
# UNIVERSITÀ DEGLI STUDI DI PADOVA

**Dipartimento di Fisica e Astronomia "Galileo Galilei"**

**Corso di Laurea in Fisica**


**Tesi di Laurea**

# Lettura veloce con FPGA del sensore a pixel ALPIDE per la realizzazione un tracciatore compatto per particelle cariche


**Relatore**

**Prof. Gianmaria Collazuol**

**Laureando**

**Andrea Nicolai**

**Mat. 1103172**



**Anno Accademico 2018/19**

# Indice

**Introduzione** **v**

**1 Sensori a pixel di Silicio** **1**
1.1 Giunzioni per rivelare particelle cariche . . . 1
1.1.1 Giunzioni p-n . . . 2
1.2 MAPS . . . 2
1.3 Energia persa da particelle cariche . . . 3
1.4 Ricostruzione . . . 4

**2 Telescopio al Silicio** **7**
2.1 Apparato sperimentale . . . 7
2.1.1 APiX . . . 7
2.2 Scattering multiplo . . . 8
2.2.1 Numero di sensori . . . 8
2.2.2 Lunghezza dell'apparato . . . 8
2.2.3 Sistema di trigger ed acquisizione . . . 8

**3 ALPIDE** **11**
3.1 L'architettura . . . 11
3.2 Hardware . . . 12
3.2.1 Il front-end analogico . . . 13
3.2.2 Il circuito digitale del pixel . . . 13
3.3 Alcune caratteristiche del sensore . . . 14
3.3.1 Soglia e rumore . . . 14
3.3.2 Efficienza . . . 15

**4 Lettura Dati** **17**
4.1 Interfacce di controllo . . . 17
4.1.1 L'operazione di lettura . . . 18
4.2 Arduino master . . . 21
4.2.1 Arduino slave come simulatore ALPIDE . . . 21
4.3 FPGA . . . 23
4.4 FPGA MASTER $\leftrightarrow$ Arduino SLAVE . . . 23

**5 Readout del chip ALPIDE con Arduino** **27**
5.1 Setup sperimentale . . . 27
5.2 Analisi dati . . . 27
5.3 Scattering . . . 29

**6 Conclusioni** **31**

**Appendice** **33**
.1 FSM1 . . . . . 33
.2 FSM2 . . . . . 35

**Bibliografia** **39**

# Introduzione

In questo lavoro di tesi abbiamo studiato il readout di un sensore a pixel di Silicio, in particolare di ALPIDE, chip sviluppato al CERN per l'upgrade di ALICE. Lo scopo finale è quello di costruire, con 4 di questi rivelatori, un telescopio per tracciare particelle cariche con alta risoluzione spaziale.
Nella tesi ci siamo concentrati inizialmente sul readout tramite microcontrollore (Arduino UNO) e poi tramite FPGA, fungenti da master, utilizzando un altro Arduino UNO come simulatore di ALPIDE e quindi come slave. I linguaggi di programmazione funzionali allo scopo sono stati rispettivamente il C++ per il microcontrollore e il VHDL per l'FPGA.
Il lavoro è stato concluso con un'acquisizione dati tramite ALPIDE in presenza di una sorgente $\beta$ e con diverse configurazioni del setup sperimentale, in modo tale da verificare gli effetti dello scattering.

# 1. Sensori a pixel di Silicio

L'oggetto della tesi è lo studio di sensori a stato solido dove sia il sensore che l'elettronica di readout del segnale sono accoppiati in uno stesso blocco monolitico di Silicio, in modo tale da ottenere una maggiore granularità ed uno spessore minore. Tali sensori vengono chiamati comunemente *Monolithic Active Pixel Sensors* (MAPS).

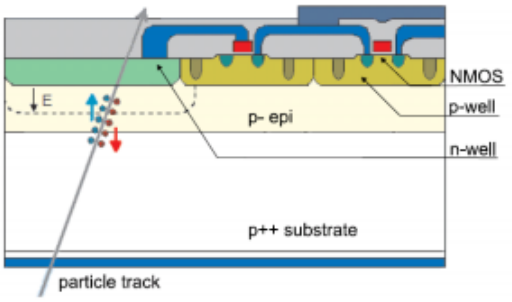



Figura 1.1: Cross section di un detector di tipo MAPS

In particolare è stato studiato il readout del sensore ALPIDE, sviluppato al CERN per l'upgrade dell'ALICE Inner Tracking System cominciato nel 2012.

## 1.1 Giunzioni per rivelare particelle cariche

Per rivelare particelle cariche si utilizzano giunzioni di semiconduttore poiché il band gap $E_g$ è molto piccolo (nel Si $E_{gap} \approx 1.12eV$). Il *band gap* è l'energia richiesta per portare un elettrone in banda di conduzione dalla banda di valenza, creando così una coppia $e^- h^+$.

Tuttavia ciò comporta che ci siano molti $e^-$ in banda di conduzione e, in presenza di un campo elettrico $E_{field}$ sufficientemente piccolo, si genera una grande corrente detta *di buio* che non permette di vedere il segnale. Per ovviare al problema si utilizzano giunzioni ricavate tramite un processo di drogaggio: aggiungendo atomi pentavalenti (donatori) al Silicio si ottiene un Silicio detto n-type, mentre utilizzando atomi trivalenti (accettori) si ottiene un Silicio p-type. Un eccesso di portatori di carica, cioè un aumento degli $n_i$, cambia significativamente le proprietà del materiale, così come la combinazione di regioni con differenti concentrazioni e/o tipo di doping permette di creare strutture con caratteristiche particolari.
Applicando un potenziale alle due regioni della giunzione dopate $n$ e $p$, si ottiene un campo elettrico insieme ai seguenti effetti:

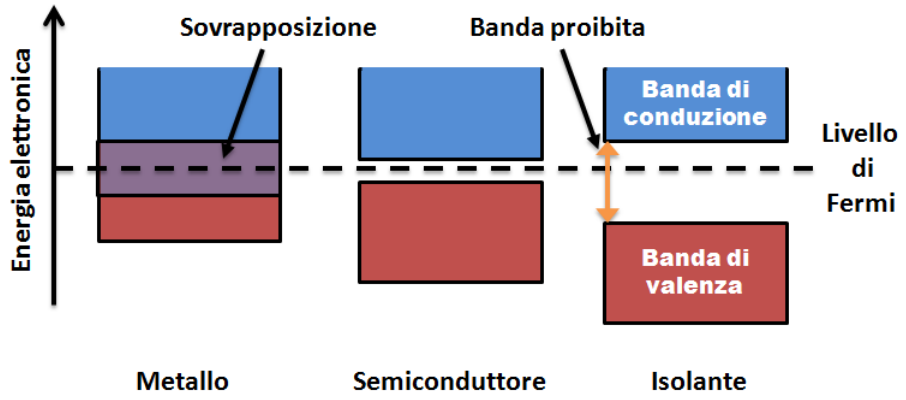



Figura 1.2: Struttura di bande di conduzione e di valenza per diversi materiali

- si viene a creare una zona svuotata in cui, non essendoci $e^-$ in conduzione, non c'è corrente di buio;
- si evita la ricombinazione fra $e^-$ e $h^+$;
- si fanno driftare gli elettroni e le lacune per generare corrente di segnale.

### 1.1.1 Giunzioni p-n

Una di queste strutture con caratteristiche particolari è detta *giunzione p-n*: la differenza di concentrazione di elettroni e lacune nelle due aree messe a contatto, inizialmente neutre, fa in modo che avvenga la diffusione degli elettroni verso il Silicio p-type che diventerà carico negativamente e, viceversa, delle lacune verso il lato n-type che si caricherà positivamente. La differenza di potenziale venutasi a creare si chiama *potenziale di built-in* $V_0$, mentre, all'equilibrio, la zona di contatto è priva di portatori di carica e sarà chiamata *regione svuotata*. Si possono stimare allora:

$$V_0 = \frac{kT}{q} \ln \sqrt{\frac{N_D N_A}{n_i^2}}$$

e la capacità di giunzione:

$$C_j = A \sqrt{\frac{e\varepsilon}{2(V_0 + V)} \frac{N_A N_D}{N_A + N_D}}$$

dove $N_A$, $N_D$ sono le concentrazioni degli accettori e donatori, $\varepsilon$ è la costante dielettrica relativa e $A$ è l'area della giunzione.

## 1.2 MAPS

Un MAPS consiste in 3 strati: un substrato p++ molto dopato che funge da supporto, un sottile strato ($\approx 10\mu m$) $p-$ epitassiale usato come volume sensibile e, sopra di questo, delle strutture n-type e p-type dette *wells*. Le prime fungono da diodi collettori per gli elettroni, mentre le seconde ospitano l'elettronica del sensore. Al diodo viene applicato un reverse bias voltage, indicato con $V_{BB}$, attraverso il substrato per aumentare le dimensioni della regione svuotata, in modo tale da aumentare il volume sensibile.

Tale struttura è ripetuta in tutte le dimensioni fino a creare una pixel matrix. Una particella carica che passa attraverso il detector in Silicio perde energia a causa di ionizzazione, creando così coppie e-h durante il percorso: gli elettroni creati nello strato epitassiale vi rimangono intrappolati non potendo superare le barriere di potenziale formate da $p^-$ - $p^{++}$ e $p^-$ - $p_{well}$. Successivamente possono raggiungere la regione svuotata, tramite duffusione termica, generando così segnale una volta entrati in contatto col

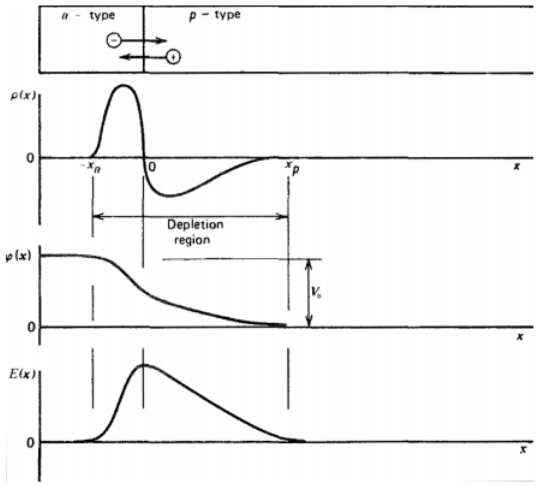



Figura 1.3: Giunzione p-n: la diffusione dei portatori di carica dà luogo ai seguenti profili di densità di carica $\rho(x)$, potenziale $\phi(x)$ e campo elettrico $E(x)$

