a 32 bits con una pipeline a 4 stadi e Instruction Set Architecture (ISA) RV32IMC + estensioni tramite le quali è possibile eseguire istruzioni specifiche per migliorare performance di algoritmi DSP e machine learning. Queste estensioni includono hardware-loop, pointer post-increment, Multiply And Accumulate (MAC) e vettorizzazione SIMD.

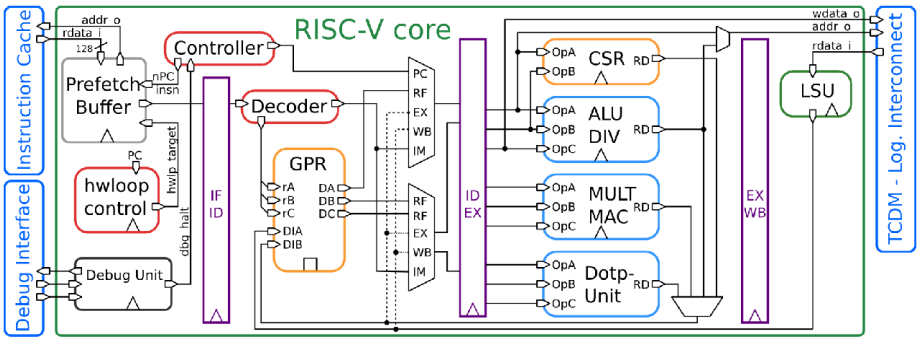


Figura 23: RISC-V ISA

Questo SoC possiede rispetto al suo predecessore PULPino un memory subsystem molto più complesso, un sottosistema di I/O autonomo (µDMA), nuove periferiche e nuovo System Development Kit (SDK). La *figura 23* mostra uno schema semplificativo dell'architettura; le interfacce Joint Test Action Group (JTAG) e Advanced eXtensible Interface (AXI) di cui è dotato danno accesso al SoC, inoltre, l'interfaccia AXI può essere utilizzata per estendere il microcontrollore con un sistema multi-core oppure con acceleratori hardware. Le periferiche invece, sono connesse al µDMA subsystem che si occupa di effettuare in modo efficiente i trasferimenti con la memoria [8]. Tutti registri dei moduli che compongono l'architettura sono memory mapped.

Questo MCU ha una frequenza che raggiunge al massimo qualche centinaio di MHz.

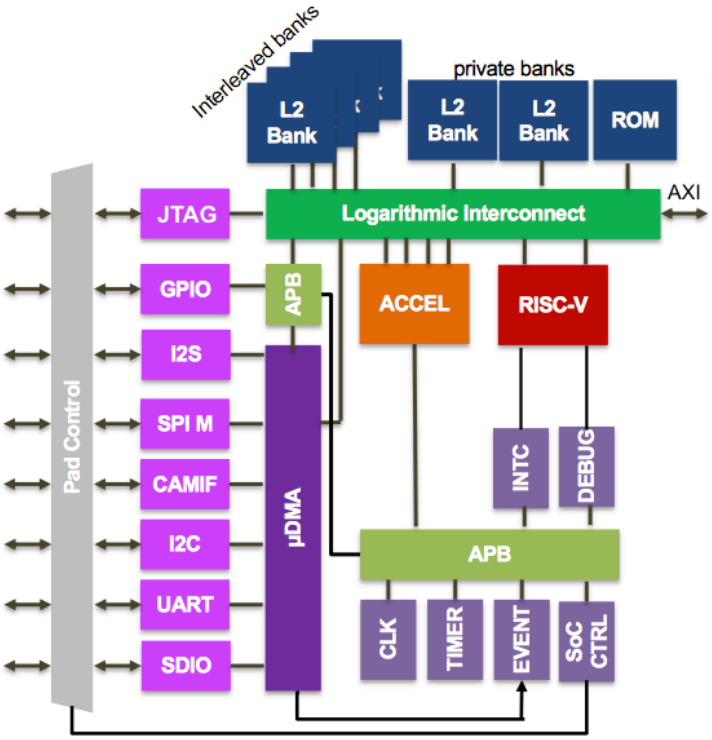


Figura 24: Architettura PULPissimo

2.3.2 PULP

PULP è un MCU multi-core costituito da un single-core PULPissimo con un parallel accelerator (Cluster) in cui vengono replicati 4-16 RISC-V cores. Una grande sfida nella progettazione multicore a basso consumo è la gerarchia della memoria. Le MCU a basso consumo tipicamente estraggono dati e istruzioni da memorie dedicate dotate di una singola porta di accesso: da un lato una configurazione di memoria così semplice non è adeguata a un sistema multicore e d'altra parte, complesse gerarchie di cache multicore non sono compatibili con target di potenza estremamente ridotti. Una buona alternativa alle caches è quella di utilizzare memorie di tipo *scratchpad* caratterizzate dal fatto di essere più piccole e meno costose in termini di accesso. Un ulteriore vantaggio è che memorie di tipo tightly coupled-data-memories (TCDMs) possono essere condivise da più core per consentire l'elaborazione sulla stessa struttura dati senza introdurre overhead hardware [9]. In PULP ci sono 2 livelli di memoria differenti, L1 per il cluster e L2 per l'intero SoC: la memoria L2 da 512KB è condivisa da tutto il SoC (PULPissimo + cluster), mentre il livello L1 da 128KB è accessibile solo dai cores del cluster ed ogni accesso avviene in media in 1 clock (TCDM). Inoltre, PULP può ulteriormente accedere ad un'area di memoria esterna L3 tramite HyperBus o QSPI [10]. L'interconnessione tra i banchi L1 e i cores nel cluster è effettuata da una full connectivity 1-cycle access crossbar che regola gli accessi e ne previene le collisioni e nel caso in cui più cores volessero accedere allo stesso banco, un arbitro che ne gestisce gli accessi fa stallare un core per 1 clock. In PULP ciascun core esegue differenti instruction stream seguendo lo schema di esecuzione Multiple Instructions Multiple Data (MIMD) ottimizzato per l'esecuzione di un Single Parallel Program su Multiple Data (SPMD). Tale ottimizzazione consiste nell'avere un'unica instruction cache condivisa dai core.

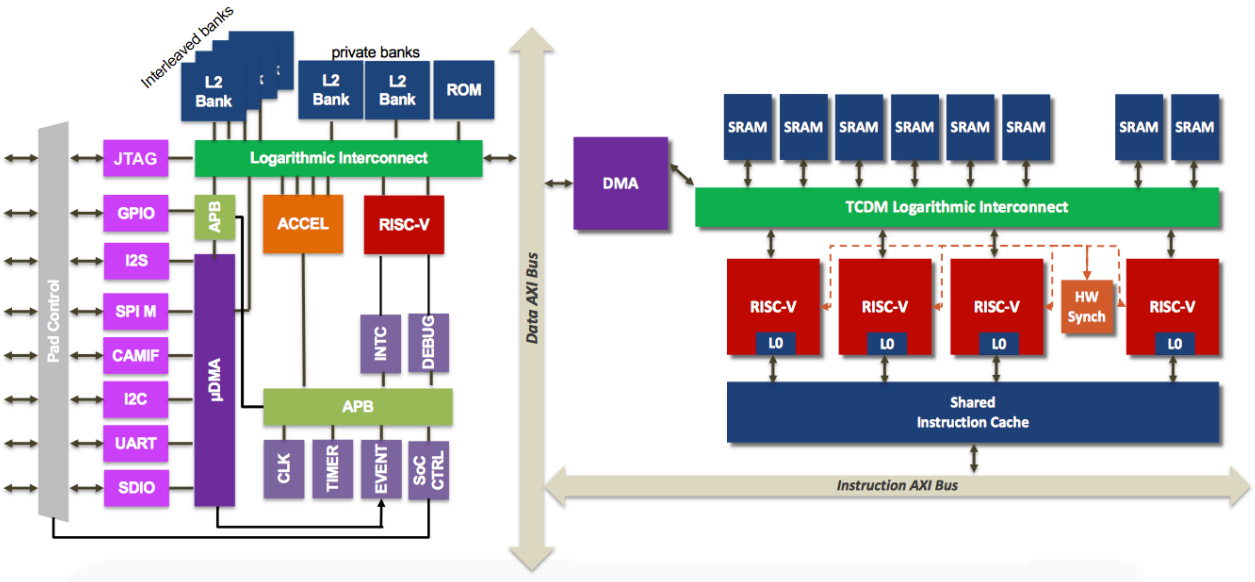


Figura 25: Architettura PULP

2.3.3 µDMA Subsystem

PULP è dotato di un sottosistema di I/O autonomo µDMA che si occupa di interagire con le periferiche. La maggior parte delle MCU avanzate dispongono di sottosistemi di I/O autonomi in grado di acquisire dati da più sensori solo quando la CPU è in stato IDLE; in questi sottosistemi tradizionali, l'interconnessione con le periferiche è condivisa con altre risorse del sistema e tutti si interfacciano alla memoria tramite la medesima porta di accesso. Come descritto in [11], in PULP, l'accesso alla memoria è regolato dalla TCDM Logarithmic Interconnect che arbitra gli accessi tra diversi agenti e ciò significa che l'architettura è dotata di molteplici porte tramite le quali è possibile accedere in memoria.

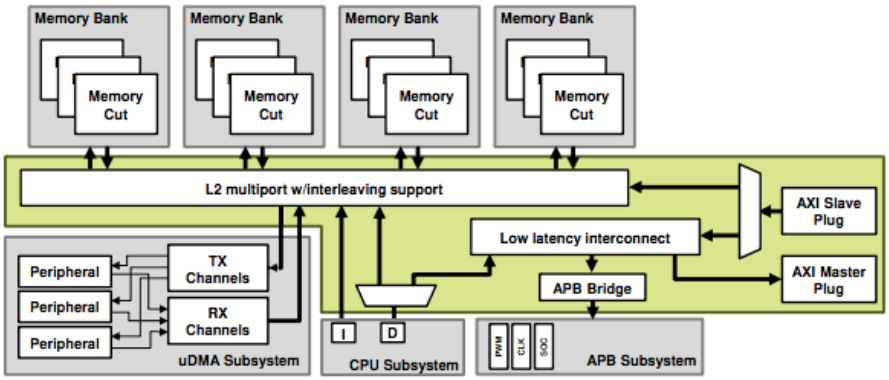


Figura 26: Interfacciamento alla memoria PULP, µDMA [11]

Una periferica può avere uno o più canali dati a seconda dei suoi requisiti di larghezza di banda e delle capacità di scambiare informazioni in trasmissione TX e ricezione RX. Questi canali sono monodirezionali, quindi sono dedicati per ciascuna funzione RX e TX. Il µDMA è dotato di 2 porte che si collegano alla TCDM interconnect, una RX per scrivere dati in memoria e l'altra TX per inviare dati verso una periferica. Queste connessioni dirette con la L2 limitano il µDMA ad accedere solo alla memoria di sistema e non permettono lo scambio diretto tra CPU, cluster o altre periferiche mappate sul bus APB. L'architettura di una generica periferica è mostrata *in figura 27.*

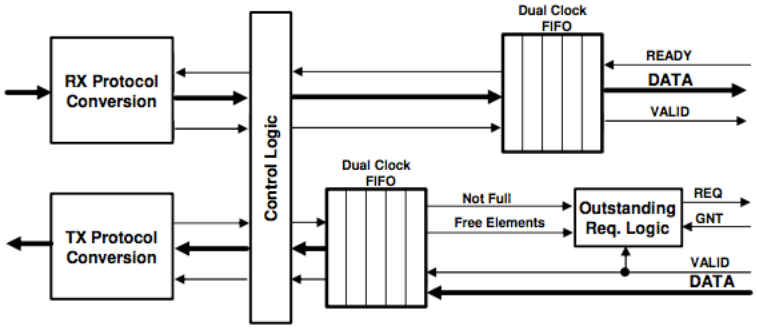


Figura 27: Periferica generica, µDMA [11]

Le porte verso la memoria sono di 32 bit mentre le interfacce a canale singolo dal lato periferiche possono avere dimensione variabile che dipende dalla periferica. Il µDMA supporta trasferimenti di 8, 16 o 32 bits che possono essere fissati durante il design o anche a run-time per adattarsi alla funzionalità specifica di una particolare periferica: UART e I2C per esempio usano canali di 8 bit poiché ad ogni trasferimento inviano sempre un singolo byte, mentre altre periferiche come SPI o I2S hanno una larghezza dati configurabile poiché dipende dal dispositivo. Per il canale RX è definito il seguente diagramma a blocchi che gestisce l'handshake con le periferiche ed esegue i trasferimenti in L2.

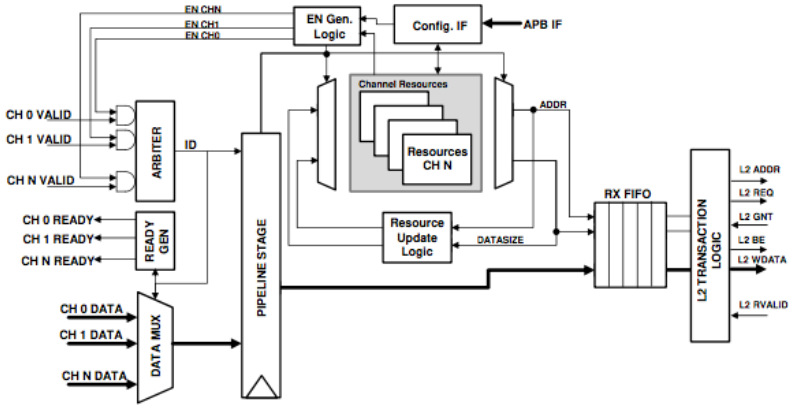


Figura 28: Architettura canale RX µDMA, µDMA [11]

Tutti i canali RX condividono la stessa porta verso la L2 e quando un dato è disponibile per essere trasferito, la periferica mette a 1 il segnale VALID e notifica al µDMA che ha un dato pronto. Il µDMA esegue un arbitraggio tra tutti i canali attivi e riconosce (con il segnale ready) alla periferica vincente il trasferimento dei dati; a questo punto la periferica considera compiuto il trasferimento e può quindi eseguire un'altra acquisizione [11].

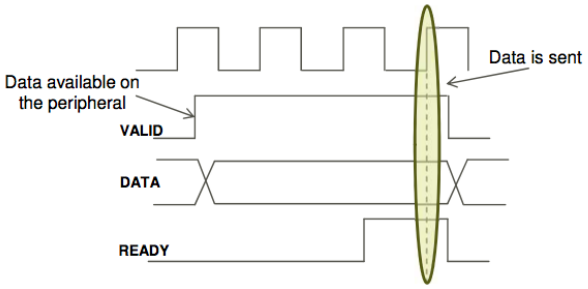


Figura 29: µDMA handshake channel RX, µDMA [11]

Similmente ai canali RX, anche quelli TX condividono la stessa porta con la memoria ma l'handshake del canale TX è più complesso rispetto RX in quanto ha percorsi separati per richiesta e risposta. Questo è necessario per dare supporto alle periferiche ad alta larghezza di banda che possono trovarsi a dover gestire richieste rimaste in sospeso per via di possibili latenze tra richiesta del dato e ricezione. Di seguito il è mostrato il diagramma a blocchi che gestisce l'handshake per i canali TX.

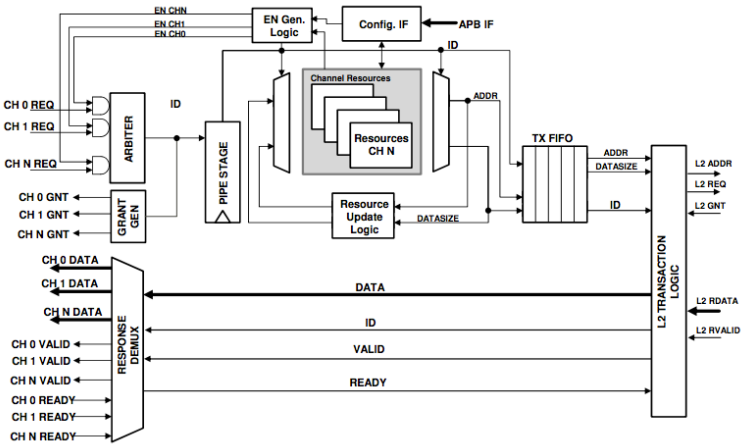


Figura 30: Architettura canale TX µDMA, µDMA [11]

Nel canale TX quando una periferica ha bisogno di dati, invia una richiesta al µDMA che arbitra le richieste ricevute e dà la concessione al canale vincente che non deve necessariamente aspettare i

dati in risposta ma può già emettere un'altra richiesta. Per mantenere la logica di handshake il più semplice possibile, si suppone che i dati di risposta non possano essere bloccati dalla periferica e che il numero di richieste in sospeso emesse dalla periferica dovrebbero essere sempre inferiori o pari al numero di posti disponibili nella FIFO locale. Le periferiche con bassa larghezza di banda non soffrono di questo problema: quando il µDMA concede una richiesta di una periferica, l'ID del canale vincente è memorizzato in uno stadio della pipeline e viene usato nel ciclo successivo per recuperare l'indirizzo e la dimensione dei dati richiesti. L'indirizzo con ID e datasize sono poi spinti nella TX FIFO. A valle della TX FIFO è posta una logica di transazione della memoria che estrae un elemento dalla FIFO TX ed esegue la lettura dalla L2. Al termine della lettura, i dati ricevuti vengono poi inviati al canale appropriato che viene estratto a partire dall'ID della richiesta servita.