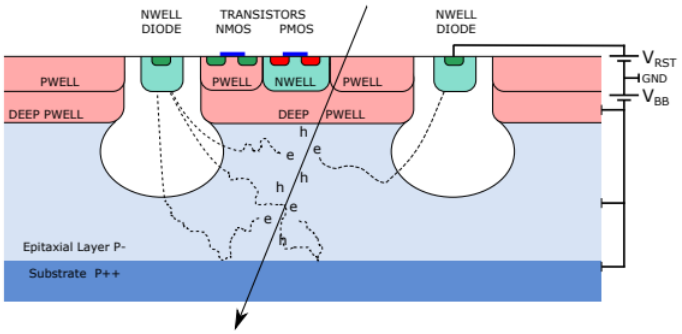



Figura 1.4: Cross section di un singolo pixel

diodo collettore, oppure in minima parte ricombinarsi. Il segnale generato dipende dalla capacità del diodo $C_p$ che a sua volta è funzione della capacità di giunzione $C_j$ e la capacità parassitica del circuito di lettura dei pixel. Un potenziale di reset è applicato mantenere la capacità di input ad un livello fissato di riferimento per poter operare. Un evento causa scarica di tale capacità di una quantità:

$$\Delta V = \frac{Q_{coll}}{C_p} \tag{1.1}$$

dove $Q_{coll}$ è la quantità di carica accumulata. Essa è calcolata integrando, per tutta la durata del segnale, la corrente indotta sull'$m-esimo$ elettrodo data dalla formula del teorema di Ramo:

$$i_m = -qv(t)E_{rip}$$

con $q$ carica della particella che si muove ad una velocità $v(t)$ vicino l'elettrodo, mentre $E_{rip}$ è detto campo di *ripesamento*: viene posto al potenziale $V = 1V$ solo l'$m-esimo$ elettrodo e tutti gli altri a 0. Per massimizzare il segnale la capacità $C_p$ deve essere la più bassa possibile: ciò può essere ottenuto sia ottimizzando il circuito di lettura, variando così $C_r$, mentre per diminuire $C_j$ si polarizza inversamente il collection diode e si presta una particolare attenzione al suo design.

## 1.3 Energia persa da particelle cariche

Lo *stopping power*, cioè l'energia mediamente persa da una particella carica non relativistica in un materiale, è dovuto principalmente ad ionizzazione e all'eccitazione dei livelli di valenza atomici ed è

descritto dalla formula di Bethe-Block:

$$-\frac{dE}{dx} \approx 4\pi r_e^2 m_e c^2 z^2 \rho Z \frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 T_{max}}{I^2} - \beta^2\right] \tag{1.2}$$

dove $r_e$ e $m_e$ sono rispettivamente il raggio classico e la massa dell'elettrone, $z$ è la carica della particella, $\rho = \frac{N_A}{A}$ è la densità del materiale, $T_{max}$ è la massima energia cinetica che può essere impartita ad un elettrone libero in una singola collisione, $I$ è il potenziale di ionizzazione della struttura atomica del bersaglio. Nel lavoro di tesi ci concentreremo su particelle poco ionizzanti (*MIP*).

La PDF che descrive le fluttuazioni di energia persa dipende dallo spessore del rivelatore: per spessori moderati, per i quali l'energia persa è superiore alle metà dell'energia iniziale, seguirà un andamento gaussiano; per spessori inferiori ai 300$\mu$m ma superiori ai 160$\mu$m, invece l'energia persa è descritta dalla distribuzione di Landau, mentre per spessori ancora minori tale modello fallisce.

Per ottenere una stima del numero delle coppie $e^- h$ create nell'attraversamento del Si da una particella MIP, si deve innanzitutto stimare l'energia persa dalla particella carica per ogni $\mu$m di materiale attraversato. Nota la perdita di energia di una particella carica in acqua ($\approx 1MeV/cm$) da (1.2) e nota la densità relativa del Si rispetto all'acqua ($\rho \approx 2.3$), l'energia persa sarà circa $\Delta E \approx 250eV/\mu m$. L'energia necessaria per la creazione di una coppia è invece $E_C \approx 3 \cdot E_{gap} \approx 3.4eV$ (valore specifico per il Si) . Di conseguenza, il numero di coppie create per $\mu m$ di spessore attraversato:

$$N_{coppie} = \frac{\Delta E}{E_C} \approx 80 \tag{1.3}$$

Ci aspetteremo di conseguenza un segnale (1.1) circa pari a:

$$\Delta V = \frac{Q_{coll}}{C_p} = \frac{80eV/\mu m}{C_p}$$

per ogni $\mu m$ di spessore dello strato epitassiale.

## 1.4 Ricostruzione

I MAPS possono essere classificati come *analogici* o *binari*: i primi funzionano leggendo semplicemente il segnale generato da ciascun pixel, i secondi invece restituiscono solo (1) o (0) a seconda che la carica accumulata sia maggiore o minore di una certa soglia. Noi ci occuperemo principalmente di quest'ultimi. Gli osservabili più importanti sono i seguenti:

- **Cluster:** è un insieme di pixel adiacenti nel quale la quantità di carica accumulata è superiore ad una certa soglia. La sua *dimensione*, o *molteplicità*, è il numero di pixel contenuti in ciascun cluster.
- **Centro di massa di un cluster:** è definito come

$$(x_{CM}, y_{CM}) = \frac{\sum_i s_i \dot{(}x_i, y_i)}{\sum_i s_i}$$

Dove $x_i$ e $y_i$ sono la posizione dell'i-esimo pixel nel cluster di cui $s_i$ è il segnale (se binario $s_i$ = 1,0).

Una tipica configurazione di apparato per eseguire test in presenza di fascio consiste in un $DUT$ (Device Under Test) posizionato in mezzo alle stazioni di tracciamento e ortogonalmente al raggio di particelle ionizzanti, come mostrato in figura 1.5.

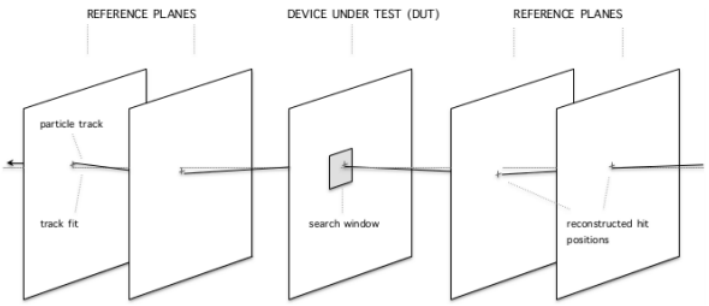



Figura 1.5: Esempio di un setup di telescopio sotto fascio

Il numero e la posizione delle stazioni può variare, ma questi parametri sono ottimizzati per ottenere la migliore risoluzione e per risolvere il problema dello scattering multiplo, come verrà spiegato brevemente in seguito. Gli obiettivi primari per un test in presenza sotto fascio sono: determinare l'efficienza del detector e la risoluzione spaziale di un sensore.

- **Detection efficiency:** per calcolarla si considera una "search window", la cui scelta delle dimensioni varia sia a seconda dell'esperimento, sia del tipo di fascio e della relativa energia. Essa è posizionata sul DUT attorno al punto in cui la traccia ricostruita lo interseca; se si forma un segnale all'interno di questa finestra, la traccia è allora conteggiata. La detection efficiency è definita come:

$$\varepsilon = \frac{k}{n}$$

  con $k$ numero delle tracce conteggiate e $n$ numero di tracce totali. La sua varianza è stimata tramite:

$$\sigma_\varepsilon^2 = \frac{(k+1)(k+2)}{(n+2)(n+3)} - \frac{(k+1)^2}{(n+2)^2}$$

  che, in termini dell'errore sull'efficienza e per $n$ sufficientemente grandi:

$$\sigma_\varepsilon \approx \sqrt{\frac{\epsilon(1-\epsilon)}{n}}$$

- **Risoluzione spaziale:** La risoluzione spaziale è determinata calcolando la varianza $\sigma_{res}^2$ della distribuzione dei residui (differenza fra la posizione di hit ricostruita e quella estrapolata sul DUT) e la varianza $\sigma_{track}^2$, che a sua volta dipende dalla risoluzione intrinseca delle stazioni di tracciamento e da fenomeni fisici, come ad esempio il multiple scattering e gli altri strumenti utilizzati nell'esperimento. Si calcola con la relazione:

$$\sigma_{sp}^2 = \sigma_{res}^2 - \sigma_{track}^2$$

Una delle problematiche principali in cui si può incorrere in un test sotto fascio è quella dello scattering multiplo che produce una deviazione nella traiettoria della particella. L'angolo formato con la direzione antecedente l'urto segue una pdf gaussiana di media zero e sigma:

$$\theta_{RMS} = \frac{13.6[MeV]}{\beta cp} z\sqrt{l[X_0]} \tag{1.4}$$

dove $p$ è il momento, $\beta c$ la velocità e $z$ la carica della particella incidente e $l[X_0]$ è lo spessore del materiale in termini della lunghezza di radiazione $[X_0]$ che, nel caso del Si $\approx 9.8cm$.
Per una particella relativistica ($\beta \approx 1$) e di momento $\approx 1GeV$, come nel caso dei $\mu$ da raggi cosmici,

l'errore introdotto sull'angolo sarà dell'ordine di $1mrad$. Tenendo in considerazione al massimo un solo evento di scattering significativo, 4 stazioni di tracciamento sono più che sufficienti per riuscire a ricostruire la traccia in presenza di sensori a pixel che, a differenza di detectors a strips, non hanno una precisione maggiore su un asse rispetto ad un altro e che per questo possono essere posizionati ortogonalmente alla direzione di incidenza del fascio. Inoltre si ricorda inoltre l'effetto charge-sharing fra pixel adiacenti che va a migliorare la precisione nella ricostruzione delle tracce nei rivelatori analogici.

Un'altra problematica nel processo di misura è legata al cosiddetto $fake - hit$. Esso è definito come il numero di segnali generati per pixel per evento in assenza di stimoli esterni. Può aver due cause: il rumore dovuto all'eccitazione termica, oppure l'RTN (Random Telegraph Noise). Poichè esso dipende sia dalla soglia che dal rumore, specifici per ogni singolo pixel, si stima il fake-hit rate medio di un sensore come il valore di aspettazione della $f(t, n)$:

$$\overline{FHR} = E[f(t,n)] = \iint f(t,n) pdf(t) pdf(n) dt dn$$

dove $pdf(t)$ e $pdf(n)$ sono densità di probabilità della soglia e del rumore e possono essere approssimate entrambe da una gaussiana. Per stimare sperimentalmente il fake-hit rate si manda un numero $N_{trig}$ di triggers al chip senza alcun segnale esterno, registrando il numero di hit del sensore $N_{hit}$.

$$FHR_{sper} = \frac{N_{hit}}{N_{pix} \cdot N_{trig}}$$

dove $N_{pix}$ è il numero di pixel nel sensore.

# 2. Telescopio al Silicio

La tesi si inserisce nell'ambito di un progetto per la realizzazione di un telescopio per particelle cariche ad alta risoluzione spaziale, finalizzato a caratterizzare nuovi sensori a pixel. In questo capitolo illustriamo brevemente i principi su cui si basa il funzionamento del telescopio che abbiamo contribuito a disegnare.