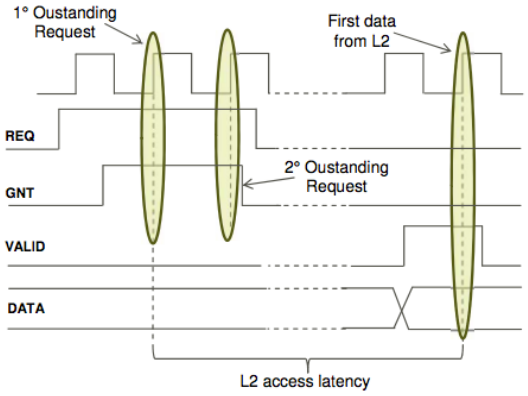


Figura 31: µDMA handshake channel TX, µDMA [11]

Per far partire una transazione su un canale associato ad una specifica periferica, è necessario programmare il µDMA via software tramite i suoi registri di configurazione mappati in memoria. In PULPissimo così come in PULP il µDMA subsystem è mappato nello spazio di indirizzamento 0x1A102000 - 0x1A103FFF e i suoi registri di controllo sono i seguenti:

Name	Address	Size	Type	Access	Default	Description
CTRL_CFG_CG	0x1A102000	32	Config	R/W	0x00000000	uDMA peripherals clock gate configuration
CTRL_CFG_EVENT	0x1A102004	32	Config	R/W	0x00000000	uDMA peripherals external event configuration

Figura 32: Registri di controllo µDMA, PULPissimo Datasheet [8]

Il registro di controllo CTRL_CFG_CG associa un bit per ciascuna periferica ed è utilizzato per effettuare il clock gating delle periferiche non necessarie. Prima di programmare i trasferimenti di una periferica è necessario abilitare il clock ponendo a 1 il bit associato. Il registro di controllo CTRL_CFG_EVENT invece, è utilizzato per associare uno specifico evento ad una periferica ed è composto da 4 bytes che identificano rispettivamente 4 eventi distinti. A seguire sono mappati i registri di configurazione delle periferiche.

Una periferica che usa entrambi i canali RX e TX dispone dei seguenti registri di configurazione necessari al µDMA per effettuare i trasferimenti appena descritti:

- RX_SADDR e TX_SADDR
- RX_SIZE e TX_SIZE
- RX_CFG e TX_CFG

- Registro Start Address (R/W)

  Configura un puntatore alla memoria che rappresenta l'indirizzo di partenza della transazione.
  Se acceduto in lettura R e il trasferimento è in corso ritorna il valore dell'attuale puntatore, altrimenti ritorna 0
  Se è acceduto in scrittura setta il puntatore.

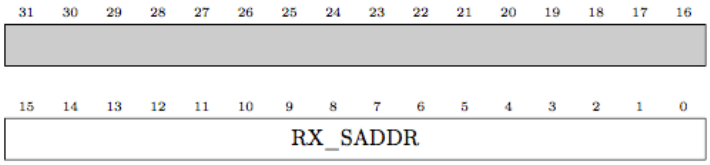


Figura 33: Registro RX_SADDR, PULPissimo Datasheet [8]

- Registro Size (R/W)

  Configura la dimensione in Bytes della transazione (max 128KB)
  Se viene acceduto in lettura R ritorna il numero di bytes che non sono ancora stati letti/scritti
  Se è acceduto in scrittura setta la dimensione del buffer.

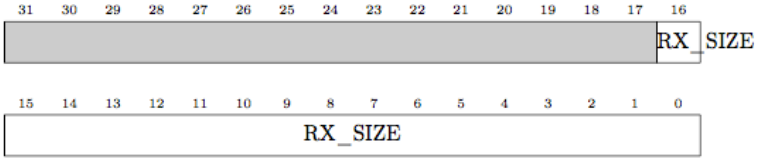


Figura 34: Registro RX_SIZE, PULPissimo Datasheet [8]

- Registro CFG

  Configura il µDMA per effettuare una transazione da/verso la memoria. Rispetto ai precedenti questo registro ha diversi campi

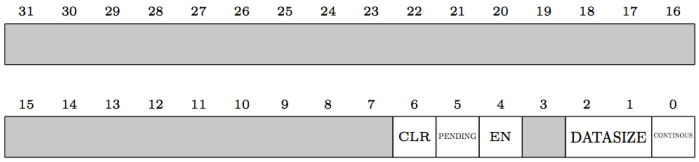


Figura 35: Registro RX_CFG, PULPissimo Datasheet [8]

  Bit 6 *CLR* (W) Ferma il trasferimento e resetta il canale:
  - 0: Disabilitato
  - 1: Abilitato

  Bit 5 PENDING (R) Indica lo stato dei trasferimenti
  - 0: canale libero
  - 1: Ci sono delle richieste in sospeso

  Bit 4 ENABLE (R/W) Abilita il canale e fa partire il trasferimento
  - 0: Disabilitato
  - 1: Abilitato

  Bits 2-1 DATASIZE (R/W) Indica la dimensione di ciascun trasferimento elementare e viene utilizzato dal µDMA per incrementare il puntatore del buffer SADDR
  - 00: +1 (8bits)
  - 01: +2 (16 bits)
  - 10: +4 (32 bits)

  Bit 0 CONTINUOUS (R/W) Abilita il canale in modalità continua (usato per streaming)
  - 0: Disabilitato
  - 1: Abilitato

2.4 Design RTL

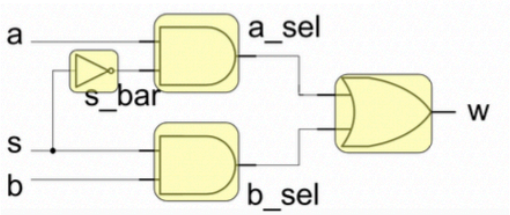
L'intera architettura PULP è stata sviluppata in System Verilog che è un Hardware Description Language (HDL) con lo scopo di descrivere il design in Register Transfer Level (RTL), ovvero specificare il comportamento di un circuito digitale per ogni ciclo di clock in termini di segnali, registri ed operazioni logiche.

Il linguaggio non specifica nel dettaglio come viene realizzato ogni singolo gate (AND, OR, NAND, NOT, ...) ma ha un livello di astrazione tale da essere paragonato a qualsiasi altro linguaggio. Quel che rende System Verilog molto semplice è che somiglia al linguaggio C anche se ha una sintassi abbastanza diversa, in quanto descrive il comportamento di un circuito logico e quindi di un hardware. In System Verilog il modulo rappresenta il componente principale che consente di descrivere un hardware: è costituito per descrivere un'interfaccia di un blocco funzionale a cui appartengono un insieme di segnali di input/output, segnali interni e specifiche funzionali.

Un circuito digitale è composto da elementi combinatori e da elementi sequenziali, perciò il linguaggio offre costrutti per poter descrivere entrambi i comportamenti e prevede la congiunzione di diversi elementi:

- Primitive

  Consentono di descrivere l'hardware ad un vero e proprio basso livello, molto simile alla descrizione di una netlist che in electronic design rappresenta una descrizione delle connessioni di un circuito e consiste in pratica ad una lista di componenti e nodi a cui questi sono connessi. L'utilizzo di un HDL è molto utile in quanto una netlist è composta da un insieme di moduli interconnessi fra loro ed ulteriormente da gate elementari, quindi semplifica di molto la descrizione rispetto alla classica interconnessione a blocchi.




```
module MultiplexerA (input a, b, s, output w);
  wire a_sel, b_sel, s_bar;
  not  U1 (s_bar, s);
  and  U2 (a_sel, a, s_bar);
  and  U3 (b_sel, b, s);
  or   U4 (w, a_sel, b_sel);
endmodule
```

Figura 36: Primitive in System Verilog

- Assign Statements

  Rappresentano un modo per specificare nel miglior modo il comportamento di un blocco logico combinatorio che al suo interno non ha al suo interno alcun elemento di memoria ma è costituito da soli gate pilotati da input logici tramite i quali vengono prodotti i rispettivi output. Lo statement *assign* consente di descrivere a livello *behavioural* una funzione combinatoria complessa che viene anche definita come continuous assignment e concettualmente indica che l'espressione a destra è valutata continuamente (dataflow). In questo modo non si descrive una netlist come nel caso precedente, ma si lascia libertà al tool di sintesi di tradurre l'espressione logica nella corrispondente netlist.

```
module MultiplexerB (input a, b, s, output w);
  assign w = (a & ~s) | (b & s);
endmodule
```

Figura 37: Assign statements

- Condition Expression

  Le espressioni condizionali risultano molto utili per descrivere il comportamento di un blocco in modo compatto e possono essere utilizzate per facilitare la descrizione di un'espressione booleana molto complessa

```
module MultiplexerC (input a, b, s, output w);
  assign w = s ? b : a;
endmodule
```

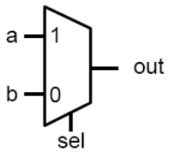
Figura 38: Condition expression

- Procedural Blocks

  I blocchi procedurali sono molto utili per descrivere espressioni logiche molto complesse e sono definiti dallo statement *always* e dalla sensitivity list associata che indica una lista di segnali. Lo statement *always* può essere utilizzato sia per la descrizione di reti combinatorie che per la descrizione di reti sequenziali. Nel caso di *always block* combinatori lo statement viene eseguito ad ogni variazione di ciascun segnale esplicitato nella sensitivity list e l'ordine di esecuzione del blocco è di tipo sequenziale. La descrizione di reti puramente sequenziali è effettuata tramite *always block* le cui sensitivity list sono pilotate dal clock ed in questo modo si esplicita la valutazione dell'intero blocco ad intervalli temporali stabiliti e su fronti ben definiti (posedge o negedge).

```
module combinational(a, b, sel,
                     out);
  input a, b;
  input sel;
  output out;
  reg out;
  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end
endmodule
```

```
module sequential(a, b, sel,
                  clk, out);
  input a, b;
  input sel, clk;
  output out;
  reg out;
  always @ (posedge clk)
  begin
    if (sel) out <= a;
    else out <= b;
  end
endmodule
```



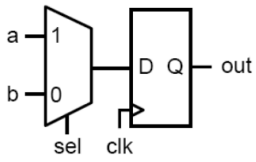


Figura 39: Always statement combinatorio e sequenziale

  All'interno di un *always block* sono ammesse due tipologie di assignment, *blocking* (=) e *nonblocking* (<=): il primo effettua una valutazione ed una assegnazione immediata, mentre il secondo effettua l'assegnazione solo quando sono state valutate tutte le espressioni sul lato destro. Come è evidente, l'assignment di tipo *blocking* non riflette il comportamento intrinseco di una logica sequenziale in cascata, perciò per gli *always block* sequenziali si usa il *nonblocking*, mentre per quelli combinatori il blocking.

Il design di un RTL comprende la descrizione ad alto livello, il partizionamento, il design della struttura dei bus e la descrizione e realizzazione dei vari componenti dell'architettura. Queste operazioni vengono suddivise in 3 step fondamentali:

1. Control/Data partitioning
2. Datapath Design
3. Control Design

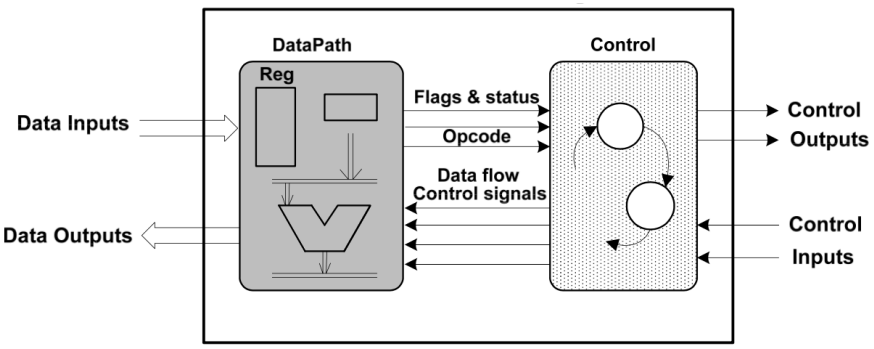


Figura 40: RTL Design

Il *datapath* è costituito sia da logica combinatoria che sequenziale ed è guidata da segnali di controllo prodotti dalla *control unit* che invece è puramente sequenziale ed è implementata da una macchina a stati finiti (FSM) che valuta i segnali ricevuti dal d*atapath* e ne guida il suo funzionamento tramite segnali di controllo.

Uno dei migliori metodi per descrivere il comportamento di una FSM in System Verilog è quello di usare il Three Always Block che usa tre *always block* separati per:

- Registro di stato presente - Sequenziale
- Valutazione dello stato futuro - Combinatorio
- Logica di output - Combinatorio

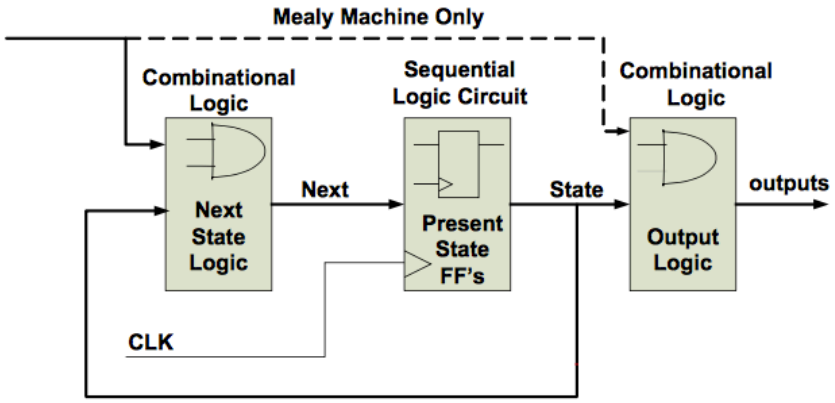


Figura 41: Three always block

Contributo

Il sistema PULP dispone di una periferica I2S con cui è possibile leggere dati fino ad un massimo di 4 dispositivi ad una frequenza massima di 10MHz utilizzando il protocollo I2S. Il mio contributo è stato quello di implementare all'interno della periferica I2S il protocollo TDM con DSP estendendo la lettura dati dalla stessa linea ad un massimo di 16 dispositivi. Inoltre, PULP non era in grado di poter trasmettere dati verso l'esterno, quindi per poter effettuare l'operazione di streaming verso ulteriori 16 dispositivi è stata introdotta anche questa funzionalità. Nel caso di multiplexing di 16 dispositivi, la frequenza massima raggiungibile è di circa 27MHz. Lo scopo di queste migliorie è stato quello di voler fare il tapeout di un chip specifico per applicazioni audio che integra al suo interno un acceleratore FFT ed un set di periferiche che oltre ad I2S ne aumentala flessibilità di utilizzo. Il lavoro ha quindi spaziato dallo sviluppo RTL della periferica I2S, all'organizzazione e definizione di tutto il sistema periferico fino al flusso di processo finale per il tapeout del chip. In questa sezione mi concentrerò principalmente sul design RTL delle nuove features integrate nella periferica I2S e sul flusso di implementazione fisica per la realizzazione del chip.

3. Sviluppo periferica I2S in PULP

In PULP la periferica I2S rispecchia lo schema architetturale mostrato in *figura 26* e dispone di due interfacce configurabili con due porte I2S separate, una è dedicata alla ricezione dei dati (slave) e l'altra alla trasmissione (master). Questa tipologia di implementazione in cui le due interfacce comunicano con l'esterno tramite porte distinte viene definita *full-duplex*; un'altra realizzazione possibile ma che richiede maggior logica di controllo e sincronizzazione è la modalità *half-duplex*. Le due modalità differiscono dal fatto che *half-duplex* alterna l'accesso al bus tra le due interfacce mentre *full-duplex* no. L'interfaccia I2S slave può ulteriormente essere configurata come PDM per ricevere dati da dispositivi che utilizzano questo protocollo.