## 2.1 Apparato sperimentale

Un telescopio a pixel è costruito appositamente per ottenere informazioni precise per esperimenti sotto fascio: la traiettoria precisa viene ricostruita utilizzando l'informazione del punto di impatto delle particelle sul $DUT$ (Detector Under Test). Nel nostro caso esso sarà il sensore APiX (*Avalanche Pixel*): altro sensore monolitico a pixel di Si basato e sviluppato in gran parte da INFN, scelto principalmente per l'ottima risoluzione spaziale e temporale.
Il telescopio sarà composto da 4 sensori ALPIDE, dei quali si parlerà più approfonditamente in seguito, posizionati lungo un braccio di una struttura meccanica che li terrà fermi ed eviterà che essi si muovano una volta investiti dalla radiazione. Il DUT sarà posto a metà del telescopio, nello spazio compreso fra la seconda e la terza stazione di tracciamento. Tutti i sensori sono posti in modo tale da essere ortogonali alla direzione di incidenza del fascio. Ci poniamo quindi in un sistema di riferimento nel quale le stazioni di tracciamento del nostro telescopio giacciono sul piano $(xy)$, ortogonalmente alla direzione del fascio $\vec{z}$.

### 2.1.1 APiX

Un esempio di un chip che sicuramente dovrà essere caratterizzato in futuro è APiX (*Avalanche Pixel*). APiX è il primo sensore a pixel che, realizzato su due piani, rileva il passaggio di particelle cariche (*rivelatore Geiger*).Ogni pixel è formato da due rivelatori a valanga allineati verticalmente che, sfruttando la coincidenza fra due eventi contemporanei, riesce a discriminare un segnale generato da un passaggio di particella carica da un segnale cosiddetto 'di buio'.
Un array di pixel 48x16 è stato realizzato come prototipo iniziale e disegnato utilizzando la tecnologia CMOS e infine integrato verticalmente attraverso la tecnica di 'bump-bonding', mentre un singolo pixel, che contiene al proprio interno l'elettronica digitale per immagazzinare e processare un generico evento di coincidenza, ha dimesioni di $50\mu m$x$75\mu m$ con un fattore di riempimento massimo del 51.6%.
La peculiarità di tale sensore è che, grazie alla discriminante della coincidenza fra i segnali generati su entrambi i piani, riesce a distinguere un segnale generato dal passaggio di particella carica da un segnale di buio: cosa altrimenti impossibile utilizzando un singolo rivelatore di tipo Geiger.

## 2.2 Scattering multiplo

La principale fonte di errore nella misura delle tracce in un telescopio è lo scattering multiplo. Altri problemi per esempio legati a fake hits sono di secondaria importanza.

### 2.2.1 Numero di sensori

L'obiettivo dell'esperimento è quello di ricostruire la traccia delle particelle che hanno attraversato il telescopio. Con un ragionamento un po' näif dimostreremo che, volendo noi considerare gli eventi con al massimo uno scattering significativo, è sufficiente che il numero di stazioni sia 4.
Nel caso in cui la particella non subisca scattering, ovviamente, si riesce a ricostruirne la traiettoria senza troppi problemi andando ad analizzare i punti di impatto sui sensori, così come se lo scattering dovesse avvenire sulla prima o sull'ultima stazione. Tuttavia, nel caso ci sia scattering o sulla seconda o sulla terza stazione di tracciamento, il segnale che vedremo considerando la proiezione sul piano $(xy)$ sarà formato da due linee spezzate intersecantisi in prossimità del detector su cui è avvenuto lo scattering. Nel caso in cui tale processo avvenga più volte, 4 stazioni non saranno più sufficienti: per poter individuare una retta sono necessari almeno due punti appartenenti a due detector contigui ed, essendo le rette al massimo due, il quinto segnale rimanente sarà il punto in cui la particella ha subito scattering: in altre parole sarà il punto in cui la traiettoria della particella ha cambiato direzione. Ipotizzando, ad esempio, che la particella scatteri anche una seconda volta, le rette da individuare saranno 3 e non più 2: avremo bisogno di almeno 3 coppie di punti e di conseguenza almeno 6 detectors. Si evince quindi che, nel caso si prenda in cosiderazione al massimo un evento di scattering, il sistema 4 detectors+DUT sia sufficiente.
Durante l'analisi dati e nel processo di ricostruzione della traiettoria, vengono tracciate le rette congiungenti ai punti dei sensori in cui è stato ottenuto il segnale. Se la retta è unica, potremo quindi dire che la particella non ha subito scattering. Se la retta invece è formata da due linee spezzate che si intersecano in una certa regione di spazio, definita con una certa tolleranza, con una certa sicurezza potremo affermare che la particella sarà stata scatterata una volta e in quel punto.

### 2.2.2 Lunghezza dell'apparato

Partendo dalla stima di errore di $\theta \approx 1mrad(1.4)$ ottenuta in precedenza, una volta nota risoluzione del detector ALPIDE, possiamo stimare quale possa essere buona lunghezza del braccio del nostro telescopio perché l'effetto di multiple scattering sia trascurabile. Detta $d \approx 5\mu m$ la risoluzione di ALPIDE ed $L$ la lunghezza del braccio del telescopio, poichè l'angolo si può approssimare come:

$$\theta \approx 1mrad = \frac{d}{L} = \frac{5 \cdot 10^{-6} m}{L}$$

Una volta invertita la formula si stima la lunghezza richiesta: $L \approx 1m$.

### 2.2.3 Sistema di trigger ed acquisizione

Al fine di comunicare all'apparato sperimentale l'inizio di un evento per procedere all'acquisizione dati sono utilizzati degli scintillatori che rilevano il passaggio di particelle cariche. In secondo luogo sono presenti due DAQ distinti e sincronizzati fra loro: uno sul telescopio e uno sul device. Per identificare univocamente il numero dell'evento in entrambi i sistemi di acquisizione si utilizza una parola a 16bit che viene trasmessa attraverso l'elettronica di readout.
Per l'acquisizione e la relativa trasmissione di dati è necessario almeno un centinaio di cicli di clock una volta azionato il trigger (cap. 4.1.1), tempo durante il quale non è possibile acquisire nuovi eventi di hit. Tenendo conto anche delle varie frequenze di clock nominali sia del microcontrollore, sia dell'FPGA

(cap. 4.2.1), è stata stimata una 'frequenza efficace' per ognuno di questi: rispettivamente 100kHz e 100MHz. Di conseguenza, per ciascuna stazione del telescopio e quindi per ogni chip presente, una stima del tempo (*latenza*) necessario per il totale passaggio delle informazioni di hit è: per l'Arduino UNO circa $10ms$ (max 100 hit/s), mentre per la scheda FPGA $10\mu s$ (max 100'000 hit/s). A seconda degli schemi di trigger e readout questa latenza può costituire il tempo morto minimo per il trigger.

# 3. ALPIDE

ALPIDE *ALice PIxel Detector* è un sensore di tipo MAPS, sviluppato al CERN dal 2012 nell'ambito dell'upgrade dell'ITS ($Inner Tracking System$) dell'esperimento ALICE e perfezionato definitivamente, grazie a studi sulle proprietà di collezione di cariche e ottimizzazione del diodo collettore, nel 2016.

## 3.1 L'architettura

ALPIDE è basato su un sensore in tecnologia CMOS TowerJazz a 180nm. Tale tecnologia fornisce la possibilità di posizionare un n-well in cima ad un p-well, così da separare tutti gli n-wells, ad eccezione dei diodi collettori, dallo strato epitassiale che è il volume sensibile. Il chip ALPIDE è prodotto su wafers con uno strato epitassiale posizionato su un substrato di tipo p e caratterizzato da un'alta resistività ($\rho > 1k\Omega \cdot cm$). Lo spessore ottimale, dopo aver testato diversi prototipi, è stato deciso essere di $25\mu m$. Il processo di costruzione TowerJazz consente l'applicazione di un bias negativo al substrato in modo tale da svuotare ancora di più lo strato epitassiale, aumentando inoltre l'intensità e l'estensione spaziale del campo elettrico presente. Di conseguenza la carica elettrica è richiamata al diodo collettore molto più rapidamente, la frazione di carica che vede il *seed pixel* è aumentata, dove per seed pixel si intende il pixel che raccoglie la maggior quantità di segnale, infine aumenta il rapporto $\frac{Q}{C}$(1.1) essendo stata ridotta $C$.

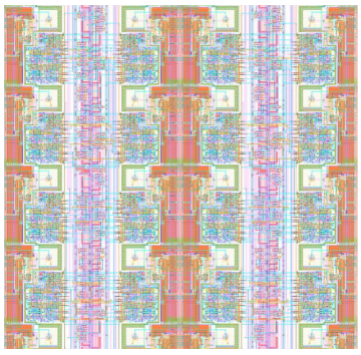


Figura 3.1: Struttura a doppia colonna dei pixel del chip ALPIDE

Ogni pixel ALPIDE contiene un amplificatore, un discriminatore ed un buffer multi-evento. I pixel sono organizzati in doppie colonne, ciascuna delle quali ne contiene 1024, come si vede in figura 3.1. In ogni chip ALPIDE ci sono 512 doppie-colonne che formano una matrice di 1024 colonne e 512 righe. La parte centrale di ogni doppia colonna ospita un circuito che propaga l'indirizzo dei pixel colpiti alla periferia. Non essendo presente alcun segnale di clock internamente al chip, i pixel che non sono stati colpiti non generano alcun segnale: il consumo di energia per pixel è molto contenuto, circa $40nW$.

## 3.2 Hardware

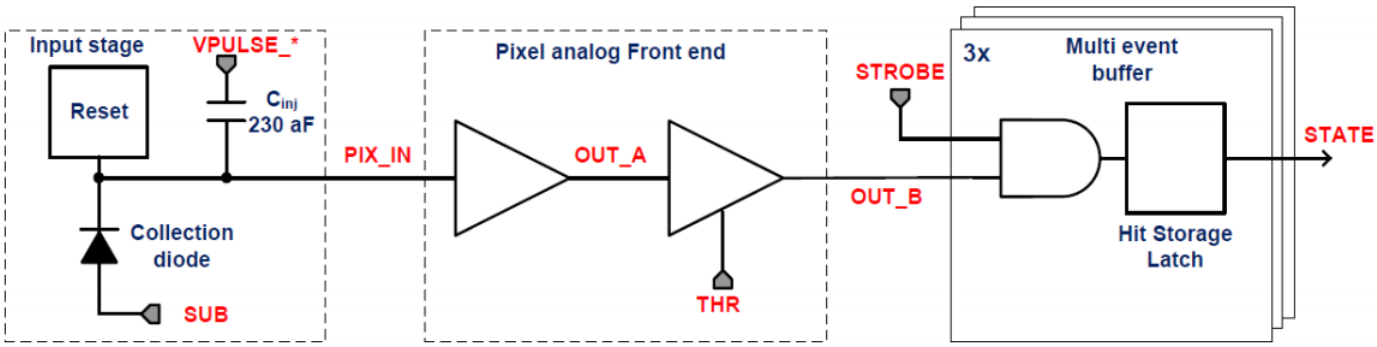



Figura 3.2: Rappresentazione schematica del circuito in pixel per un chip ALPIDE

Nello stadio di input è presente un diodo collettore, un reset continuo del nodo di input e una capacità $C_{inj}$ che serve ad iniettare cariche di prova nel front-end analogico. Collezionata la carica, c'è una rapida caduta di potenziale al nodo di input del circuito di amplificazione che il reset lentamente fa tornare al valore fissato di riferimento (fig. 3.3). Il reset, nel chip finale, è ottenuto con l'uso di un diodo.

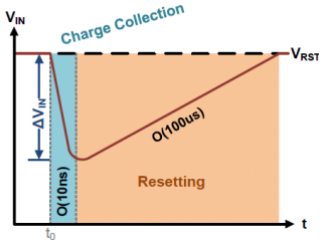



Figura 3.3: Caduta di potenziale e salita a livello di riferimento sul nodo di input

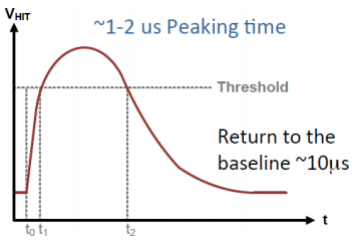



Figura 3.4: Reshaping del segnale sul nodo di input dell'amplificatore

La caduta di potenziale sul nodo di input dell'amplificatore viene rimodellata in un segnale con una durata fino a $10\mu s$ e un picco di $\sim 2\mu s$ (fig. 3.4). Il fine tuning, lo shaping e la discriminazione del segnale *(charge threshold)*, sono variati andando a modificare i voltaggi e le correnti di bias, regolabili grazie ai chip DACs 8-bit *(Digital-to-Analogue Converters)*.

Per tutta la durata del tempo in cui il segnale eccede la soglia, l'output del discriminatore resta alto. Nel caso in cui contemporaneamente il segnale di *STROBE* sia mandato dal trigger comune a tutti i pixel, lo stato di output del discriminatore è salvato in un registro. Ogni pixel è provvisto di 3 registri ciascuno che fungono da buffer multi-evento (fig. 3.5). Un latch nel circuito digitale permette di individuare, e ignorare, un pixel se questo è rumoroso o malfunzionante.

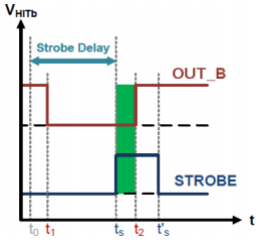



Figura 3.5: Effetto del discriminatore e OUTPUT in presenza di segnale di STROBE

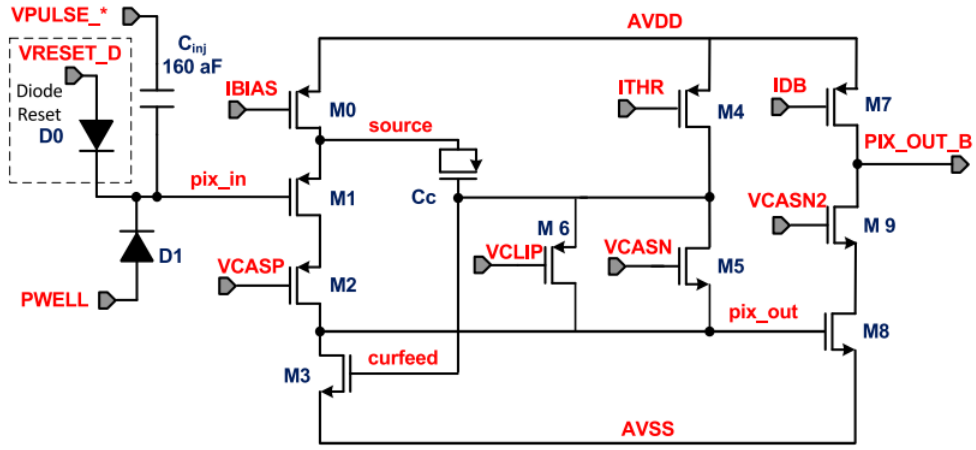



Figura 3.6: Raffigurazione del front end analogico di un chip ALPIDE

### 3.2.1 Il front-end analogico

Il diodo collettore nell'immagine 3.6 è indicato da D1, mentre D0 ha la funzione di riportare il nodo di input ($pix\_in$) ad un valore di riferimento VRESETD.
Una particella che colpisce il detector abbasserà il potenziale di $pix\_in$ dell'ordine di decine di $mV$, aumentando così la corrente attraverso il source follower M1 e facendo in modo che il nodo *source* segua l'input. Di conseguenza la carica immagazzinata nel nodo *source* viene vista dal nodo $pix\_out$. In aggiunta, l'accoppiamento fra il nodo di *source* e quello di $curfeed$ ridurrà la corrente attraverso M3 per l'effetto di un feedback in corrente. Il potenziale $pix\_out$ si alzerà, mettendo in condizione M8 di condurre e, nel caso in cui la carica depositata dalla particella sia superiore ad una certa soglia, il nodo $PIX\_OUT\_B$ raggiungerà potenziale nullo e fungerà da input per il circuito digitale. La soglia viene regolata aggiustando i voltaggi e le correnti di bias, in particolare i segnali $ITHR$, $VCASN$ e $IDB$.

### 3.2.2 Il circuito digitale del pixel

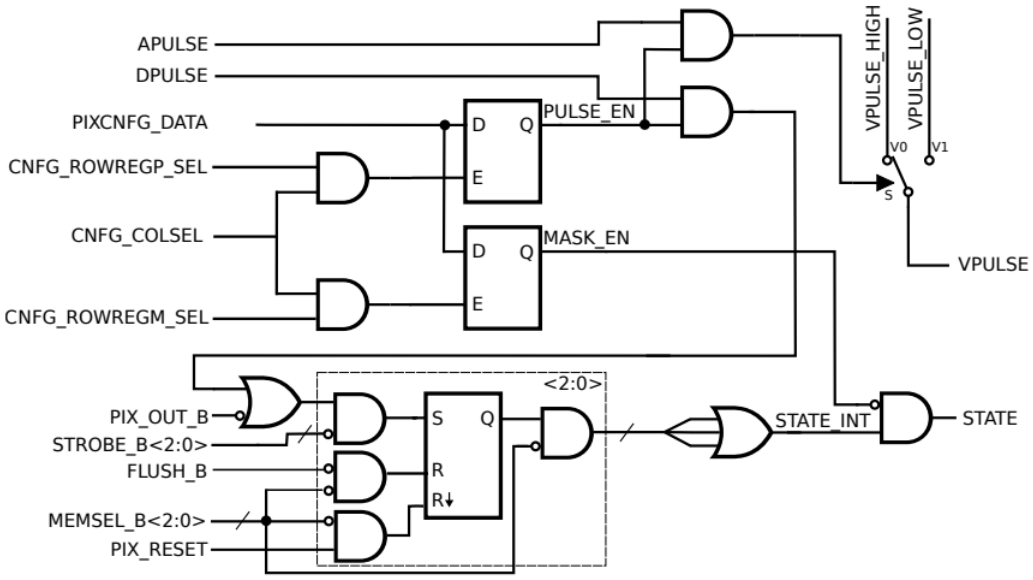



Figura 3.7: Sezione del circuito digitale di un chip ALPIDE

Come si evince dalla fig. 3.7 che rappresenta la sezione del circuito digitale di un ALPIDE, ogni pixel ha al suo interno 3 registri. Ognuno di essi è un latch di tipo SR che registra l'informazione sull'evento legato ad una particella. Il segnale di *Set* è alto nel caso in cui siano presenti contemporaneamente sia il segnale $STROBE\_B$ che $PIX\_OUT\_B$ o, in alternativa $DPULSE$ durante la caratterizzazione del circuito. Un registro può esser selezionato per le operazioni di *read/reset* utilizzando il corrispondente segnale di selezione $MEMSEL\_B$: il reset può avvenire sia a causa di un segnale $PIXRESET$ generato dal Priority Encoder durante il readout, oppure da un $FLUSH\_B$ globale.
Il circuito provvede a due funzioni programmabili: il *masking* ed il *pulsing*. Se il bit di controllo $MASK\_EN$ è 1, lo $STATE$ di output è tenuto a 0 indipendentemente dagli altri segnali: può esser utile nel caso di pixel rumorosi o malfunzionanti. Il pulsing analogico invece consiste nell'iniettare una carica di test attraverso il nodo di input grazie a $C_{inj}$ (fig. 3.6). L'ampiezza del segnale di pulse è la differenza fra $VPLSE\_HIGH$ e $VPLSE\_LOW$.

## 3.3 Alcune caratteristiche del sensore

### 3.3.1 Soglia e rumore

Uno dei parametri fondamentali di funzionamento di un chip è la *soglia* che, come discusso precedentemente, è in funzione dei segnali di bias $ITHR$, $VCASN$ e $IDB$ che sono comuni a tutti i pixel. Per ottenerne una stima si può iniettare $N$ volte una carica di pulse $Q_{inj}$ attraverso la capacità $C_{inj}$ in modo tale che:

$$Q_{inj} = C_{inj}(VPLSE\_HIGH - VPLSE\_LOW)$$

dove $VPLSE\_HIGH$ e $VPLSE\_LOW$ in modulo sono inferiori a 1.8V e sono settati tramite gli 8-bit DACs.
Per ogni evento di test $Q^i_{inj}$, su $N^i_{inj}$ cariche iniettate totali , $N^i_{hit}$ sono superiori alla soglia e quindi generano un segnale. La soglia viene definita come la carica in funzione della quale i pixel almeno il 50% delle volte generano un segnale.