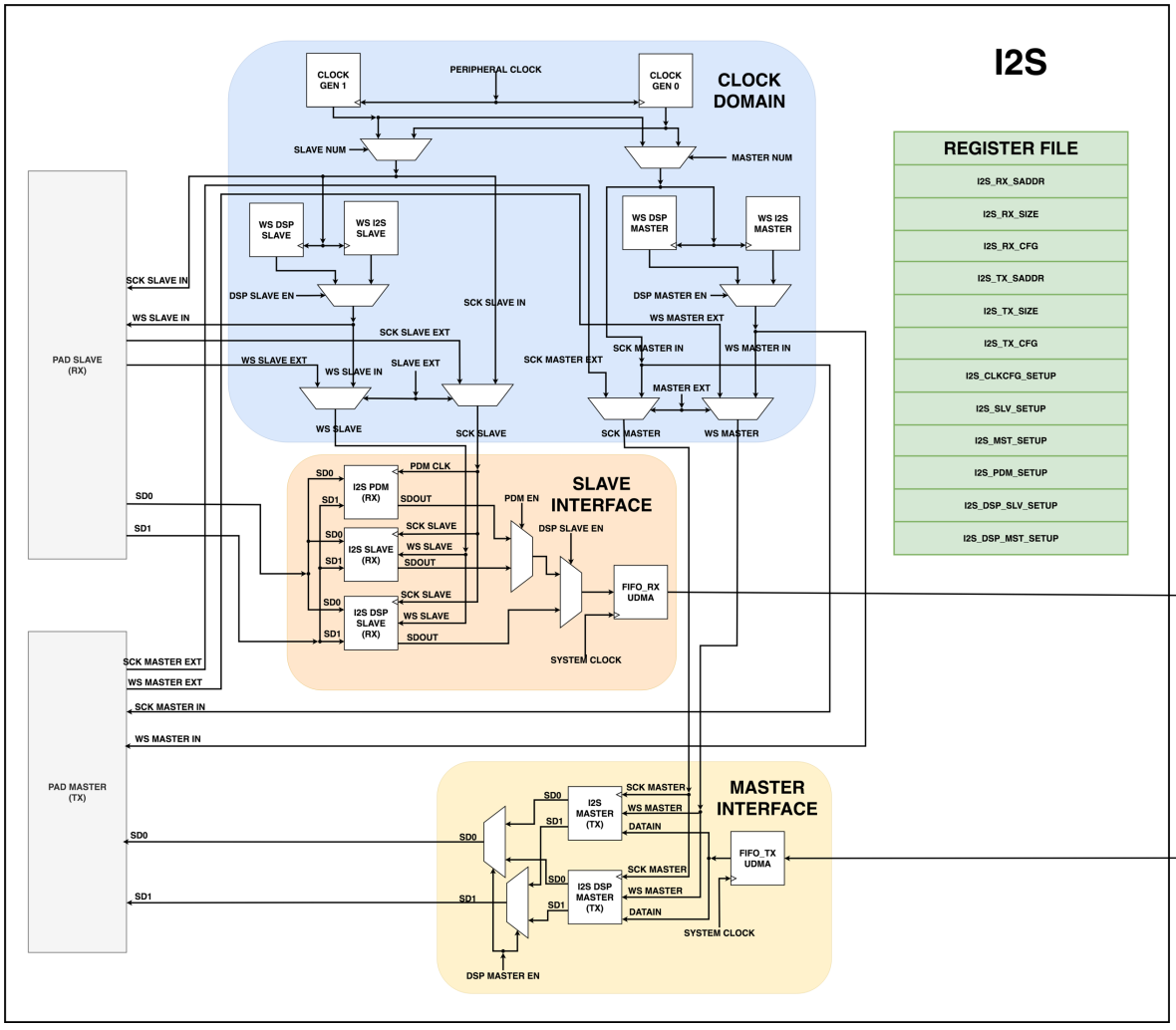


Figura 42: Schema periferica I2S in architettura PULP

Come mostrato in *figura 12*, I2S non impone una modalità precisa per la generazione dei segnali di sincronismo; per offrire la massima flessibilità la periferica è stata progettata per essere configurata secondo le diverse tipologie di interconnessione evidenziate.

3.1 I2S Register File

I registri di configurazione della periferica I2S sono i seguenti:

Name	Address	Size	Type	Access	Default	Description
I2S_RX_SADDR	0x1A102300	32	Config	R/W	0x00000000	RX Channel 0 I2S uDMA transfer address of associated buffer
I2S_RX_SIZE	0x1A102304	32	Config	R/W	0x00000000	RX Channel 0 I2S uDMA transfer size of buffer
I2S_RX_CFG	0x1A102308	32	Config	R/W	0x00000004	RX Channel 0 I2S uDMA transfer configuration
I2S_TX_SADDR	0x1A102310	32	Config	R/W	0x00000000	TX Channel I2S uDMA transfer address of associated buffer
I2S_TX_SIZE	0x1A102314	32	Config	R/W	0x00000000	TX Channel I2S uDMA transfer size of buffer
I2S_TX_CFG	0x1A102318	32	Config	R/W	0x00000004	TX Channel I2S uDMA transfer configuration
I2S_CLKCFG_SETUP	0x1A102320	32	Config	R/W	0x00000000	Clock configuration for both master, slave and pdm
I2S_SLV_SETUP	0x1A102324	32	Config	R/W	0x00000000	Configuration of I2S slave
I2S_MST_SETUP	0x1A102328	32	Config	R/W	0x00000000	Configuration of I2S master
I2S_PDM_SETUP	0x1A10232C	32	Config	R/W	0x00000000	Configuration of PDM module

Figura 43: I2S Register File, PULPissimo datasheet [8]

I registri con denominazione I2S_RX e I2S_TX sono i registri di configurazione del µDMA subsystem già analizzati nel capitolo precedente, i restanti sono registri specifici della periferica e sono utilizzati per controllarne il funzionamento.

Per l'implementazione della modalità TDM con DSP sono stati mantenuti i registri di configurazione esistenti per configurare clock e interfacce master/slave, inoltre sono stati aggiunti due nuovi registri (ciascuno associato ad una interfaccia) dedicati esclusivamente alla configurazione della nuova funzionalità.

Prima di procedere alla descrizione dei nuovi registri segue la descrizione dei registri I2S_CLKCFG_SETUP e I2S_SLV_SETUP. La descrizione del registro I2S_MST_SETUP è tralasciata perché segue l'omonimo registro associato all'interfaccia slave.

Il registro più importante della periferica è senza dubbio il registro di configurazione dei clock; questo viene utilizzato per selezionare le sorgenti di clock interne o esterne per entrambe le interfacce, per effettuare clock gating dei generatori di clock interni e per configurare i due generatori affinché producano la frequenza di funzionamento desiderata. Il registro I2S_CLKCFG_SETUP è mostrato in *figura 44*.

Figura 44: Registro I2S_CLKCFG_SETUP, PULPissimo Datasheet [8]

- Bit 31 MASTER_NUM (R/W) - Seleziona il generatore di clock per il master (0: master - 1: slave)
- Bit 30 MASTER_EXT (R/W) - Seleziona la sorgente di clock MASTER_NUM (0: interna - 1: esterna)
- Bit 29 SLAVE_NUM (RW) - Seleziona il generatore di clock per lo slave (0: master - 1: slave)
- Bit 28 SLAVE_EXT (R/W) - Seleziona la sorgente di clock SLAVE_NUM (0: interna - 1: esterna)
- Bit 26 PDM_CLK_EN (R/W) - Quando settato a 1 il clock slave di output è preso dal modulo PDM
- Bit 25 MASTER_CLK_EN (R/W) - Abilita il generatore di clock selezionato per il master
- Bit 24 SLAVE _CLK_EN (R/W) - Abilita il generatore di clock selezionato per lo slave
- Bits 23-16 COMMON_CLK_DIV (R/W) - MSB dei clock divider master e slave
- Bits 15-8 SLAVE_CLK_DIV (R/W) - LSB clock divider slave
- Bits 7-0 MASTER _CLK_DIV (R/W) - LSB clock divider master

Il registro di configurazione I2S_SLV_SETUP viene utilizzato per configurare l'interfaccia slave che riceve i dati dall'esterno e li mette sul canale RX del µDMA il quale si occuperà di effettuare i trasferimenti in memoria secondo le la configurazione stabilita per il canale RX. Nello specifico questo registro è così suddiviso:

Figura 45: Registro I2S_SLV_SETUP, PULPissimo Datasheet [8]

- Bit 31 SLAVE _EN (R/W) - Abilita l'operazione di lettura dall'interfaccia slave
- Bit 17 SLAVE _2CH (R/W) - Abilita l'acquisizione dati dalle due linee dati SD
- Bits 16 SLAVE _LSB (R/W) – Abilita la lettura a partire da LSB (0: MSB first – 1: LSB first)
- Bits 12-8 SLAVE_BITS (R/W) – Imposta il numero di bits per ciascuna word
- Bits 2-0 SLAVE _WORDS (R/W) – Imposta il numero di word per ciascuna fase I2S

Si fa presente che per estendere il multiplexing a 16 dispositivi, il campo SLAVE _WORDS è stato aumentato di un bit poiché 3 bits non sono sufficienti.

Per l'implementazione fisica della modalità TDM con DSP sono stati previsti due nuovi registri di 4 bytes I2S_DSP_SLAVE_SETUP e I2S_DSP_MASTER_SETUP mappati rispettivamente alla fine dello spazio di indirizzamento della periferica I2S agli indirizzi 0x1A102330 e 0x1A102334. Anche questi, come I2S_SLV_SETUP e I2S_MST_SETUP sono uguali e fanno riferimento a ciascuna interfaccia. Nel dettaglio, ciascun registro è composto come segue:

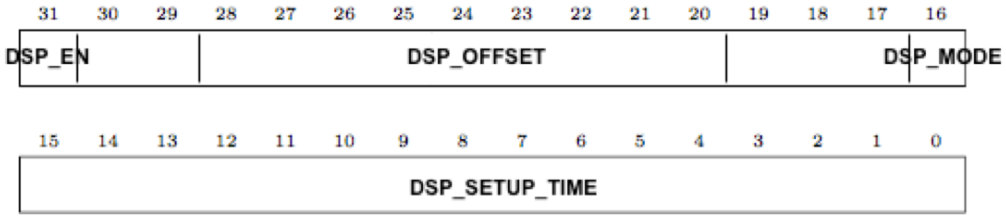


Figura 46: Registro I2S_DSP_SETUP

- Bit 31 DSP_EN (R/W) - Abilita la modalità DSP
- Bit 28-20 DSP_OFFSET (R/W) – Seleziona i formati A e B della modalità DSP ed effettua shifting
- Bits 16 DSP_MODE (R/W) – Seleziona le modalità DSP (0: DSP MODE 0 – 1: DSP MODE 1)
- Bits 15-0 DSP_SETUP_TIME (R/W) – Imposta un numero di cicli di clock prima del segnale di sincronismo WS.

L'ultimo campo DSP_SETUP_TIME è stato inserito per necessità da parte di alcuni dispositivi che hanno bisogno di un certo tempo per effettuare la loro inizializzazione prima di ricevere il segnale WS e ciò introduce la possibilità di poter configurare la periferica affinché possa rispettare i tempi di setup dei dispositivi che hanno questa necessità.

Supponendo di voler programmare l'interfaccia slave per comunicare con 8 dispositivi usando la TDM DSP (MODE 0 e formato B) come in *figura 21* è necessario configurare i registri del µDMA e della periferica nel seguente modo e nel seguente ordine:

1. CTRL_CFG_CG:

   Enable clock periferica I2S

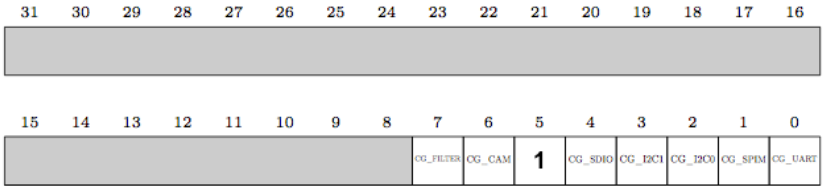


Figura 47: Configurazione CTRL_CFG_CG per TDM DSP, 8 dispositivi

2. RX_SIZE:

   32Bytes (4 Bytes x 8)

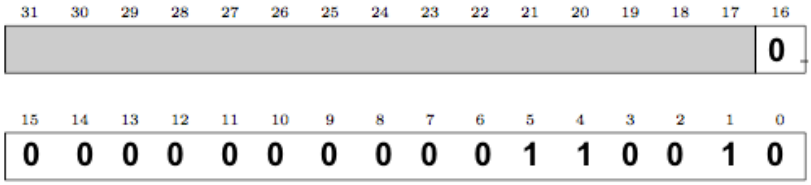


Figura 48: Configurazione RX_SIZE per TDM DSP, 8 dispositivi

3. I2S_SLV_SETUP:

   en_slave=1 - disable 2CH e LSB first - slave_bits=31 - slave_words= 7

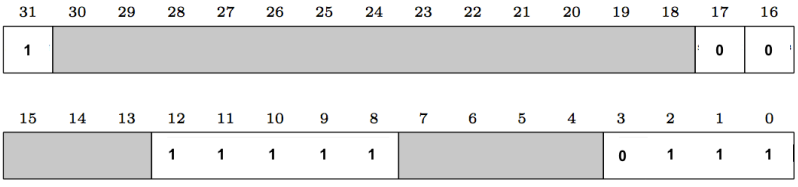


Figura 49: Configurazione I2S_SLV_SETUP per TDM DSP, 8 dispositivi

4. I2S_DSP_SLAVE_SETUP:

   en_dsp=1 - dsp_mode=0

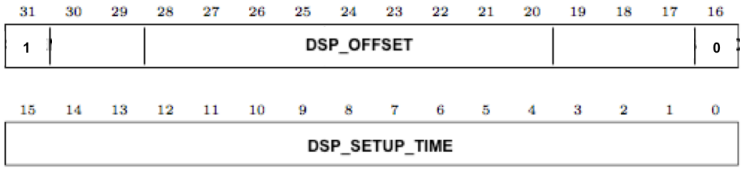


Figura 50: Configurazione I2S_DSP_SLAVE_SETUP per TDM DSP, 8 dispositivi

In questo caso la periferica può essere programmata con offset 0 in quanto leggerà in continuous mode gli slots che vengono trasmessi sulla linea immediatamente dopo WS. Al termine della lettura degli 8 slots verrà emesso un nuovo segnale WS e ricomincerà il processo di acquisizione dei dati.

5. I2S_CLKCFG_SETUP:

   slave_num=1 - slave_ext=0 - pdm_clk_en=0 - master_clk_en=0 - slave_clk_en=1

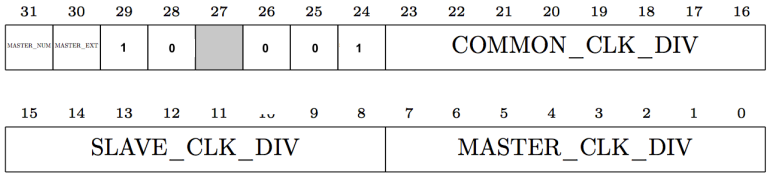


Figura 51: Configurazione I2S_CLKCFG_SETUP per TDM DSP, 8 dispositivi

In questo modo l'interfaccia slave è configurata per usare il generatore di clock interno associato allo slave. Volendo, si sarebbe potuto associare allo slave il clock generator del master semplicemente ponendo slave_num=0 e master_clk_en=1. Questo è comodo quando sia master che slave funzionano alla stessa frequenza e quindi si fa clock gating su un generatore.

6. I2S_RX_CFG:

   enable=1 - datasize=32 bits - continuous=1

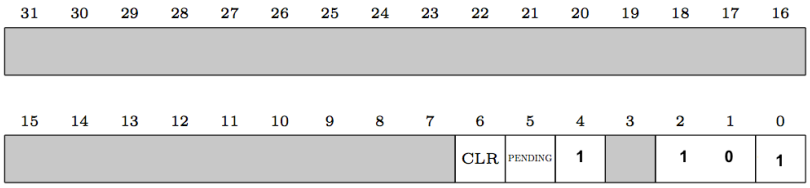


Figura 52: Configurazione I2S_RX_CFG per TDM DSP, 8 dispositivi

3.2 Dominio di clock

Il µDMA subsystem riceve in input due sorgenti di clock provenienti da due Phase Locked Loop (PLL) che generano il *system clock* e *peripheral clock*. Il *system clock* è usato per sincronizzare le FIFO dei canali TX e RX e per le scritture nel register file mentre *peripheral clock* viene usato come riferimento per generare il clock interno SCK. Come è già stato accennato, la periferica I2S dispone di due generatori di clock interni configurabili che vengono associati rispettivamente all'interfaccia master e slave. La periferica istanzia questi generatori di clock all'interno di un modulo denominato *i2s_clkws_gen* in cui sono gestiti i domini di clock evidenziati in *figura 53*.

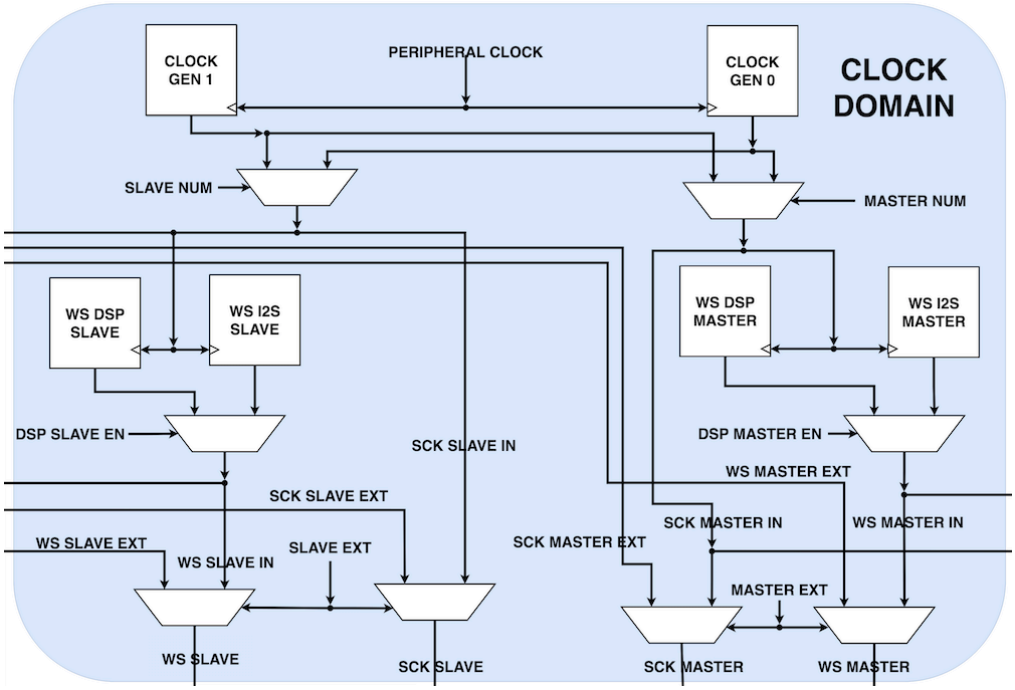


Figura 53: Dominio di Clock I2S

L'implementazione dei CLOCK GEN 0 e CLOCK GEN 1 è molto semplice: viene utilizzato un contatore che conta i periodi del *peripheral clock* e quando arriva al limite inverte il segnale di uscita *sck_o*. Nel caso del generatore dell'interfaccia slave, il limite del contatore proviene da un registro *cfg_div_1_i* che è la concatenazione dei bits COMMON_CLK_DIV e SLAVE_CLK_DIV del registro I2S_CLKCFG_SETUP. In System Verilog l'istanza del generatore di clock associato all'interfaccia slave appare come segue:

```
i2s_clk_gen i_clkgen1
  (
   .clk_i        ( clk_i ),
   .rstn_i       ( rstn_i ),
   .sck_o        ( s_clk_gen_1 ),
   .cfg_clk_en_i ( s_clk_gen_1_en ),
   .cfg_clk_en_o (  ),
   .cfg_div_i    ( cfg_div_1_i )
   );
```

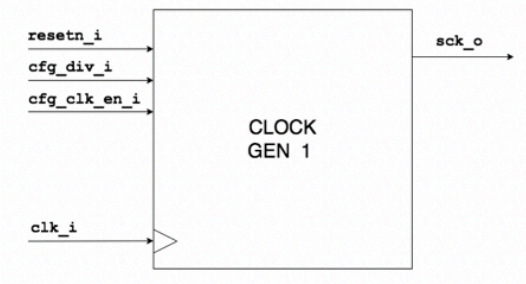


Figura 54: Generatore del segnale SCK per l'interfaccia slave

Questo segnale *sck_o* è il riferimento su cui si basa l'elaborazione dell'interfaccia slave e tutti i registri che essa contiene, analogamente l'interfaccia master basa la sua computazione sul segnale *sck_o* emesso dal suo generatore di clock.

3.3 Generatore WS con configurazione DSP

A partire dal segnale SCK, un modulo dedicato si occupa di generare il segnale WS che viene propagato verso l'esterno e anche in questo caso, come per i generatori di clock sono presenti due generatori del segnale WS associati a ciascuna interfaccia.

Per implementare il protocollo DSP sono stati inseriti due nuovi moduli per la generazione del segnale WS che possono essere programmati sia per comunicare con un solo dispositivo che per implementare TDM fino a 16 dispositivi.

Il protocollo prevede che il segnale WS rimanga attivo per una pulsazione della durata di un periodo SCK e che abbia una frequenza FS programmabile. In riferimento alla *figura 55*, sono stati definiti i seguenti stati di una FSM che implementa il funzionamento del protocollo.

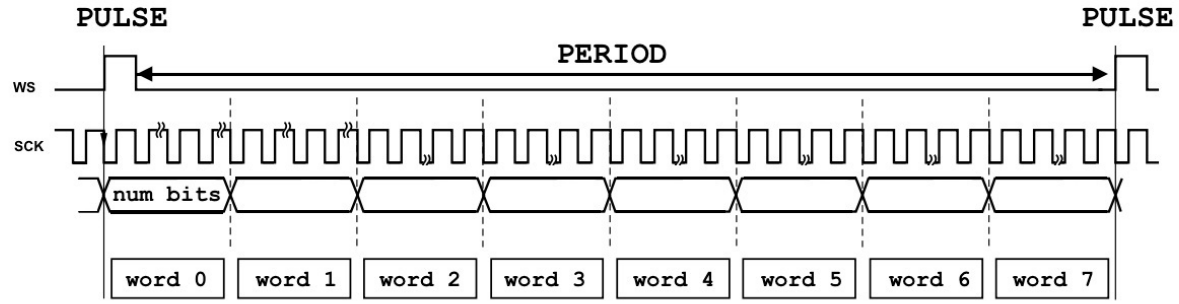


Figura 55: Stati FSM per generazione del segnale WS in configurazione DSP

L'andamento del segnale WS può essere rappresentato utilizzando soli 2 stati, PULSE che pone WS=1 e PERIOD che lo setta a 0 per un numero di periodi di clock dati dal numero di dispositivi connessi e dalla lunghezza di ciascun frame. Come è stato accennato, nella progettazione si è tenuto conto che alcuni dispositivi potrebbero richiedere un tempo di setup prima di ricevere il segnale WS=1, quindi per implementare questa possibilità è stato necessario introdurre un ulteriore stato di WAIT in cui il generatore di WS attende il trascorrere del tempo impostato come numeri di cicli di clock a partire dalla frequenza SCK.

La macchina a stati completa appare come in *figura 56*:

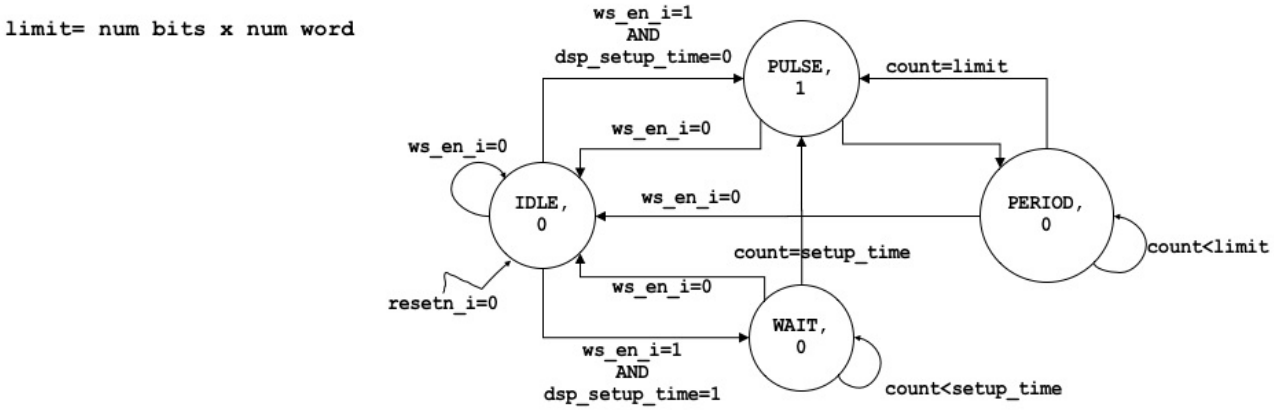


Figura 56: FSM generatore WS in configurazione DSP

È importante notare che nello stato PULSE non vi è condizione di uscita e non vi è alcuna situazione di permanenza dello stato in quanto è uno stato puramente transitorio in cui il sistema permane per un solo ciclo di clock prima di transitare autonomamente nello stato PERIOD. L'unica condizione che implica una transizione verso uno stato diverso da PERIOD è *ws_en_i=0* che pilota la FSM nello stato IDLE in cui permane fino alla nuova abilitazione (*en_dsp*=1 del registro

I2S_DSP_SLAVE_SETUP). Per poter implementare le DSP MODE 0 e DSP MODE 1 è stato necessario fare in modo che la macchina a stati fosse pilotata dai due fronti di SCK (posedge e negedge). Se la modalità selezionata è la DSP MODE 0, il segnale WS è pilotato sul fronte negativo (*figura 55*), quindi la macchina a stati è pilotata da un *always block* che agisce su un clock proveniente da un inverter del segnale SCK, mentre per la modalità DSP MODE 1 la macchina a stati è pilotata da un *always block* che agisce sui fronti positivi di SCK.

3.4 RX/TX DSP Channel

Come è stato più volte specificato, il protocollo DSP ha principalmente due modalità operative che differiscono l'una dall'altra rispetto al fronte di clock in cui viene pilotato il segnale WS. Per ciascuna variante il protocollo specifica la sincronizzazione tra chi trasmette e chi riceve nel seguente modo:

- DSP MODE 0 (WS=1 sul fronte di discesa di SCK)
  Il mittente mette i dati sulla linea SD sui fronti negativi, mentre il destinatario li campiona al successivo fronte positivo
- DSP MODE 1 (WS=1 sul fronte di salita di SCK)
  Il mittente mette i dati sulla linea SD sui fronti positivi, mentre il destinatario campiona i dati al successivo fronte negativo

Per implementare queste funzionalità nel canale RX è stato inserito all'interno dell'interfaccia slave un nuovo modulo denominato *i2s_dsp_rx_channel* che si occupa di campionare i dati dalla linea SD e passarli al µDMA tramite il protocollo di handshake mostrato in *figura 28.* Il comportamento è regolato dalla seguente FSM che definisce 3 stati: IDLE, OFFSET, RUN.

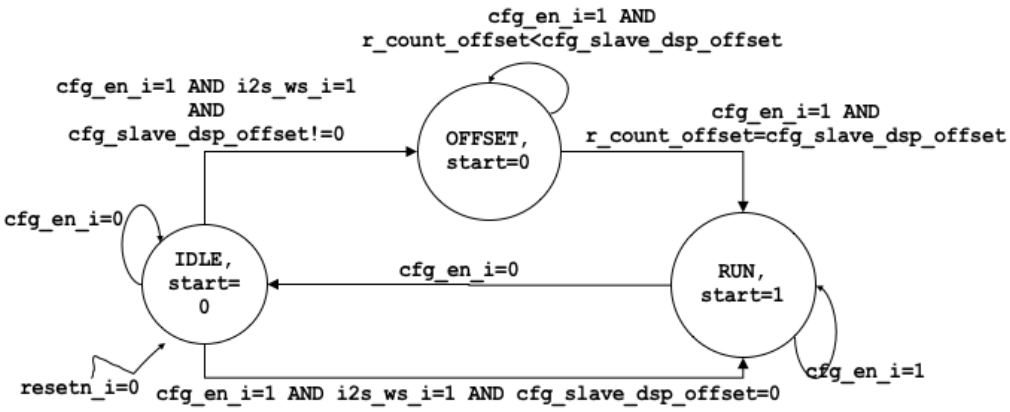


Figura 57: FSM RX channel

Il sistema è solitamente in stato IDLE ed effettua una transizione verso OFFSET o RUN solo quando la modalità è attiva e il segnale WS=1. La transizione nello stato OFFSET avviene solo se la periferica è programmata per implementare la modalità A con uno shift programmato, altrimenti transita in RUN. L'uscita della macchina a stati è un segnale di start che regola l'acquisizione dei dati del datapath.

L'implementazione dello stato OFFSET è molto semplice e consiste sostanzialmente in un contatore che si occupa di contare il numero di clock SCK per cui shiftare l'inizio dell'acquisizione che avviene nello stato RUN.

Il campionamento dei dati avviene in modo seriale e i dati letti dal bus vengono inseriti in uno shift register. Al termine dell'acquisizione di un frame della lunghezza di *n bits*, il registro a scorrimento è memorizzato in un registro ombra che mantiene l'acquisizione fino al termine della successiva che la sostituisce. In questo modo il µDMA ha a disposizione *n bits* clock per poter effettuare il trasferimento, se c'è congestione di richieste in sospeso il dato viene perso e sostituito dal successivo. Il campionamento dei dati per la linea SD0 è effettuato dalla rete logica in *figura 58* che mostra la concatenazione dei bits nello shift register associato alla linea. Il case statement in figura si occupa di concatenare un certo numero di 0 negli MSB del registro a scorrimento nel caso in cui la trasmissione sia configurata con LSB_FIRST=1.

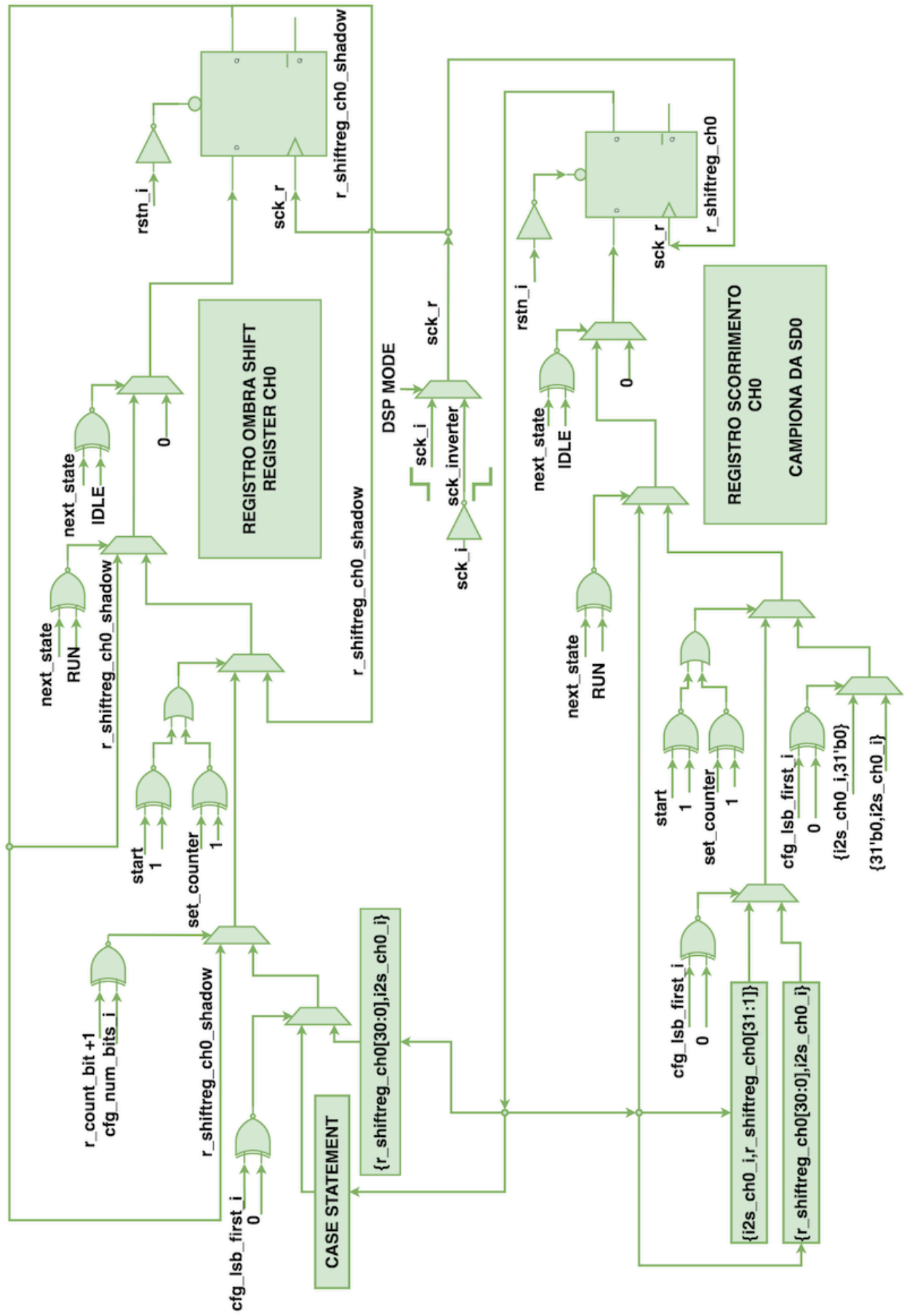


Figura 58: Diagramma a blocchi RX channel SD0

Per sincronizzare l'acquisizione sui due fronti del clock è stato inserito un inverter che ne effettua una inversione e tramite un multiplexer si seleziona la sorgente associata alla modalità operativa. In questo modo, tutti i registri dell'interfaccia campionano i dai sul fronte positivo se DSP MODE=0 e sul fronte negativo se DSP MODE=1. Al termine dell'acquisizione di un frame, oltre ad aggiornare il contenuto del registro *r_shiftreg_ch0_shadow* viene inviato al µDMA il segnale di VALID come stabilito dal protocollo di handshake per il canale RX (*figura 29*). Alla ricezione del segnale READY da parte del µDMA, la periferica considera concluso il trasferimento e rimuove il segnale precedentemente asserito.

Anche per l'interfaccia master è stato sviluppato un modulo *i2s_dsp_tx_channel* che implementa la trasmissione dati ed opera sul fronte opposto del clock rispetto a quello usato dall'interfaccia slave. Il comportamento del canale è regolato dalla seguente FSM che definisce 4 stati: START, SAMPLE, WAIT e RUN.