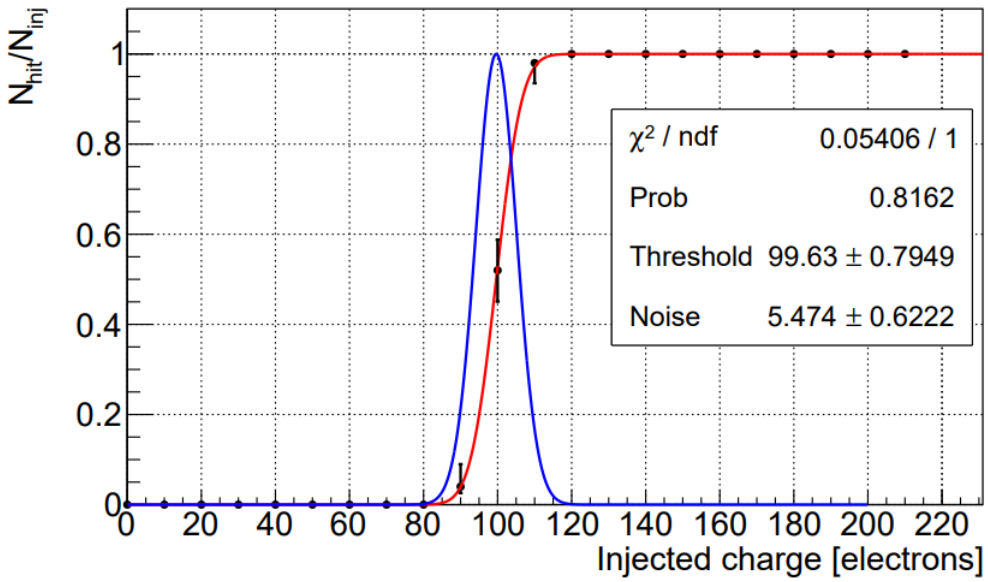



Figura 3.8: S-curve. La linea rossa è la funzione $erf$, mentre in blu la sua derivata, cioè il rumore di un pixel

Nella figura (3.8), detta *s-curve*, è plottato il rapporto $N^i_{hit}/N^i_{inj}$ in funzione della carica iniettata. Inoltre ipotizziamo che il segnale elettronico segua una gaussiana:

$$f(x) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]$$

dove $\mu$ è la soglia e $\sigma$ il rumore di un pixel. Lo stesso procedimento si applica a tutti i pixel: studiando poi la distribuzione delle soglie e del rumore e trovandone media e varianza, si riuscirà a caratterizzare il chip ALPIDE.

### 3.3.2 Efficienza

Una volta noti lo spessore dello strato epitassiale di ALPIDE ($\approx 25\mu m$) e la stima del numero di coppie create per $\mu m$ di spessore (1.3), potremo quindi stimare l'efficienza del sensore. In totale, nel volume sensibile, saranno generate in media $25\mu m \cdot 80\text{coppie}/\mu m \approx 2000$ coppie. Essendo l'estremo superiore della soglia $\approx 120e^-$ (fig. 3.8) e seguendo il processo una statistica poissoniana il cui parametro è $\lambda = 2000$, l'efficienza sarà quindi:

$$\epsilon = 1 - cdf(n = 120, \lambda = 2000) \approx 1$$

L'efficienza sarà allora del 100% per il nostro tipo di sensore: siamo praticamente certi che qualsiasi particella carica che attraversi il nostro chip sarà rivelata e genererà un segnale.

# 4. Lettura Dati

In questo capitolo illustreremo il lavoro principale svolto in questa tesi. Dopo una breve introduzione sulle modalità di lettura di ALPIDE saranno discussi:

- lettura con Arduino
- utilizzo di Arduino come simulatore di ALPIDE
- lettura di ALPIDE (simulato da Arduino) con l'uso di FPGA

## 4.1 Interfacce di controllo

Un diagramma a blocchi semplificato di un chip ALPIDE è rappresentato in fig.4.1, dove sono evidenziati con riquadri colorati i due modi di readout possibili: in blu quello lento *(Control Bus)*, in rosso quello veloce *(Parallel Data & Serial out)*. In questo lavoro noi utilizzeremo il primo.

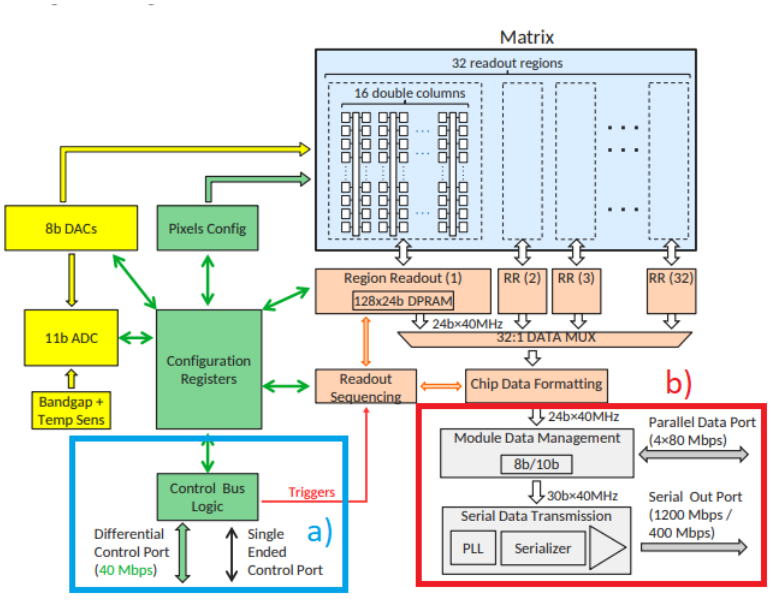



Figura 4.1: Diagramma della struttura a blocchi di un chip ALPIDE
*a*)Control Bus *(lettura lenta)*
*b*)Parallel Data & Serial out *(lettura veloce)*

Il chip ALPIDE utilizza due porte per il bus di controllo e l'acquisizione lenta: una porta differenziale $DCTRL$ e una $CTRL$ single-ended in uso a seconda dell'operazione da eseguire. L'interfaccia supporta quindi una comunicazione di tipo $bidirezionale$ e $half - duplex$: i dati possono essere scambiati in entrambe le direzioni, ma non simultaneamente. La differenza principale fra i segnali che passano

attraverso queste due porte è, in aggiunta alla velocità di trasmissione dati per ogni ciclo di clock gestito dal master, è il tipo di parola scambiata. L'interfaccia di controllo per l'acquisizione lenta ha due scopi principali:

1. provvedere l'accesso bidirezionale a registri interni, comandi, configurazione e memoria;
2. mandare un segnale di trigger o di broadcast di altri segnali sincroni.

La transazione sul Bus di Controllo è costituita da una sequenza di 10 bit. Un carattere (fig. 4.2) corrisponde allo scambio di un singolo byte ed è composto da un *leading bit* 0, il *byte* vero e proprio (8 bit) e un bit $trailing - stop$ 1. La convenzione per la trasmissione seriale è $LSB$ (Less Significant Bit) first.

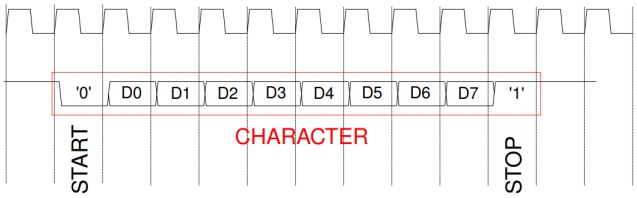



Figura 4.2: Esempio di singola parola scambiata attraverso la porta CTRL

Nel caso tuttavia sia stata abilitata la codifica $Manchester$, cosa che avviene di default, il chip ALPIDE risponderà sulla linea DCTRL in modo seriale e seguendo lo standard IEEE8 802.3 per indicare la fase/antifase (Fig. 4.3).

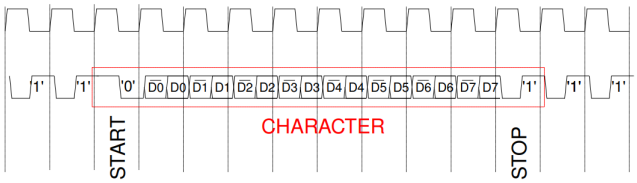



Figura 4.3: Esempio di parola scambiata sul bus DCTRL con Manchester encoding attivato (di default)

Nel caso tuttavia in cui la codifica Manchester sia stata disabilitata, i dati sono trasmessi attraverso la linea $CTRL$ ricevendoli dai due registri perifierici (HI $0x0012$, LO $0x0013$) in modo seriale e non più in parallelo. Di conseguenza la ricezione dei dati sarà più lenta.
In questo lavoro di tesi la codifica Manchester sarà disabilitata e la comunicazione avverrà con un protocollo simile a quello $I^2C$: due linee di trasmissione, una per il clock ed una bidirezionale per i dati. Tale scelta è stata fatta per una questione di semplicità e tempo, anche se la comunicazione veloce fra FPGA master ed ALPIDE sarebbe stata sicuramente migliore utilizzando la porta $DCTRL$.

### 4.1.1 L'operazione di lettura

Una valida transazione di controllo comincia con una stringa di caratteri predefiniti detti **OPCODEs**. Nel nostro caso invieremo solo il comando necessario per ordinare la lettura della memoria dei registri (*Start Unicast Read*): $RDOP$ $0x4E$. Per altri esempi di transazioni si veda la tabella 4.4.

L'operazione di lettura si basa su 4 bytes inviati dal bus master al chip ALPIDE, seguiti da 3 bytes di risposta. I primi 4 caratteri sono rispettivamente il $READ\ OPCODE$, il $CHIP\ ID$ per localizzare univocamente il chip, mentre gli ultimi due servono per specificare i registri interni $REG\_ADDR$ (ad esempio HI, LO visti in precedenza).
Dopo aver ricevuto le istruzioni, il chip inizia una fase detta di *bus turnaround* e a quel punto trasmette il proprio $CHIP\ ID$, che deve coincidere col precedente, seguito dai due byte di dati. I caratteri sono intervallati da delle fasi di *idle*, durante le quali lo slave continua a trasmettere '1' per un numero di cicli di clock guidato dal master.

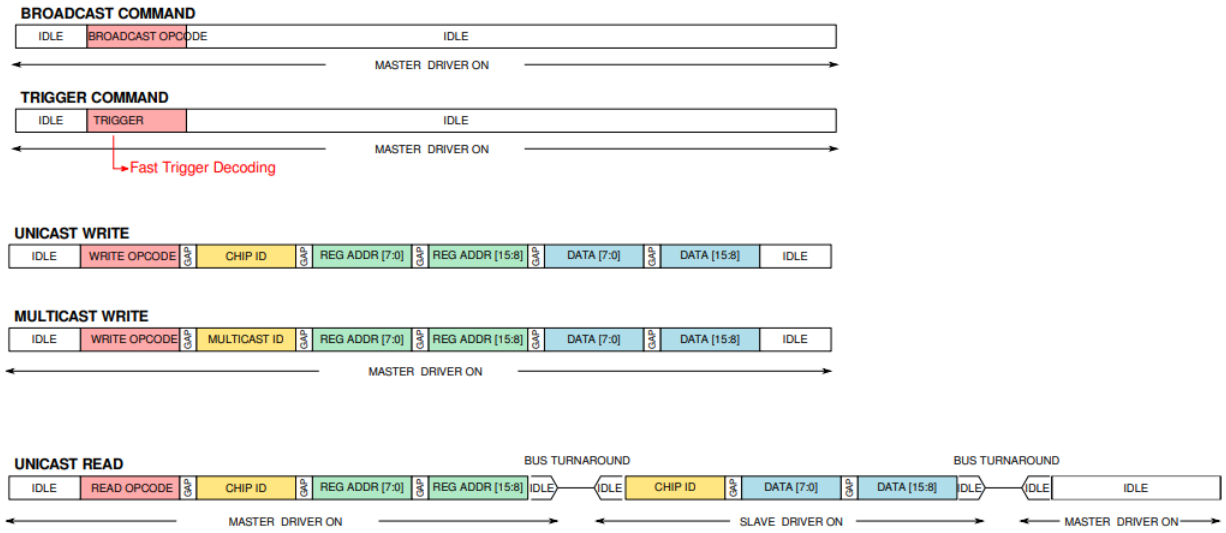



Figura 4.4: Esempio di transazioni valide sul Bus di controllo

### OPREAD - La risposta dello slave

In figura 4.5 sono rappresentati i segnali in ingresso ed in uscita dalla linea dei dati in funzione del numero di cicli di clock. Il totale del tempo di risposta lasciato allo slave è di 50 cicli di clock per l'invio di 3 bytes in totale: il $CHIP\ ID$ e i 2 bytes con le parole da decodificare ($DATAL$ e $DATAH$).