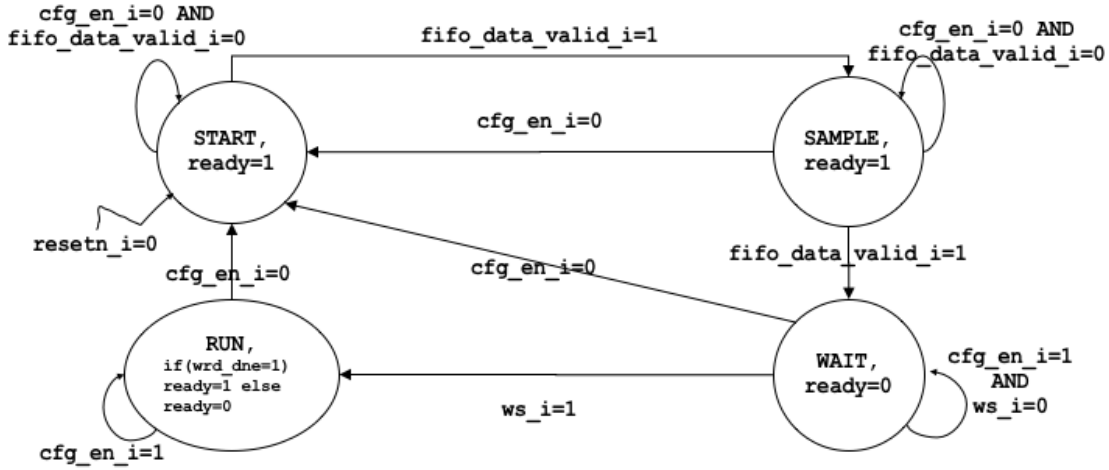


Figura 59: FSM TX channel

La macchina a stati proposta implementa il protocollo di handshake per il canale TX ed effettua i trasferimenti verso l'esterno. La macchina è solitamente in stato START fino a quando viene abilitato il canale TX ed il µDMA asserisce alla periferica il segnale VALID e a questo punto la periferica salva in uno registro interno il primo dato ricevuto ed effettua una nuova richiesta ponendo a 1 un segnale di ready e fa transitare la FSM nello stato SAMPLE in cui rimane in attesa del nuovo VALID associato al secondo dato restituito dal µDMA. In sostanza, la periferica, prima di iniziare i trasferimenti riempie due registri interni e si mette in attesa del segnale WS per poter iniziare a trasferire i dati sul bus. A questo punto quando viene ricevuto il segnale WS la periferica inizia a mandare il contenuto del primo dato (ricevuto in START) e quando il datapath ha terminato la trasmissione mette a 1 un segnale di sincronismo per la control unit (denominato *word_done)* che si occupa di asserire nuovamente il segnale di *ready* per richiedere un nuovo dato al *µDMA*. Nel mentre il datapath inizia a trasmettere il secondo dato ricevuto nello stato SAMPLE, al termine emetterà un nuovo segnale di *word_done* e ricomincerà nuovamente il ciclo descritto.

3.5 I2S Hardware Abstraction Layer

Per consentire la programmazione della periferica sono state implementate diverse funzioni all'interno del PULP runtime environment mostrato in *figura 60.*

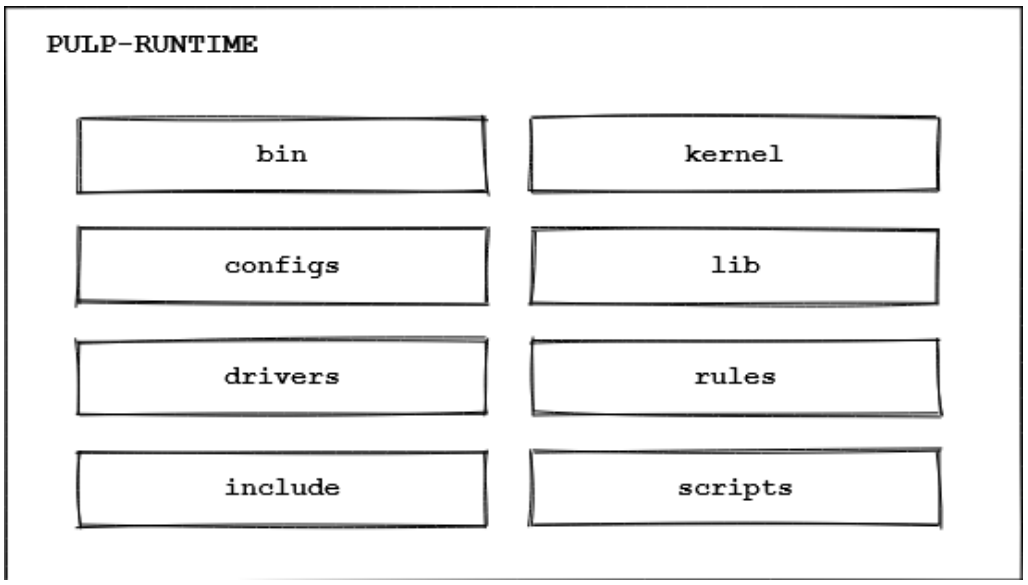


Figura 60: PULP Runtime environment

PULP runtime si occupa principalmente di effettuare il boot di PULP dalla bootrom e di guidare il microcontrollore al *main* del programma ed è semplicemente una configurazione minimale per poter essere eseguita velocemente ed è costituita da diverse cartelle:

- *Bin:* Contiene alcuni scripts che interagiscono con i tool per FPGA, alcuni dei quali sono usati per convertire file .elf in qualcosa che è può essere caricato in FPGA oppure nella testbench VSIM.
- *Configs:* Contiene diversi scripts che sono usati per effettuare il source dell'architettura di riferimento (PULPissimo, PULP)
- *Drivers:* sono contenuti i driver UART
- *Kernel:* Effettua principalmente l'inizializzazione degli FLL, dell'interrupt controller e dello stack. Successivamente giuda il core all'esecuzione del main.
- *Lib:* contiene le librerie di alcune funzioni c delle stantard library
- *Rules:* contiene delle regole per la compilazione
- *Scripts:* contiene diversi scripts per esempio per eseguire dei test sull'architettura
- *Include:* contiene l'Hardware Abstraction Layer (HAL) in cui sono definite tutte le funzioni utilizzabili per poter interagire con il core e con i moduli che compongono l'architettura di riferimento.

Tutte le funzioni che interagiscono con la periferica I2S sono contenute in un header file denominato *udma_i2s_v2.h* che contiene delle *define* che aiutano ad indentificare per ciascun registro l'offset e il bit meno significativo di ciascun campo. Inoltre, contiene la definizione e l'implementazione di tutte le funzioni che effettuano scrittura e lettura dei suddetti registri. Queste operazioni sono effettuate tramite *ARCHI_WRITE* e *ARCHI_READ* che richiedono rispettivamente un indirizzo base e un offset. Nel caso della periferica I2S l'indirizzo base è 0x1A102000, mentre gli offset dei registri vanno da *0x0* a *0x34.* A partire da queste funzioni basilari per la scrittura/lettura dei registri, sono state implementate altre funzioni che, presi in input i parametri fondamentali per la configurazione della modalità TDM con DSP, ne effettuano la programmazione.

Le funzioni messe a disposizione dello sviluppatore sono le seguenti:

```
void configure_i2s_dsp_slave ( uint32_t rx_addr, uint32_t buff_size, uint32_t lsb_first,
                               uint32_t num_bit, uint32_t num_words, uint32_t sel_clk,
                               uint32_t dsp_setup, uint32_t dsp_mode, uint32_t dsp_offset,
                               uint32_t clk_freq );

void configure_i2s_dsp_master ( uint32_t tx_addr, uint32_t buff_size, uint32_t lsb_first,
                                uint32_t num_bit, uint32_t num_words, uint32_t sel_clk,
                                uint32_t dsp_setup, uint32_t dsp_mode, uint32_t dsp_offset,
                                uint32_t clk_freq );
```

Come è evidente, entrambe hanno gli stessi parametri di input:

- RX/TX _ADDR: Puntatore ad un indirizzo di memoria
- BUFF_SIZE: Dimensione in bytes della transazione
- LSB_FIRST: Flag per abilitare l'invio/ricezione a partire dal bit LSB
- NUM_BITS: Numero di bits di ogni frame
- NUM_WORDS: Numero di frame da inviare/ricevere
- SEL_CLK: Selettore clock
- DSP_SETUP: Tempo di setup espresso in cicli di clock
- DSP_MODE: Seleziona una modalità DSP
- OFFSET: Setta lo shift del frame in numero di clock
- CLK_FREQ: Imposta la frequenza di clock del segnale SCK

Ciascuna delle funzioni appena introdotte effettua (tramite le HAL descritte precedentemente) la sola configurazione dei registri e dei campi associati all'interfaccia per cui sono state sviluppate.

La simulazione in hardware della periferica è stata effettuata tramite il tool QuestaSiM, sviluppato da Mentor Graphics per la verifica funzionale di linguaggi HDL. Il PULP runtime environment guida il microcontrollore all'esecuzione del programma, e tramite QuestaSim è possibile visualizzare e verificare il comportamento di tutta l'architettura analizzando le forme d'onda di interesse. Per poter applicare degli stimoli al microcontrollore è stata sviluppata una testbench che è un codice di simulazione usato per applicare al design una determinata sequenza di input col fine di verificarne l'output. La testbench istanzia il Design Under Test (DUT) di PULP e viene utilizzata per connettere i Verification IP (VIP), ovvero moduli HDL sviluppati per simulare il comportamento di dispositivi esterni al design. Nel caso della periferica I2S, sono state sviluppate diverse VIP che hanno aiutato lo sviluppo, la verifica funzionale e la correzione di errori del design; nello specifico sono stati sviluppati dei VIP capaci di inviare e ricevere dati col fine di simulare il funzionamento di microfoni e codec. Al termine del design, le VIP appena citate sono state rimosse per essere sostituite da una semplice retroazione del segnale SD mostrata in *figura 61* per rendere possibile il test simultaneo di entrambe le interfacce senza aver bisogno di moduli HDL esterni al design.

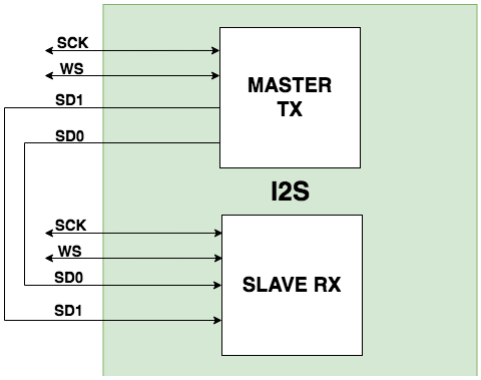


Figura 61: Schema retroazione testbench

Dopo aver configurato la testbench affinché le due interfacce siano connesse tra loro è stato possibile verificarne il corretto funzionamento tramite l'esecuzione di un programma di test che prima inizializza il buffer TX e poi configura la periferica per effettuare uno streaming continuo in cui l'interfaccia master mette sul bus il contenuto del buffer TX e l'interfaccia slave campiona i dati e li passa al µDMA per essere scritti nel buffer RX.

Segue la configurazione di un programma di test usato per la validazione della periferica:

SLAVE CONIFG	**MASTER CONIFG**
WS Freqency: 48 KHz	WS Freqency: 48 KHz
Num bits: 32	Num bits: 32
Num words: 16	Num words: 16
LSB first (0: no, 1: yes): 0	LSB first (0: no, 1: yes): 0
Setup time: 0	Setup time: 0
Offset: 0	Offset: 0
TDM DSP mode: 0	TDM DSP mode: 0
Clock (0: master, 1: slave): 0	Clock (0: master, 1: slave): 0

Le forme d'onda principali delle due interfacce sono mostrate in *figura 62* e corrispondono rispettivamente al canale TX dell'interfaccia master e al canale RX dell'interfaccia slave. Il tool QuestaSim consente di verificare la corrispondenza delle operazioni definite in RTL ad ogni variazione di SCK e di verificare la corretta configurazione dei timing relativi alle frequenze SCK e WS.

Il canale TX riceve dal µDMA il primo dato letto dalla memoria (8) e inizia a trasmetterlo sul bus *i2s_ch0_*o alla ricezione del segnale WS. Allo stesso tempo il canale RX campiona i dati sul bus e dopo 32 cicli di clock mette a disposizione del µDMA il dato pronto per essere trasferito in memoria e attiva il segnale di VALID. Al termine del primo trasferimento, il canale TX dispone già del successivo dato (10) ed inizia a trasmetterlo immediatamente dopo il primo che verrà completamente ricevuto dopo ulteriori 32 cicli di clock.

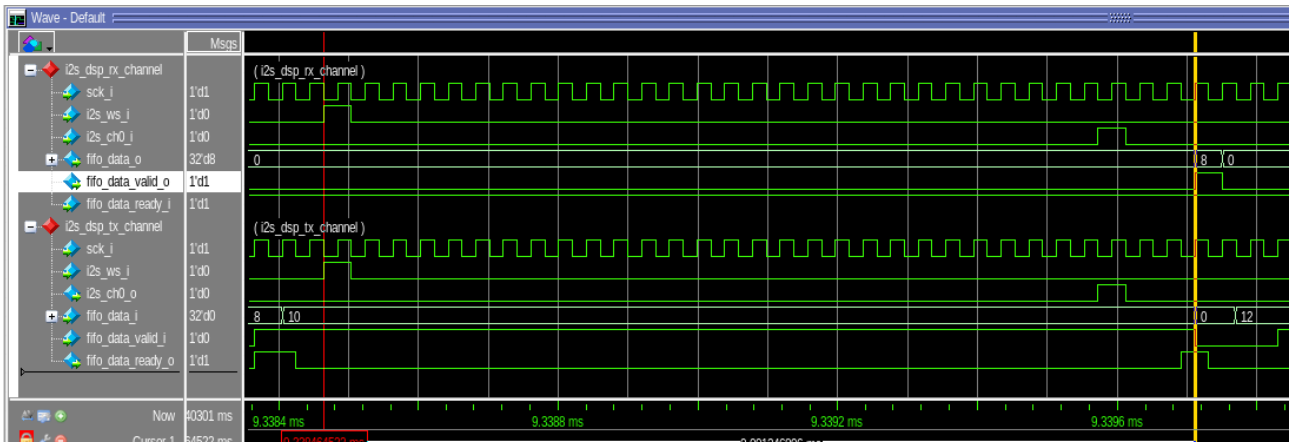


Figura 62: Invio e ricezione del primo frame

Questo processo va avanti fino all'invio della sedicesima word prima che il segnale WS assuma nuovamente il valore logico 1 e come è mostrato in *figura 63,* il tool QuestaSim dispone di un terminale di output sul quale vengono stampate le *printf* inserite nel codice di test che oltre a configurare la periferica si occupa di verificare che i dati ricevuti nel buffer RX siano uguali a quelli contenuti nel buffer TX. Il test termina così con esito positivo ed il funzionamento della periferica è quello desiderato.

A questo punto la periferica può essere validata ed è possibile procedere al successivo step per l'implementazione fisica e la realizzazione del chip.

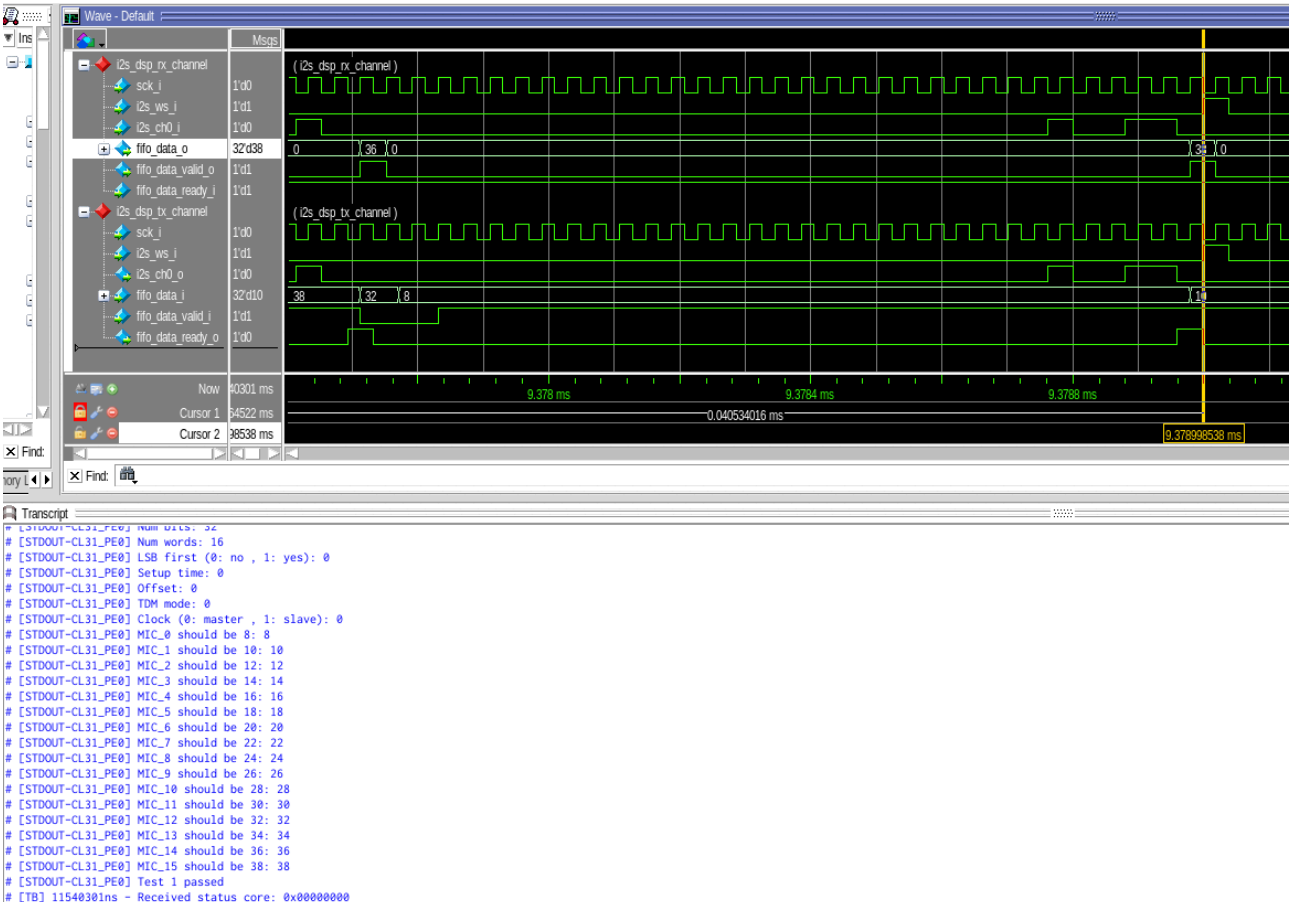


Figura 63: Validazione periferica I2S

3.6 Risultati

Con l'implementazione del protocollo DSP viene sia quadruplicato il numero di dispositivi da cui è possibile ricevere dati e sia concessa la possibilità di trasmissione verso un massimo di 16 dispositivi, funzionalità che precedentemente mancava. Inoltre, viene eliminata la latenza che intercorre tra l'invio di due canali (L/R) associati ad un dispositivo. Se si utilizza la TDM con I2S è evidente che per poter ricevere l'intero frame del dispositivo 0 è necessario che tutti i dispositivi connessi alla linea trasmettano il canale L, pertanto il tempo che intercorre tra l'inizio della trasmissione L0 e la completa ricezione R0 del singolo device aumenta proporzionalmente con l'aumentare dei dispositivi. Nel caso della TDM con DSP la ricezione di un frame associato ad un dispositivo non ha latenze intermedie tra due canali, ciò garantisce una maggiore velocità di acquisizione che è un requisito fondamentale per effettuare audio processing in tempo reale.

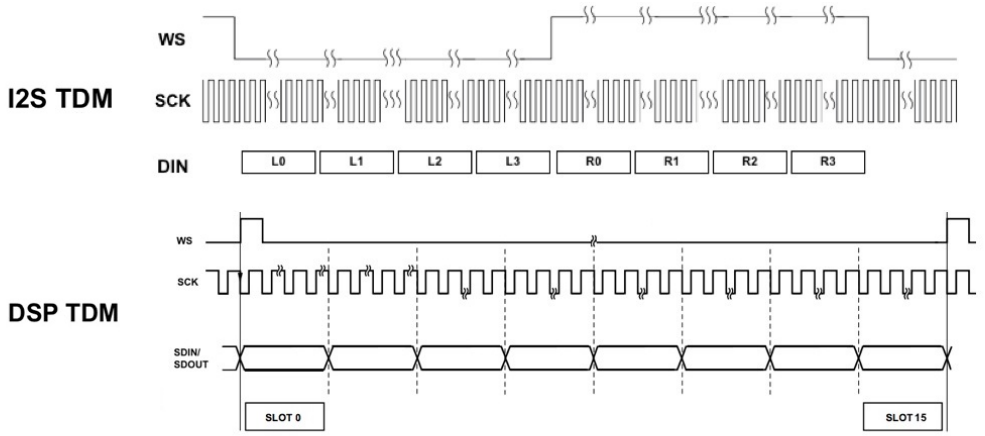


Figura 64: Confronto TDM I2S e TDM DSP

Supponendo che sulla linea siano multiplexati *K* dispositivi con frame di *nbits* è possibile definire la latenza per ciascun dispositivo come:

$$I2S\ Latency = \frac{nbits}{2} * (K + 1) * Tclk$$

$$DSP\ Latency = nbits * Tclk$$

Confrontando le features di cui era dotato PULP con quelle che sono state integrate è possibile notare un netto miglioramento in termini di flessibilità e performance ottenibili, inoltre è evidente che la latenza che intercorre tra la ricezione dei canali L/R nel caso di TDM con DSP è indipendente dal numero dei dispositivi ed equivale al numero di bits del frame, mentre TDM con I2S e solo 4 dispositivi ha una latenza di 80 clock per la completa ricezione di un singolo frame.

	I2S in PULP	
	Legacy I2S	New I2S with DSP
Max devices on RX	4	16
Max Devices on TX	0	16
Max SCK MHz	10	27
Latency between L/R	80	32

nBits	32

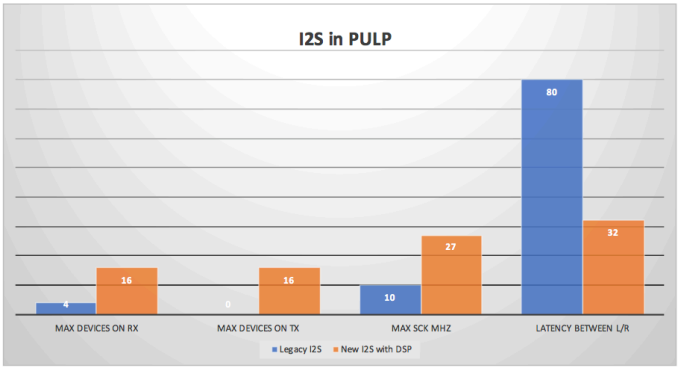


Figura 65: Confronto versioni I2S in PULP

4. Echoes

Un ulteriore contributo oltre all'implementazione delle nuove funzionalità nella periferica I2S in PULP è stato quello di voler progettare e fare il tapeout di un MCU specifico per applicazioni audio.

Il microcontrollore prende il nome di Echoes ed è un single core basato su PULPissimo dotato di 256KB di memoria interleaved con 16 banchi da 16KB, due banchi privati di SRAM da 32KB e una ROM da 8KB. Ciò che lo rende unico nel suo genere è che in aggiunta al protocollo TDM è presente un acceleratore hardware Fast Fourier Transform (FFT) con cui è possibile effettuare un'analisi dello spettro dei segnali audio o estrarre dei pattern per il riconoscimento di alcuni suoni. Molti microcontrollori effettuano questa operazione tramite librerie specifiche per l'esecuzione dell'FFT ma rispetto all'elaborazione hardware di Echoes hanno peggiori performance in termini di esecuzione ed efficienza energetica e questo è il principale motivo per cui Echoes è stato progettato. Questo MCU ha ulteriore flessibilità di utilizzo grazie al set di periferiche di cui è dotato:

- HyperBus
- I2S
- UART
- QSPI
- SPIM
- I2C

Inoltre, se lo sviluppatore finale non ha necessità di utilizzare tutte le periferiche elencate può decidere di configurare il pad del microcontrollore per utilizzare i pin delle periferiche come General Purpose Input/Output (GPIO).

Col fine di salvaguardare area e power consumption, la memoria on-chip L2 è stata ridotta a 256KB che viene compensata, se necessario, grazie alle due periferiche SPI e Hyperbus con le quali è possibile connettere della memoria aggiuntiva off-chip.

Come mostrato in *figura 67* Echoes dispone di un sottosistema autonomo µDMA che regola lo scambio dati tra periferiche e L2, un acceleratore FFT ed una Floationg Point Unit FPU. Il core utilizzato per questo chip è un CV32E40P che è un processore a 32bit basato su RISC-V con 4 stadi di pipeline (*figura 66*).

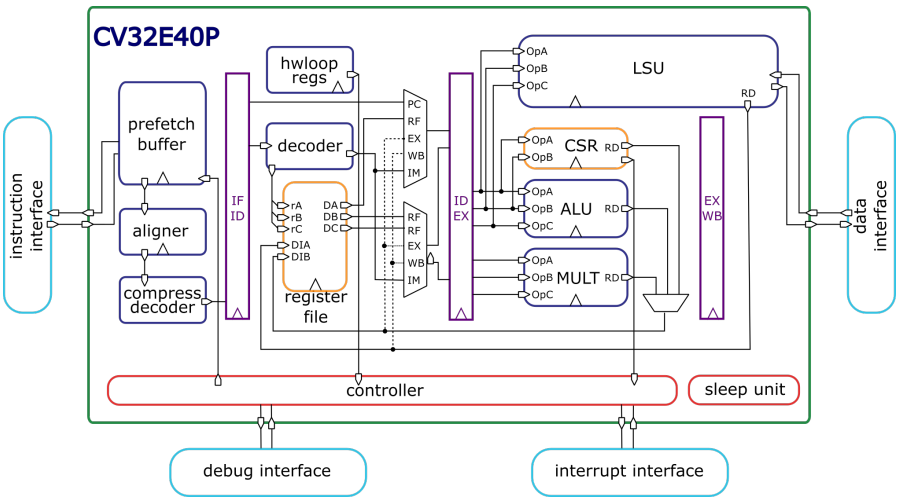


Figura 66: CV32E40P core

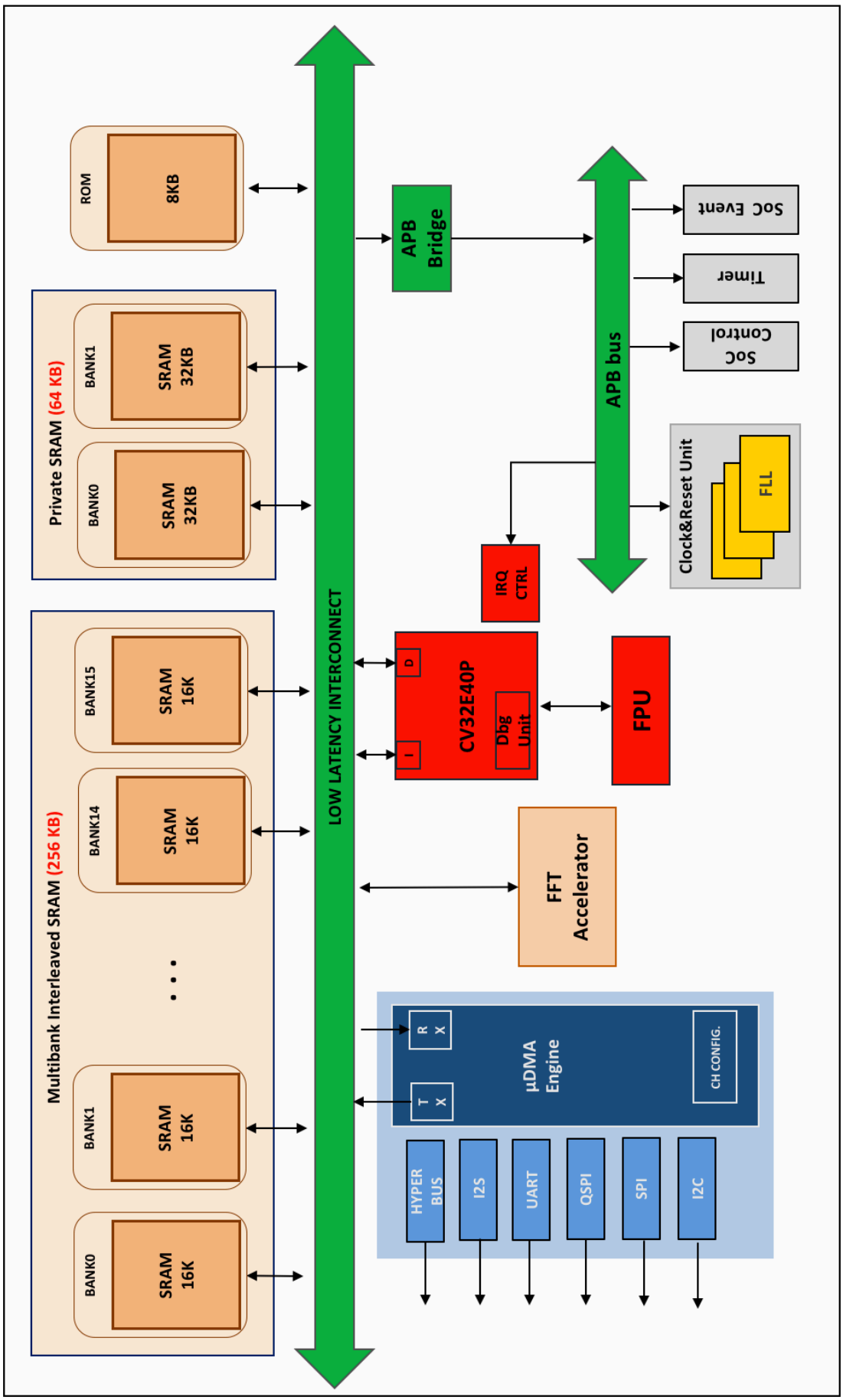


Figura 67: Architettura di Echoes

Il flusso di sviluppo ha seguito i seguenti step fondamentali per la progettazione e realizzazione di un circuito integrato che in generale è diviso in due fasi distinte del design:

- Front-End
- Back-End.

La prima fase di Front-End è stata ampiamente descritta con l'implementazione della TDM, e come è evidente ha un approccio logico al design in cui si definiscono le specifiche funzionali dell'architettura che vengono implementate in RTL, validate tramite il tool QuestaSim ed infine sintetizzate per ottenere una netlist di gates specifici. Nella seconda fase di Back-End che vedremo fra poco invece, l'approccio è più fisico al design, si ha a che fare infatti con specifiche fisiche delle celle elementari con cui l'intero design viene realizzato su silicio. Anch'esso è composto da diverse fasi che guidano i designer alla realizzazione del chip e alla conclusione del processo di tapeout.

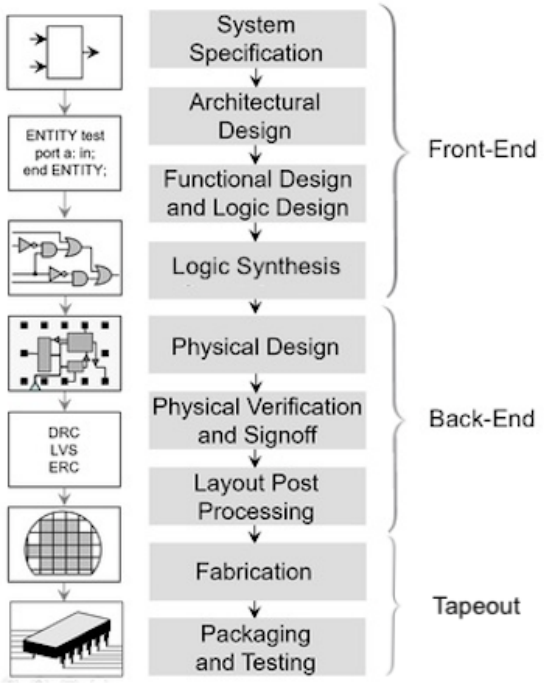


Figura 68: Flusso realizzazione IC

4.1 Sintesi RTL

Il processo di sintesi è un processo che conclude la fase di Front-End del design in cui si ha a che fare con un approccio puramente logico. Si occupa di convertire l'RTL in una netlist di gates specifici per una tecnologia ed è ottimizzata per soddisfare un insieme di vincoli prestabiliti. Quello che occorre per effettuare il processo di sintesi sono: Design RTL, Standard Cells Library e un set di vincoli.

La netlist di gates è solitamente un file verilog molto grande che contiene solo allocazione e interconnessione tra celle standard (AND, OR, Buffer, Inverter, Flip-Flop, Latch, ecc..). Queste celle sono fornite dalla fonderia in forma di librerie definite Standard Cells Library che contengono tutti i gates che possono essere prodotti in un design completamente digitale per una particolare tecnologia (5nm, 22nm, 65nm, ecc..), inoltre contengono una collezione di files che specificano tutte le informazioni necessarie ai vari tool di Electronic Design Automation (EDA). Queste standard cells rispettano determinate specifiche per essere perfettamente manipolate dagli algoritmi di sintesi e place & route.

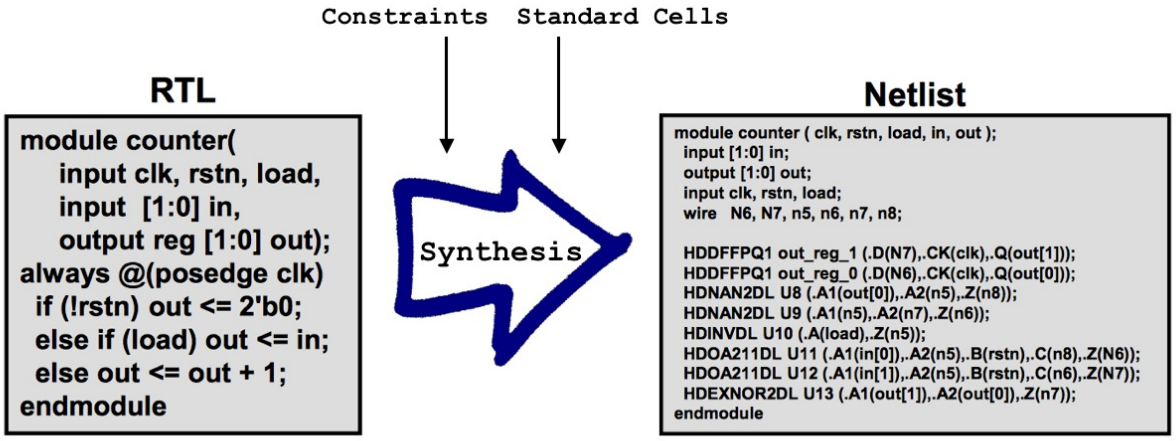


Figura 69: Sintesi RTL

Il processo di sintesi è un processo molto complesso costituito da molte fasi:

1. Syntax Analysis
   Prima di iniziare la sintesi del design RTL si sottopone il codice sorgente System Verilog ad un controllo per verificare che non abbia errori di sintassi e che sia sintetizzabile.