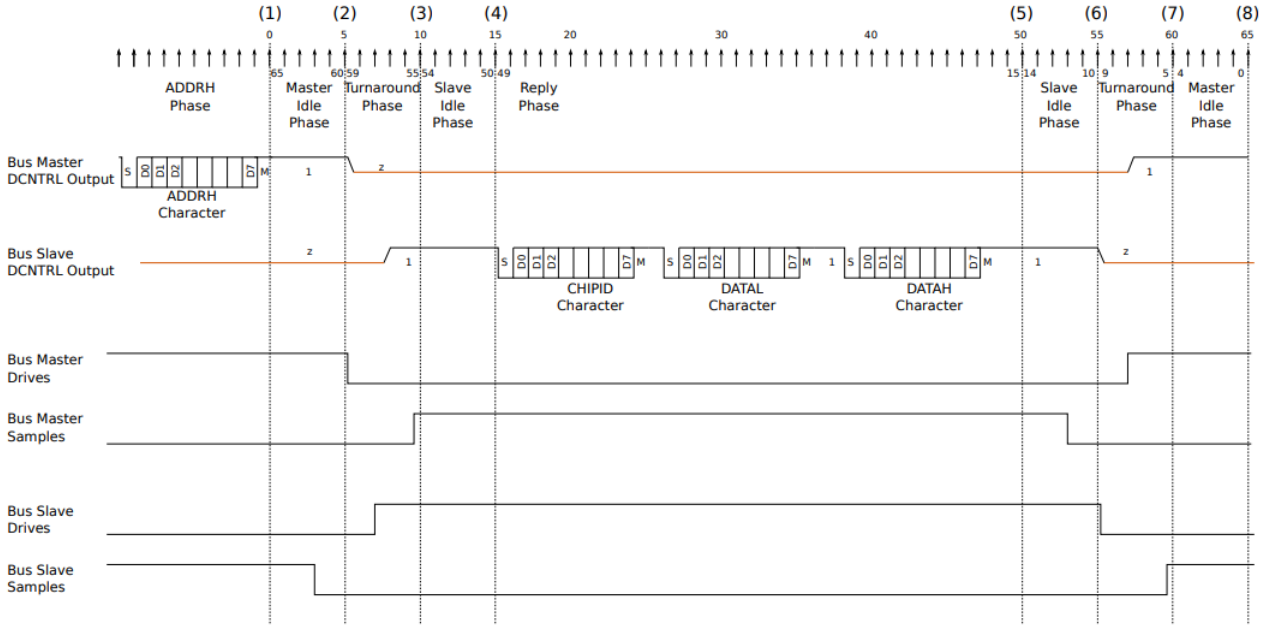



Figura 4.5: Diagramma di timing della risposta di una RDOP. Il segnale è visto in uscita alla linea DCNTRL, ma è pressoché uguale a quello visto utilizzando la porta single-ended.

### Decodifica parole

Una volta ricevute le parole da decodificare, queste possono di diverso tipo come si vede in tabella 4.6.

Le parole utili al nostro lavoro sono le seguenti:

**IDLE** parola utilizzata per segnalare che i dati non sono pronti per essere trasmessi.

| Data Word | Length (Bits) | Value (binary) |
|---|---|---|
| IDLE | 8 | 1111_1111 |
| CHIP HEADER | 16 | 1010<chip_id[3:0]><BUNCH_COUNTER_FOR_FRAME[10:3] > |
| CHIP TRAILER | 8 | 1011<readout_flags[3:0]> |
| CHIP EMPTY FRAME | 16 | 1110<chip_id[3:0]><BUNCH_COUNTER_FOR_FRAME[10:3] > |
| REGION HEADER | 8 | 110<region_id[4:0]> |
| DATA SHORT | 16 | 01<encoder_id[3:0]><addr[9:0]> |
| DATA LONG | 24 | 00<encoder_id[3:0]><addr[9:0]>_0_ <hit_map[6:0]> |
| BUSY ON | 8 | 1111_0001 |
| BUSY OFF | 8 | 1111_0000 |

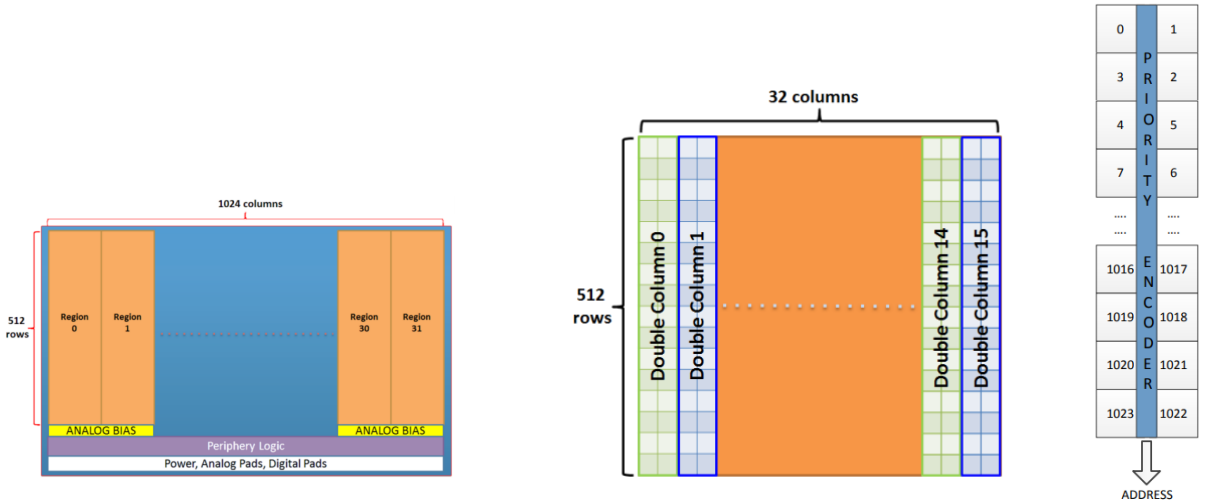
Figura 4.6: Tipi di parole che ALPIDE può restituire durante una OPREAD

**CHIP HEADER** sta ad indicare l'inizio di ogni pacchetto di dati: successivamente 4 bit identificano il chip, mentre altri 8 bit il tempo in cui il dato è stato acquisito.

**REGION HEADER** parola utilizzata per indicare l'inizio di una trasmissione di dati da una particolare regione (fig. 4.7a) individuata da un indice di 4 bit. I dati sono inviati sequenzialmente ed in ordine crescente: nel caso in cui una regione non registri nessun evento di hit, il suo header viene omesso e la trasmissione continua fino a quando una regione registra qualcosa.

**DATA SHORT** parola che contiene la locazione geografica utile ad individuare il singolo pixel colpito: attraverso l'indice dell'*encoder id* presente all'interno di ciascuna regione (fig. 4.7b) e del relativo indirizzo del pixel (fig. 4.7c), generato dall'encoder stesso.

**CHIP TRAILER** byte che si trova alla fine di ogni pacchetto di dati. E' seguito da 4 bit che rappresentano altrettante flags relative o all'evento, o allo stato del chip.




(a) *Disposizione geometrica delle regioni*

(b) *Sezione di una regione*

(c) *Indirizzi pixel generati dal priority encoder*

Figura 4.7: Scomposizione di un chip ALPIDE

I dati devono essere trasmessi un byte alla volta con la convenzione $MSB$ (Most Significant Byte) first, mentre le parole formate da 16 o 24 bit (si veda la tabella 4.6) non possono essere divise da un comando IDLE o BUSY.

## 4.2 Arduino master

Una volta noto il protocollo di scambio dati fra il chip ALPIDE e il master, sono state utilizzate le librerie già esistenti e sviluppate per il cosiddetto *Arduino shield*, implementato al CERN, come mezzanina per Arduino. Il PCB dove è bondato ALPIDE è stato disegnato in modo tale da avere le linee elettriche per il controllo e lo scambio dati su pin secondo uno schema compatibile con gli ingressi I/O di Arduino. Un esempio di dati acquisiti, in presenza di sorgente $\beta$, è presentato nel capitolo successivo (5.2). In seguito sono riportati alcuni dei segnali che sono stati utilizzati per implementare un readout tramite il microprocessore ARDUINO che sfrutta l'interfaccia di comunicazione lenta. Successivamente abbiamo importato questa comunicazione in FPGA.

**POWER** Porta di ingresso per l'alimentazione del chip ALPIDE.

**RST_N** Segnale globale *active-low* di reset la cui porta, in caso di applicazioni in cui non sia necessario, può esser lasciata sconnessa.

**CLK** Segnale di clock in ingresso ad ALPIDE che temporizza la trasmissione dei dati.

**CTRL** Porta di controllo bidirezionale (BUS) e single ended per la trasmissione dei dati, un bit alla volta a tempo di clock.

Il lavoro principale svolto in questo lavoro è quello di, tenendo conto del valore economico e la fragilità di un chip ALPIDE, simulare la comunicazione fra un Arduino UNO master ed un altro Arduino UNO slave, fingendo che questo sia lo stesso ALPIDE e tale per cui il protocollo di trasmissione dati sia il medesimo. In tutti i codici del paragrafo successivo il linguaggio di programmazione sarà il C++.