2. Library Definition
   In questa fase viene specificato al sintetizzatore qual è il target tecnologico e dove trovare le librerie delle celle che a loro volta sono suddivise in diversi formati per mettere insieme informazioni di diverso tipo:
   - Specifiche funzionali (.v)
   - Specifiche fisiche e di forma (.gds e .lef)
   - Specifiche di transistor (.spi)
   - Specifiche di Timing/Power (.lib)
3. Elaboration, Binding e pre-mapping optimization
   Durante questo step il tool di sintesi converte l'RTL in una struttura dati di tipo booleana (elaboration) spesso rappresentata tramite tabelle di verità o alberi binari, collega i moduli non booleani alle celle foglia (binding) e infine minimizza la logica con lo scopo di ridurre il numero di attributi della funzione di output (pre-mapping optimization). Questa rappresenta la fase principale del processo di sintesi e il design ottenuto è mappato su porte logiche generiche che prescindono dalla tecnologia.

4. Constraints Definition
   In questa fase il tool inserisce dei vincoli di timing alla logica estratta nello step precedente. L'attività di analisi sul rispetto dei timing della logica è definita come Static Timing Analysis (STA) ed ha come obiettivo quello di verificare che i vincoli di max/min delay siano rispettati per tutti i percorsi del design. Un design è un'interconnessione di gates e net in cui ogni componente necessita di un determinato tempo per svolgere la sua funzione perciò si introducono due tipologie di ritardi: gate delay e net delay.

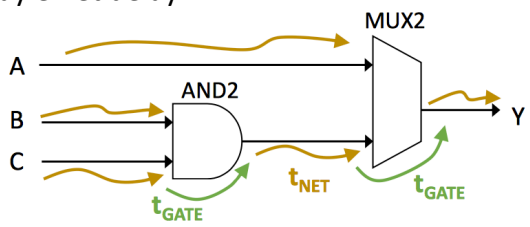


Figura 70: Gate delay e net delay

   Inoltre, dato che la maggior parte del design è costituito da logica sequenziale, bisogna tener presente che anche questi elementi sono soggetti a vincoli di timing definiti come: *setup* time, *hold* time e propagation delay ($t_{CLK->Q}$).

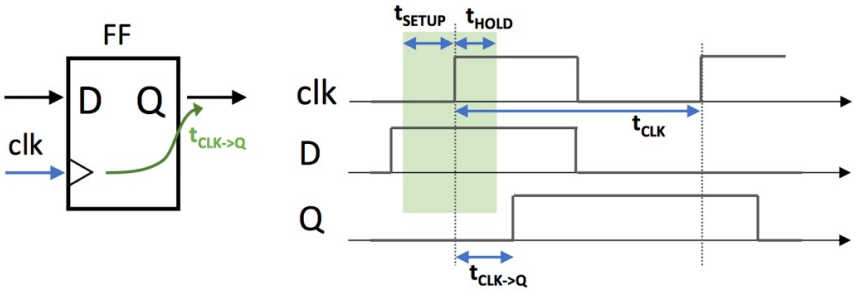


Figura 71: Setup time, Hold time propagation delay

   Il *setup time* impone che il dato arrivi con un certo anticipo rispetto al clock e nel caso di elementi in cascata, la sua propagazione deve essere sufficientemente veloce affinché possa essere correttamente catturato dal prossimo fronte di clock. In STA questo vincolo viene tradotto come *max delay* le cui violazioni sono solitamente il risultato di percorsi molto lenti. Il *setup time* setta inoltre la massima frequenza operativa.

   L'*hold time* impone che il dato non subisca variazioni dopo il fronte di clock e nel caso di elementi in cascata il ritardo del percorso intermedio deve essere sufficientemente lungo per non essere catturato dallo stesso fronte. In STA questo vincolo è tradotto come *min delay* le cui violazioni sono solitamente il risultato di percorsi troppo corti.

   L'imposizione di questi vincoli è effettuata tramite script Synopsys Design Constraints (SDC).

5. Technology mapping
   In questa fase il tool seleziona i gates dalle librerie tecnologiche per implementare il circuito digitale

6. Post-Mapping Optimization
   Il tool dopo aver piazzato le celle tecnologiche cerca di applicare diverse trasformazioni alla logica per cercare di migliorare il costo di realizzazione

7. Report and Export
   In questa fase vengono prodotti i report finali di sintesi e la rispettiva netlist.

Un report molto importante per la fase di Back-End è il report di area. Il sintetizzatore, sulla base delle dimensioni di ciascun gate descritto dalle standard cells library, fornisce ai designer i dettagli riguardo allo spazio necessario sul silicio per poter mappare l'intero design.

Per Echoes è stato utilizzato il tool di sintesi Synopsys che ha eseguito le diverse fasi appena descritte fornendo come output il seguente report di area e la netlist di gate del design che è il punto di partenza per il flusso di Back-End.

```
****************************************
Report : area
Design : echoes
Version: Q-2019.12
Date : Tue Jun 22 14:44:27 2021
****************************************
Library(s) Used:
sc8_cln65lp_base_lvt_tt_typical_max_1p20v_25
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/sc8_cln65lp_base_lvt_tt_typical_max_1p20v_25c.db)

sc8_cln65lp_base_hvt_tt_typical_max_1p20v_25c
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/sc8_cln65lp_base_hvt_tt_typical_max_1p20v_25c.db)

sc8_cln65lp_base_rvt_tt_typical_max_1p20v_25c
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/sc8_cln65lp_base_rvt_tt_typical_max_1p20v_25c.db)

tpdn65lpnv2od3tc1
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/tpdn65lpnv2od3tc1.db)

tsmc65_FLL_tt_typical_max_1p20v_25c
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/tsmc65_FLL_tt_typical_max_1p20v_25c.db)

gtech
(File: /usr/pack/synopsys-2019.12-kgf/syn/libraries/syn/gtech.db)

rf_sp_hdf_hvt_rvt_4096x32m8_tt_1p20v_1p20v_25c
(File: /scratch/smatt/echoes/tsmc65/echoes_top/technology/db/rf_sp_hdf_hvt_rvt_4096x32m8_tt_1p20v_1p20v_25c.db)

Number of ports:                 306649
Number of nets:                  298668
Number of cells:                 174101
Number of combinational cells:   140760
Number of sequential cells:      31244
Number of macros/black boxes:    54
Number of buf/inv:               15844
Number of references:            45

Combinational area:              318428.799193
Buf/Inv area:                    18373.119660
Noncombinational area:           553224.648736
Macro/Black Box area:            1990715.382812
Net Interconnect area:           0.000000

Total cell area:                 2862368.830741
Total area:                      2862368.830741
```

Come è evidente, l'intero design occupa circa 3mm^2 ai quali bisogna aggiungere l'area occupata dal padframe (non considerata in fase di sintesi) che interconnette il design al mondo esterno. Il target è stato quindi fissato a 4mm^2 che è una dimensione ragionevole per un chip single core. Inoltre, le specifiche di area derivano dalla dimensione dei singoli gate che dipendono sia dalla tecnologia e sia dalla fonderia che produce i chip. La tecnologia scelta per Echoes è 65nm di TSMC.

4.2 Implementazione fisica

Per effettuare l'implementazione fisica del design ci si sposta da tool che operano con un approccio logico a quelli che operano con un approccio fisico. Il punto di partenza per il Back-End sono i vincoli sviluppati per il processo di sintesi e la rispettiva netlist di gate.

Il primo step fondamentale è quello del *floorplanning* in cui viene effettuato il mapping dei blocchi che compongono il chip. Nello specifico, si decide il posizionamento dei pad di I/O, il numero e il posizionamento dei pad di alimentazione, la distribuzione dell'alimentazione nel chip e delle sorgenti di clock. L'obiettivo principale del *floorplan* è quello di minimizzare area, ritardi e congestione di routing.

In generale, l'area di un chip è suddivisa nel seguente modo:

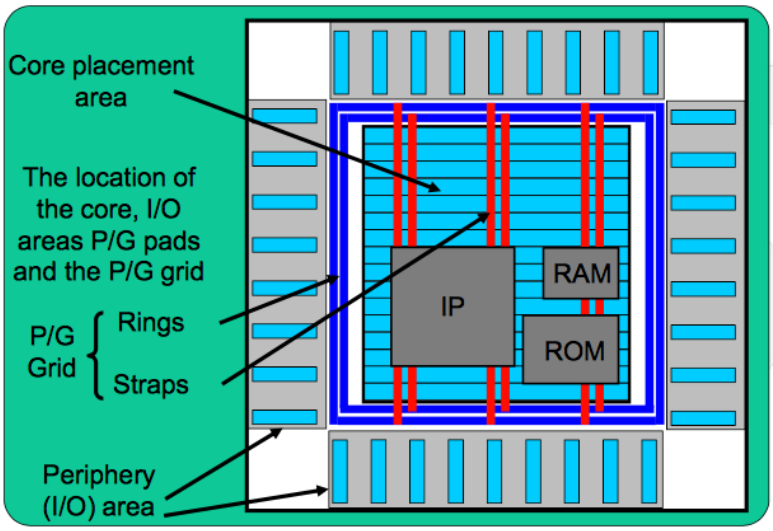


Figura 72: Visione interna di un Chip

La superficie più interna è definita core area e rappresenta la superficie di silicio in cui viene mappato il design (IP, RAM, ROM, PLL), mentre la superficie più esterna definita I/O area è la superficie in cui viene posizionato il pad del chip che interconnette la core area con il mondo esterno. Il numero di pin totali del chip dipende dal tipo di package scelto che a sua volta definisce pin specifici per domini di power specifici: VCC/GND pad e VCC/GND core.

In base al numero di pin al contorno imposto dalle periferiche e dal core di Echoes è stato scelto il package QNF56 che ha un totale di 56 pin. Il mapping dei pin al contorno di Echoes sui pin del package ha prodotto il bonding diagram mostrato in *figura 73* che esplicita e da un'idea di come sarà l'interconnessione finale del chip.

La periferica I2S per esempio avrà i pin dell'interfaccia slave e master sul lato destro del chip rispettivamente a partire dal pin 30 e dal pin 34 del package.

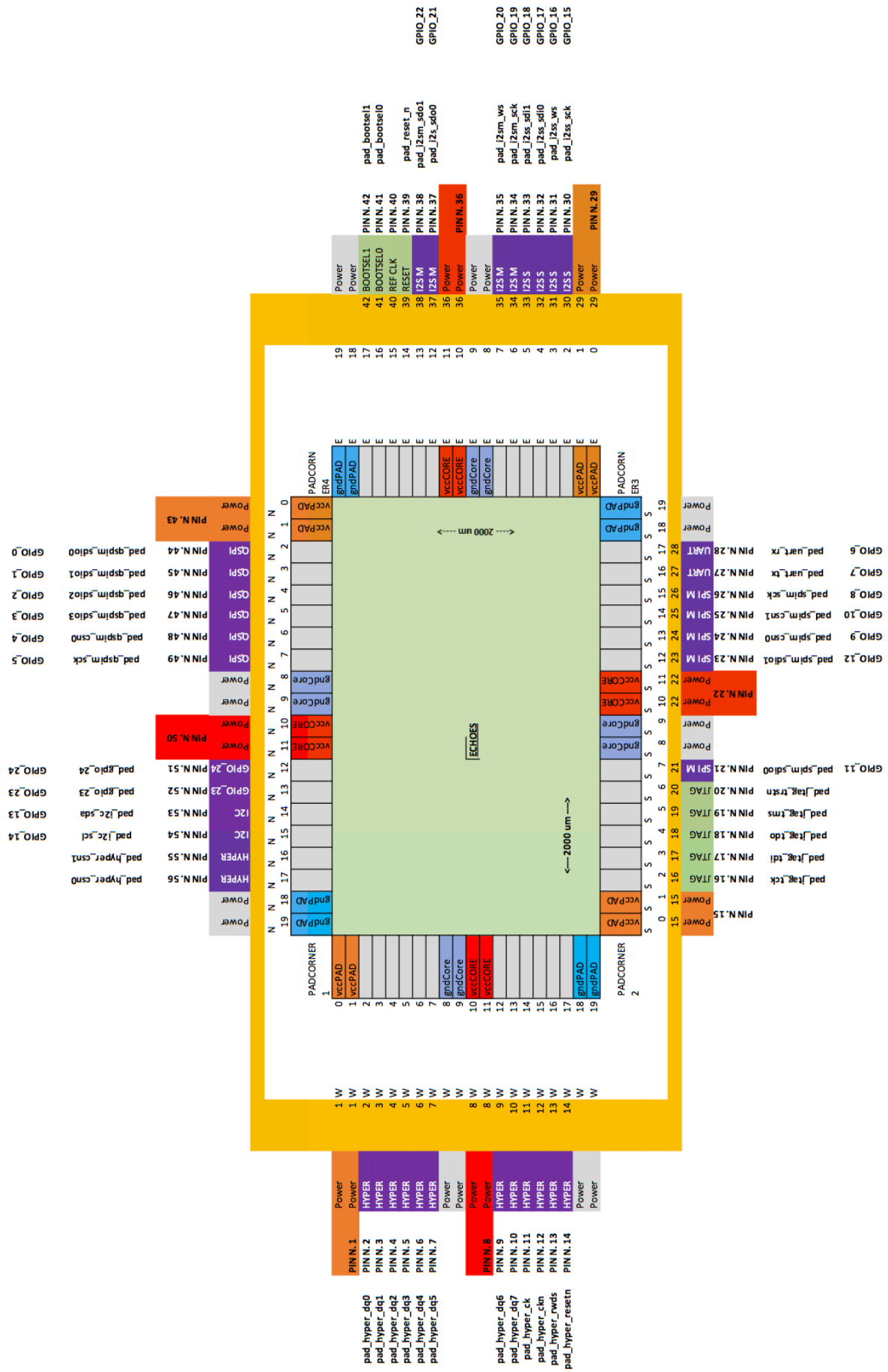


Figura 73: Echoes bonding diagram

Per la fase di Back-End di Echoes è stato utilizzato il tool Innovus che effettua il processo di place and route del design. A partire dalla netlist prodotta da Synopsys, il tool estrae i blocchi che caratterizzano il design. Come è evidente in *figura 74* il design di Echoes è incluso in un unico blocco (viola), mentre per quel che riguarda RAM e PLL vengono estratti dei blocchi a parte poiché sono inseriti nel design come macro, ovvero componenti tecnologici progettati e forniti da TSMC.

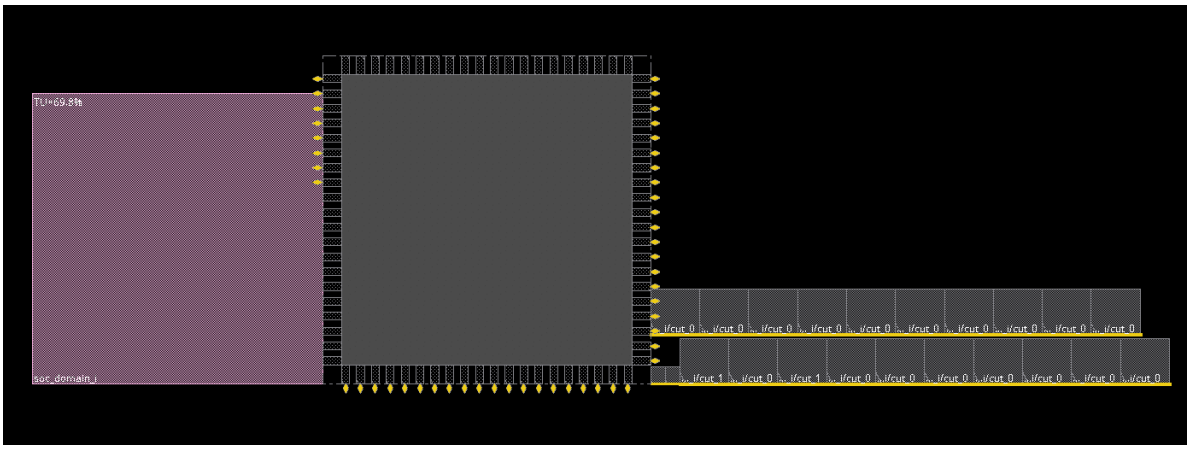


Figura 74: Echoes pre Floorplan

In Echoes ci sono 16 banchi di memoria L2 sui quali sono mappati 16KB di memoria e 2 banchi di memoria privata su ciascuno dei quali ci sono 2 memorie da 16KB.

Per cercare di aiutare il tool ad effettuare un place del design in modo più efficiente affinché non ci sia congestione di routing si è deciso di mappare le memorie nel seguente modo lasciando libera la parte centrale della core area in cui verrà distribuito ed interconnesso il design (*figura 75*). A tal proposito, il numero di interconnessioni che può essere mappato all'interno di una certa area è limitato dal numero di metal layer disponibili, dallo spessore delle interconnessioni e il minimo distanziamento richiesto tra le interconnessioni. In tecnologia 65nm, TSMC prevede 9 livelli di metal e le celle del design vengono mappate solo nel layer più basso ME1, gli altri invece vengono utilizzati solo per il routing e distribuzione di power. La connessione tra due livelli differenti è effettuata tramite le *via* che sono delle connessioni verticali tra layer adiacenti.