### 4.2.1 Arduino slave come simulatore ALPIDE

La difficoltà principale nell'implementazione del codice dello slave è fare in modo che quest'ultimo sia sensibile alle variazioni del segnale di clock: lo scambio di dati in input ed in output deve avvenire sul fronte di salita *rising_edge* del clock guidato dal master, mentre la lettura sul fronte negativo. Inoltre bisogna prestare particolare attenzione ad adattare le impedenze della porta $CTRL$ per rispettare l'alternanza di segnali INPUT/OUTPUT. Per il primo punto è stata implementato un metodo *read_clk*() della classe (scritta ad hoc) $Sim\_ALPIDE$, il quale attende uno o piu' (*int n*) cicli di clock dal master per procedere all'operazione successiva:

```
bool Sim_ALPIDE::read_clk(int n) {

  int cnt = 0;
  int lastclockstate = 0;

  while(true) {
  int clockstate = digitalRead(SIM_ALPIDE_PIN_CLK);
    if (clockstate != lastclockstate)  {
     if (clockstate == HIGH)  cnt++;
     else if (cnt == n) return true;
     }
    lastclockstate = clockstate;
    }
}
```

Si può ora implementare un metodo per la lettura di singole parole (1 byte) inviate dal master:

```
uint8_t Sim_ALPIDE::readbyte() {
  pinMode(SIM_ALPIDE_PIN_CTRL,INPUT);

  bool startfound = (digitalRead(SIM_ALPIDE_PIN_CTRL)==LOW );      // stop bit
  read_clk();
  uint8_t data = 0;

  for (int i=0;i<8;++i) {                                          // data bits, LSB first
    data >>= 1;
    if (digitalRead(SIM_ALPIDE_PIN_CTRL)==HIGH) data|=0x80;
    read_clk();
  }

 bool stopfound = (digitalRead(SIM_ALPIDE_PIN_CTRL)==HIGH);        // stop bit
  read_clk();

  if (!startfound) return -1;
  if (!stopfound ) return -2;
  return data;
}
```

Il codice scritto per guidare *int n* cicli di idle è invece il seguente:

```
void Sim_ALPIDE::idle(int n) {
  pinMode(SIM_ALPIDE_PIN_CTRL,OUTPUT);
  for (int i=0;i<n;++i) {
    read_clk();
    digitalWrite(SIM_ALPIDE_PIN_CTRL, HIGH);
  }
}
```

Infine, per poter comunicare dati al master, sempre attendendo il fronte di salita del segnale $CLK$, è stato implementato il metodo seguente:

```
void Sim_ALPIDE::sendbyte(uint8_t data) {

  digitalWrite(SIM_ALPIDE_PIN_CTRL,LOW );             // start bit
  pinMode(SIM_ALPIDE_PIN_CTRL,OUTPUT);
  read_clk();

  for (int i=0;i<8;++i) {                             // data bits, LSB first
    digitalWrite(SIM_ALPIDE_PIN_CTRL, data>>i&0x1);
    read_clk();
  }

  digitalWrite(SIM_ALPIDE_PIN_CTRL,HIGH);              // stop bit
  read_clk();
}
```

Un problema che è stato riscontrato nella trasmissione dei dati è la velocità con cui il master comunica il segnale di clock allo slave: se accade troppo velocemente alcuni cicli non vengono letti. Per

tale ragione è stato introdotto un delay di 5ms fra un fronte d'onda e l'altro. Lo stesso problema si presenterà anche nella scrittura del codice in VHDL per l'FPGA: avendo una frequenza di clock molto superiore a quella di Arduino UNO (16 MHz vs 450 MHz) il segnale di clock va quindi rallentato.

## 4.3 FPGA

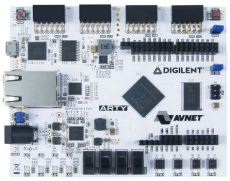


Figura 4.8: Arty Artix-7 FPGA Development Board

Una volta scritti i metodi fondamentali per mettere in comunicazione Arduino slave con il master, si passa all'oggetto vero e proprio della tesi: verrà utilizzata una scheda di sviluppo *Arty Artix-7 FPGA Development Board* (fig. 4.8) della Digilent che fungerà da master per comunicare con Arduino UNO in cui sono caricati i codici del capitolo 4.2.1. Arty ARTIX7 è stata scelta perché presenta una serie di pin compatibili con quelli di Arduino. In tal modo l'ALPIDE shield può essere inserito come mezzanina direttamente su Arty.

L'FPGA (*Field Programmable Gate Array*) indica, in elettronica digitale, un dispositivo in Silicio formato da un circuito integrato programmabile utilizzando un linguaggio di descrizione hardware: nel nostro caso VHDL (*Very High Speed Integrated Circuit-HDL*) in framework VIVADO. In generale gli FPGA contengono array di blocchi logici programmabili e collegamenti riconfigurabili che ne permettono la interconnessione. Tali blocchi logici possono svolgere diverse funzioni: dalle più semplici operazioni logiche OR o NAND fra segnali, a multiplexers, decodificatori a svariate linee di ingresso e di uscita utilizzando registri, memorie interne (SPI, DPRAM, flip-flops ecc).

## 4.4 FPGA MASTER ↔ Arduino SLAVE

Per l'obiettivo di questo lavoro di tesi, cioè permettere la comunicazione fra un microcontrollore Arduino UNO, facente veci di un chip ALPIDE, e la scheda FPGA Arty7 sono state implementate in VHDL un paio di macchine a stati finiti ($FSM$, *Finite State Machine*). Come brevemente descritto in precedenza, uno dei problemi principali è stata la frequenza di clock sostenibile dallo slave: ogni ciclo di clock in uscita dalla porta $CLK$ deve allora essere rallentato a circa 50'000 cicli di clock interno alla scheda, pena la comparsa di una capacità parassita con conseguente perdita di dati e lettura erronea di istruzioni da parte di Arduino.

### IPBus e DPRAM

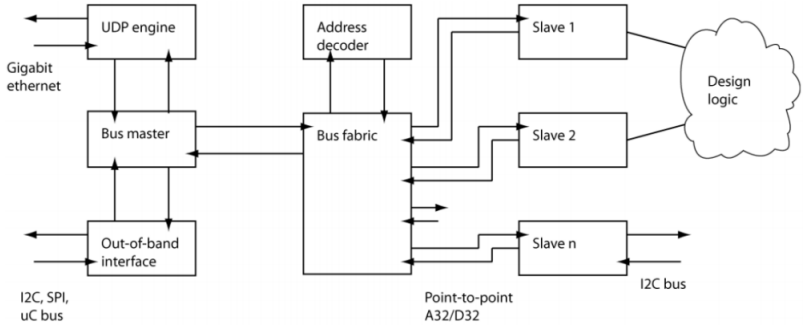



Figura 4.9: IPBus struttura del firmware

L'IPBus è un protocollo seriale basato su pacchetti di controllo, sviluppato al CERN nell'ambito dell'esperimento del CMS, e che serve per leggere e modificare dati contenuti internamente all'FPGA in registri mappati di dimensione 32bit e il cui indirizzo a sua volta è di 32 bit. Negli ultimi tempi questo protocollo sta prendendo piede per l'acquisizione dati, in particolare in fisica elettronica e

ricerca di particelle. La suite IPBus per questo progetto consiste nel firmware vero e proprio, che viene caricato attraverso un bitstream nell'FPGA, e dalla liberia uHAL che, grazie ad API di Python, permette la scrittura e la lettura di transazioni attraverso l'IPBus. Verrà utilizzata solo quest'ultima funzione. Ogni IPBus host device (in questo caso la scheda Arty) ha un indirizzo IP (10.10.10.100) e una porta numerata attraverso la quale accetta i pacchetti di controllo dell'IPBus. uHAL utilizzando un file .xml riceve in input l'indirizzo del registro al quale accedere per primo, permettendo così agli indirizzi mappati in memoria di avere una struttura gerarchica. Un altro file .xml è usato invece per configurare il collegamento Arty7 ↔ host (PC). Un terzo file contiene uno script Python per poter utilizzare i due precedenti.
Una DPRAM (*Dual Port Ram*) permette a due differenti Bus un accesso simultaneo, controllato da un processore, ed indipendente alla memoria. L'accesso è quindi garantito sia ad un master, sia all'host in cui si vogliono analizzare i dati.

### Invio di istruzioni MASTER→SLAVE

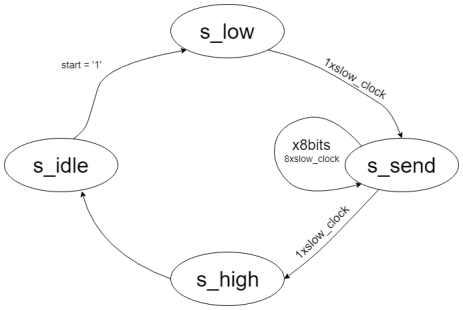



Figura 4.10: Schema di FSM per l'invio di informazioni al fake ALPIDE

Lo schema che rappresenta la FSM implementata per l'invio di istruzioni è visibile in figura 4.10. Qui, e successivamente, si parlerà di *clock* in sostituzione dello *slow_clock*, cioè al segnale in uscita dal master che regola i tempi per la trasmissione dei dati e non il clock interno della scheda, in altre parole sarà il clock rallentato. La macchina a stati si trova inizialmente nello stato *s_idle*. Una volta ricevuto il pulse di *start*, nel nostro caso inviato tramite un pulsante fisico della scheda FPGA, entra nello stato *s_low* in cui il segnale trasmesso allo slave è basso per tutta la durata di un clock. Nel caso in cui questo non sia '0', lo slave produrrà un errore. Al successivo fronte positivo del segnale di clock, il master entrerà nello stato *s_send* in cui invierà il byte con le istruzioni per lo slave per 8 cicli di clock, sempre partendo dal bit meno significativo. Una volta ultimata la parola, al colpo di clock successivo il master invierà un '1' essendo entrato nello stato *s_high*, per poi tornare nuovamente in *s_idle*. Come in precedenza, nel caso in cui il bit finale sia '0', lo slave produrrà una flag di errore. Il codice in VHDL utilizzato per la FSM lo si può trovare in appendice (App. .1).