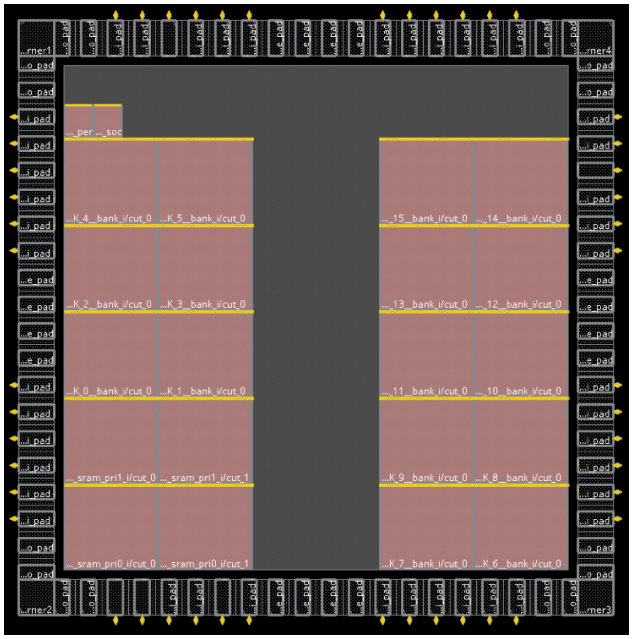


Figura 75: Floorplan Echoes

Come è evidente in *figura 75* le memorie non sono completamente attaccate fra loro e sono circondate da un sottile spazio (*halo*) riservato all'interconnessione delle memorie con il design e alle linee di alimentazione. I blocchi più piccoli situati sopra le macro delle memorie sono i due PLL che generano rispettivamente il *system clock* e il *peripheral clock*. Fino ad ora, nella descrizione RTL e in tutto il flusso Front-End non si è preso in considerazione il fatto che gli elementi del design devono essere alimentati per poter funzionare. Le celle tecnologiche infatti, prevedono connessioni di VDD e GND, bisogna quindi fare in modo che il chip sia completamente alimentato e che ogni regione abbia una tensione operativa stabile. *In figura 76*, notiamo che l'area del core è circondata da un anello di alimentazione (*power ring*) costituito da due linee metalliche (una per VDD e una per GND/VSS) che distribuiscono uniformemente l'alimentazione intorno al chip. Per lasciare spazio al power ring, è necessario quindi lasciare una certa distanza tra I/O e core. Le celle standard inoltre, sono progettate in modo tale che, se poste una accanto all'altra, i loro pin di VDD e GND possono essere collegati con una linea metallica orizzontale ciascuna. Queste linee metalliche di alimentazione definite *power stripes* sono relativamente strette (0,3 µm nella tecnologia 65nm) e ricoprono tutta l'area del core formando una griglia di alimentazione situata negli ultimi due strati metallici per connettersi al *power ring*. Da qui viene distribuita verticalmente l'alimentazione delle celle che si trovano nel primo livello.

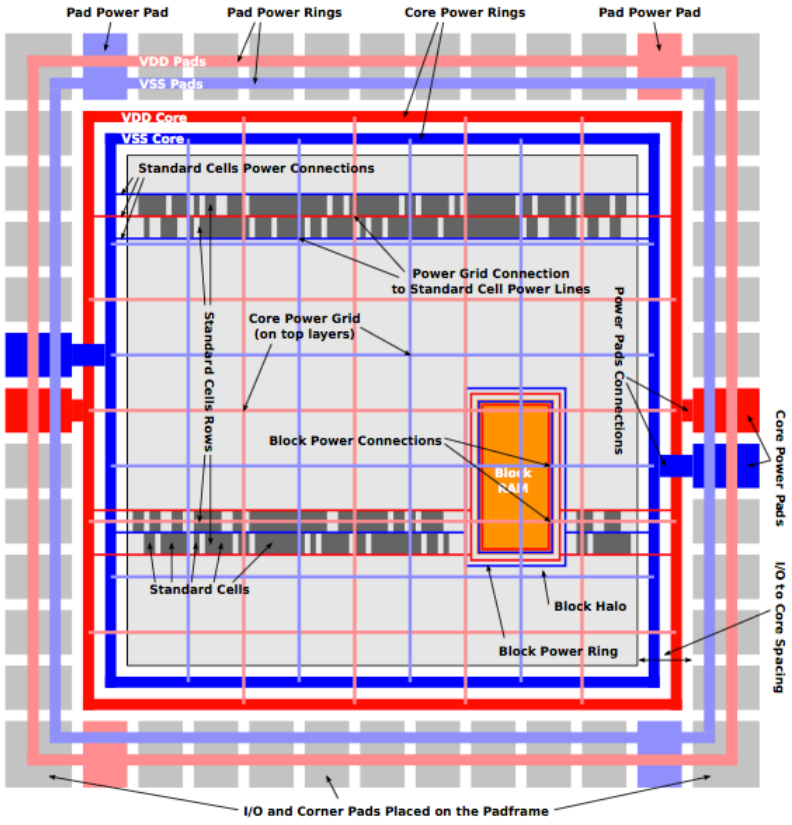


Figura 76: Power Plan

Al termine del *power plan*, come è mostrato in *figura 77* il chip è ricoperto da una griglia che distribuisce alimentazione su tutta la sua superficie.

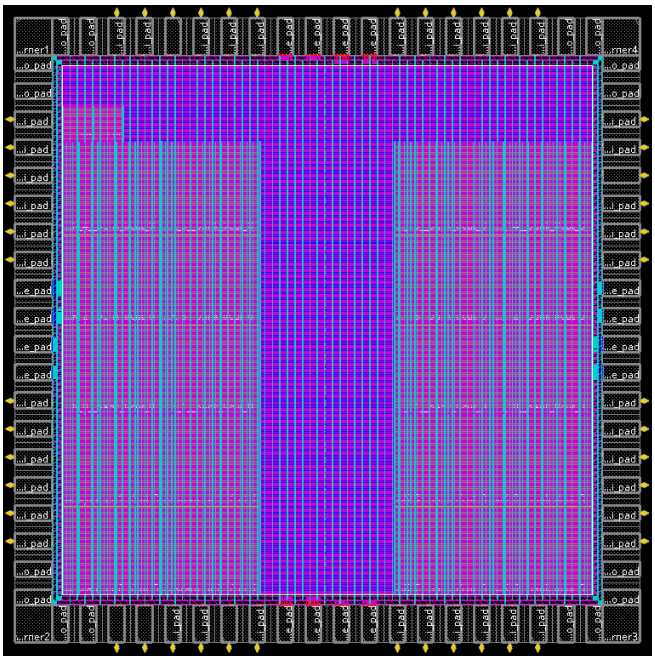


Figura 77: Power Plan Echoes

Solitamente, come è stato precedentemente accennato, gli MCU sono dotati di diverse power mode (LOW POWER, SLEEP, STANDBY, STOP) col fine di essere power efficient. Nel caso di Echoes invece, dato che si tratta di un MCU progettato per fare streaming, è stata definita una sola *power mode* RUN.

In relazione alle modalità operative di power, è necessario ricordare che i chip moderni sono realizzati in tecnologia CMOS che offre maggiore resistenza alle variazioni e ha una leakage power idealmente nulla dovuta alla complementarità della rete di Pull-Down (N-MOS) e Pull-Up (P-MOS). I tre fattori che contribuiscono alle variazioni di performance di un transistor sono: variazioni di processo P, tensione operativa V e temperatura di esercizio T.

Le variazioni nei parametri di processo P possono essere legate alla densità di concentrazione delle impurità, spessori di ossido e profondità di diffusione. Questo introduce variazioni nella resistenza del foglio e nei parametri del transistor come le variazioni della tensione di soglia. In relazione alla tensione operativa V, si ricorda che le prestazioni di un transistor dipendono dall'alimentazione che influisce sul ritardo di una cella. Infine, l'ultimo fattore che contribuisce a modificare le performance di un transistor è la variazione di temperatura che a sua volta dipende dalla tensione operativa, dalla switching activity e dalla capacità di dissipare potenza. Sulla base di queste considerazioni, le variazioni di PVT portano il transistor ad operare su diversi corner che identificano le diverse performance ottenibili e vengono identificati tramite nomi convenzionali di 2 lettere in cui la prima è associata all'NMOS e la seconda al PMOS. Ci sono 5 possibili corner così definiti: Slow-Slow (SS), Fast-Slow (FS), Fast-Fast (FF), Slow-Fast (SF), Tipical-Tipical (TT).

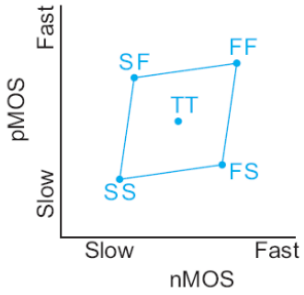


Figura 78: CMOS corners

Le librerie tecnologiche forniscono per ogni cella diverse implementazioni che operano su diversi livelli di tensione e che hanno quindi caratteristiche fisiche differenti. Inoltre, per ciascuna sono definite le proprietà funzionali della cella nei diversi corner che vengono utilizzate per le simulazioni del circuito e la verifica delle performance ottenibili stabilendo il range operativo ottimale del chip, che nel caso di Echoes è 200MHz a 1.20V in TT.

Il processo di Placement mappa ogni singola cella del design su ME1 cercando di rispettare i vincoli di timing, area e alimentazione. Come mostrato in *figura 79* il placement del design è effettuato nello spazio interposto fra le macro delle memorie e dei PLL.

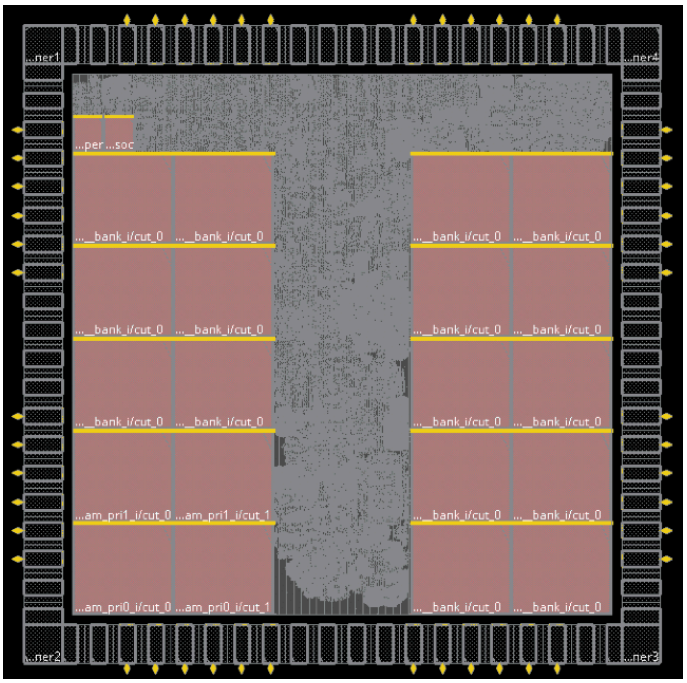


Figura 79: Placement Echoes

La verifica dei vincoli di timing fino ad ora è stata effettuata prendendo come riferimento un clock ideale, a questo punto quel che viene fatto è dotare il chip di un albero di clock che interconnette tutte le celle sequenziali del design. Questo passaggio è definito Clock Tree Synthesis (CTS) e il tool che ne esegue il routing si occupa di preservare l'integrità del segnale di clock proveniente dal PLL affinché non subisca disturbi e venga propagato rispettando i vincoli di max/min delay. Inoltre, l'obiettivo del CTS è quello di minimizzare il più possibile gli skew, ovvero la differenza nel tempo di arrivo tra diversi elementi sequenziali: tutti gli elementi sequenziali dovrebbero ricevere il segnale di clock nello stesso istante.

L'ultimo step è quello del routing in cui vengono interconnesse tutte le celle del design.

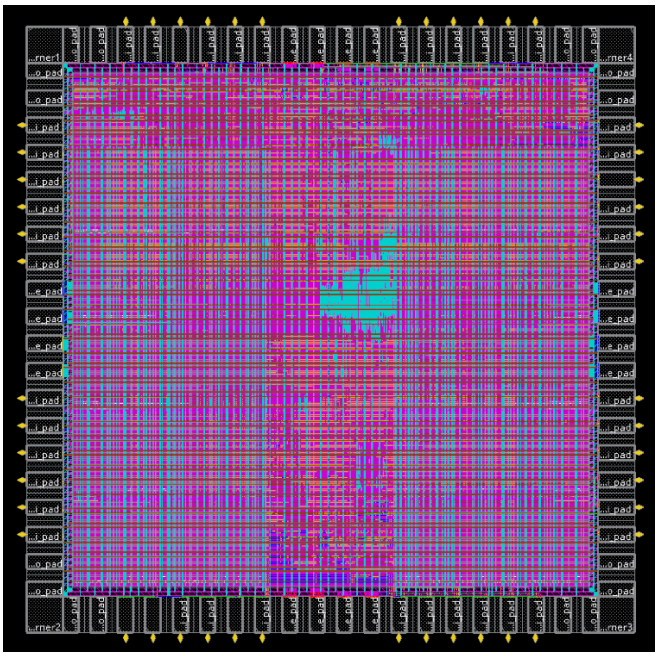


Figura 80: Echoes routing

A questo punto il design del chip è quasi completato e bisogna verificare che il layout prodotto corrisponda alla netlist iniziale e che soddisfi i requisiti di produzione.

L'obiettivo del processo di Back-End è quello di convertire la netlist ottenuta dal processo di Front-End in un layout fisico che contiene essenzialmente per ciascun layer le informazioni sulle geometrie del design. Queste informazioni anche definite come "maschera" vengono utilizzate dai produttori di IC come stampo per la produzione del chip. Sostanzialmente viene mandato in produzione un grosso database in cui sono definite per ogni livello tutte le geometrie del chip. I produttori di IC stabiliscono diverse regole per la produzione di ciascun layer e garantiscono di essere in grado di poter produrre un IC che funziona solo se queste sono rispettate rigorosamente. Queste regole sono conosciute come regole di design. Il processo di Design Rules Constraints (DRC) è un processo importantissimo in cui tutto il design viene verificato per mettere alla luce eventuali violazioni.

Come è stato menzionato, i produttori di IC ricevono solo un database con tutte le informazioni geometriche del design e prima di inviare questo database è importante verificare che l'insieme di poligoni prodotti dal tool Innovus rappresenti fedelmente il circuito digitale. Questo processo è definito come Layout Versus Schematic (LVS) ed il tool che lo esegue effettua principalmente due operazioni: crea una netlist di transistor a partire dal design fisico e la confronta con la gate level netlist che è la base del design. A questo punto, se le due netlist coincidono il database che viene inviato ai produttori di IC rispecchia il design, altrimenti no. Nel nostro caso Echoes ha rispettato i vincoli ed è stato inviato a TSMC per la produzione, ora attendiamo l'arrivo del chip per effettuare i test finali su silicio e scoprire come suona.

Segue il design finale di Echoes e un riassunto delle principali caratteristiche.

Figura 81: Echoes

Applicazione	Audio Processing	
Tecnologia	65nm	
Produttore	TSMC	
Package	QFN56	
Area	$4mm^2$	
Numero Gate	2600 kGE	
Memoria	256KB RAM – 8KB ROM	
I/O	• QSPI (GPIO 0-5) • UART (GPIO 6-7) • SPIM (GPIO 8-12) • I2C (GPIO 13-14)	• I2S (GPIO 15-22) • HYPERBUS • GPIO23 • GPIO24
Alimentazione	1.2 V	
Power	106 µW @ 1MHz	
Clock	166MHz in SS - 200 MHz in TT	

Bibliografia

[1] L. Turchet, G. Fazekas, M. Lagrange, H. S. Ghadikolaei e C. Fischione, " The Internet of Audio Things: State of the Art, Vision, and Challenges", IEEE Internet Of Things journal, vol. 7, no. 10. https://ieeexplore.ieee.org/document/9099251

[2] J. Lewis, "Common Inter-IC Digital Interfaces for Audio Data Transfer", Technical Article MS-2275. https://www.analog.com/media/en/technical-documentation/technical-articles/ms-2275.pdf

[3] Freescale Semiconductor, "Low Power Stereo Codec with Headphone Amp", Document Number SGTL5000 Rev. 6.0, 11/2013. https://www.nxp.com/docs/en/data-sheet/SGTL5000.pdf

[4] Texas Instruments, "PCMD3180 Octal-Channel, PDM Input to TDM or I2S Output Converter", Document Number SBSA14, May 2020. https://www.ti.com/lit/ds/symlink/pcmd3180.pdf

[5] Texas Instruments, "TLV320AIC3107 Low-Power Stereo Codec With Integrated Mono Class – D Amplifier", Document Number SLOS545D,December 2014.https://www.ti.com/product/TLV320AIC3107

[6] U. Agarwal, N. Kekre "Using TLV320AIC3x Digital Audio Data Serial Interface With Time-Division Multiplexing Support", Application Report Texas Instruments, Document Number SLAA311, July 2006. https://www.ti.com/lit/an/slaa311/slaa311.pdf

[7] PULP Platform. https://pulp-platform.org/projectinfo.html

[8] PULPissimo Datasheet. https://github.com/pulp-platform/pulpissimo/blob/master/doc/datasheet/datasheet.pdf

[9] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, e L.Benini "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices", IEEE Transactions on very large scale integration (VLSI) systems, vol. 25, no. 10, OCTOBER 2017. https://ieeexplore.ieee.org/abstract/document/7864441

[10] PULP Datasheet. https://github.com/pulp-platform/pulp/blob/master/doc/datasheet.pdf

[11] A. Pullini, D. Rossi, G. Haugou e Luca Benini, "µDMA: An autonomous I/O subsystem for IoT end-nodes", International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017. https://ieeexplore.ieee.org/document/8106971