### Lettura di dati MASTER←SLAVE e riscrittura nella DPRAM

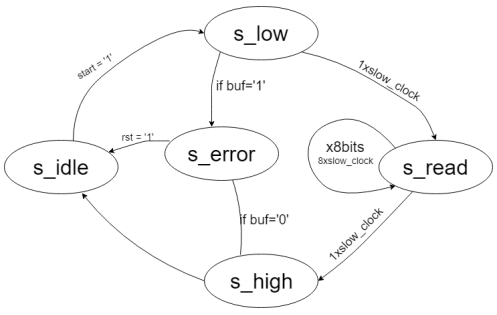



Figura 4.11: Schema di FSM per lettura di informazioni dal 'fake' ALPIDE

Una volta inviate le istruzioni ad ALPIDE contenenti ad esempio l'$OPCODE$ e il $CHIP\ ID$, questo risponderà con le informazioni richieste. In figura 4.11 è rappresentato un diagramma della FSM implementata per la lettura di informazioni dal 'fake' ALPIDE e conseguente riscrittura nella DPRAM. Inizialmente la macchina a stati si trova nello stato *s_idle*: una volta ricevuto il pulse di *start*, al solito attraverso pulsante fisico, entra nello stato *s_low* dove si attenderà '0'. Se così non fosse, entrerà nello stato *s_error* dove, per tornare in *s_idle* e poter leggere una nuova parola avrà bisogno di un segnale di reset. Altrimenti ad ogni ciclo di clock successivo, essendo entrato nello stato *s_read*, leggerà la parola un bit alla volta sempre partendo da quello meno significativo. Una volta trasmesso interamente il byte, sul fronte d'onda positivo del clock la FSM andrà nello stato *s_high* nel quale si aspetterà di ricevere '1'. Se così non fosse, entrerà nello stato *s_error* fino a quando non arriverà un segnale *rst* che la riporti in *s_idle*. Il codice in VHDL

utilizzato per la FSM lo si può trovare in appendice (App. .2).
Successivamente la parola letta viene scritta in un indirizzo di memoria della DPRAM, a cui si accede tramite la combinazione fra lo script di Python e i file .xml descritti in precedenza.

### Esempio

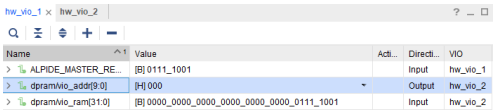



Figura 4.12: Utilizzo di VIO per debug di segnale

Per verificare che il codice in 4.2.1 e .2 sia funzionante, si procede a caricare in Arduino il codice contenuto nel metodo *sendbyte*. Mentre la parola che il master, sul quale è caricato il bitstream generato da .2, dovrà leggere ed immagazzinare nell'indirizzo di RAM $0x000000$ è, per questo particolare esempio, $0b01111001$. Per leggere la parola contenuta utilizzando l'IPBus si utilizzerà una macchina virtuale Linux Ubuntu 18.04.2 LTS configurata ad hoc. Come si può vedere una volta lanciato lo script di Python, la scrittura sulla DPRAM e la lettura attraverso protocollo IPBus sulla DPRAM è avvenuta correttamente. I codici implementati dunque funzionano.

```
> protocol : ipbusudp-2.0
> hostname : 10.10.10.100
> port : 50001
> path :
> extension :
> arguments :

@ Address [0x000000] is stored the value : 0x79
@ Address [0x000001] is stored the value : 0x00
@ Address [0x000002] is stored the value : 0x00
@ Address [0x000003] is stored the value : 0x00
@ Address [0x000004] is stored the value : 0x00
@ Address [0x000005] is stored the value : 0x00
```

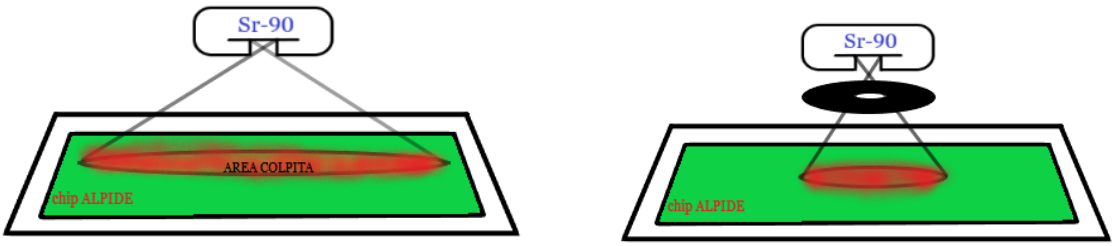
Figura 4.13: Lettura tramite protocollo IPBus

# 5. Readout del chip ALPIDE con Arduino

Abbiamo svolto un'acquisizione dati di test in presenza di diversi setup sperimentali, tutti con lettura tramite Arduino ed utilizzando una sorgente di raggi $\beta$.

## 5.1 Setup sperimentale

In presenza dello Stronzio-90 (sorgente di tipo $\beta$) viene acceso il chip ALPIDE per acquisire eventi di hit utilizzando Arduino UNO come master. Risultando molto complesso triggerare sul singolo evento senza assorbire l'elettrone a bassa energia, abbiamo sfruttato uno dei metodi presenti nella libreria *Arduinoshield*: è stato aperto un gate di 1000ms in cui si integrano tutti gli hit da elettoni emessi dalla sorgente. La geometria dell'apparato sperimentale è schematizzata in figura 5.3.
Nel primo caso (fig. 5.1a) le particelle $\beta$ sono emesse attraverso un foro di diametro 8(1)mm orientato verso il basso. Il disco contenente la sorgente ha a sua volta un'altezza di 8(1)mm. Il chip ALPIDE invece si trova ad una distanza $d \approx 15(2)$mm, misurato a partire dal punto più alto del disco.
Nel secondo caso, immediatamente a contatto con la sorgente, è stato aggiunto un collimatore con un foro centrale il cui diametro è stato stimato in 1.2(1)mm. La funzione di tale dischetto, di spessore 1.0(1)mm è quella di restringere il raggio del cono in cui sono emesse le particelle e, di conseguenza, l'area in cui ALPIDE le rileva.



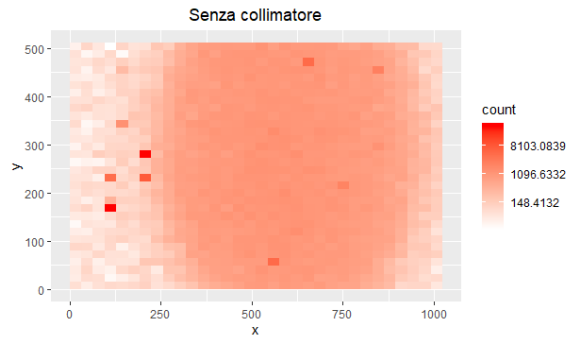

(a) *In assenza di collimatore* (b) *Con collimatore*

Figura 5.1: Schema della geometria dell'esperimento (figure non in scala)

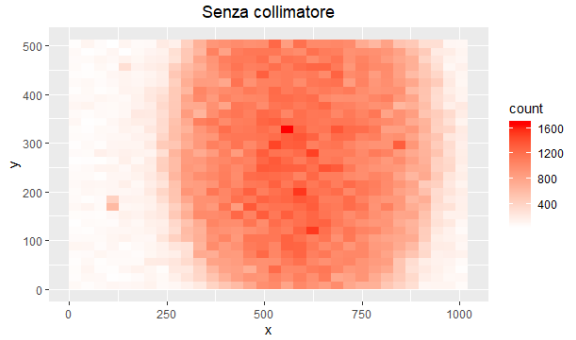
## 5.2 Analisi dati

Inizialmente vengono mascherati i pixel difettosi/rumorosi attraverso un metodo presente nelle librerie *Arduino shield*: sono facilmente riconoscibili in quanto ad ogni finestra di acquisizione generano un segnale 'fake'. Il conteggio degli eventi di hit è rappresentato nelle seguenti immagini: quella più a sx (figg. 5.2a, 5.2c) in scala logaritmica per evidenziare i pixel rumorosi e le aree colpite, mentre in quella

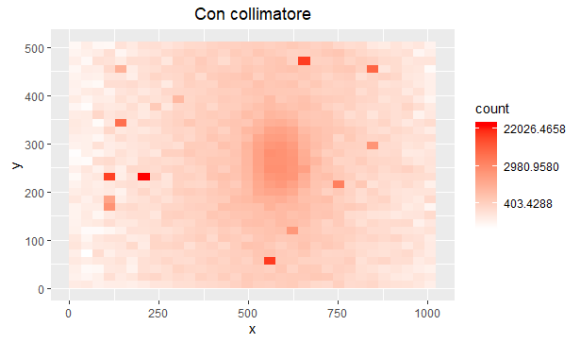
a dx (figg. 5.2b, 5.2d) il conteggio degli eventi è in scala decimale dopo aver 'silenziato' i pixel con più di 1000 conteggi assumendo si tratti di pixel rumorosi.



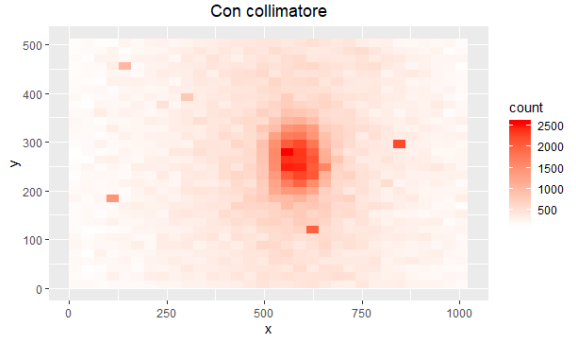

(a) *Senza collimatore (log scale)*




(b) *Senza collimatore (lin scale) senza pixel rumorosi*



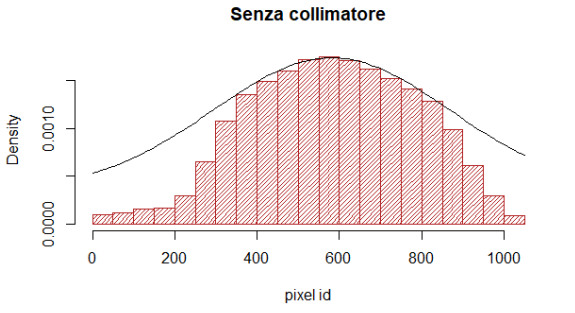

(c) *Con collimatore (log scale)*



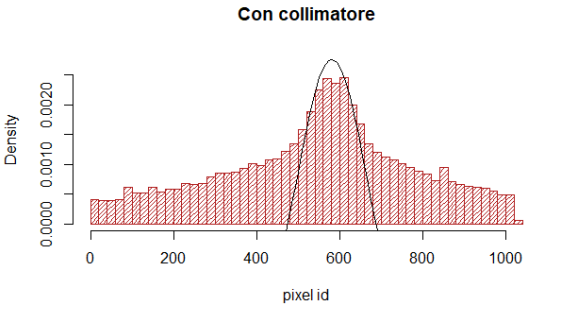

(d) *Con collimatore (lin scale) senza pixel rumorosi*

Figura 5.2

Come si può notare il conteggio eventi, nel secondo caso in presenza di collimatore, sono confinati in una regione di area minore. Nota la geometria dell'apparato e l'isotropia dell'emissione di particelle $\beta$ da parte della sorgente, si può allora semplificare il problema portandoci ad un'analisi in una sola dimensione: nel nostro caso sono state scelte le $x$ in quanto, in entrambe le prese dati, il diametro del disco che indica l'area colpita del chip è interamente contenuto nel range $pix\_id = \{1 : 1024\}$. Il profilo che ci attenderemo in prossimità del centro dell'area colpita è di tipo gaussiano.




(a) *La gaussiana ha* ($\mu = 581(1),\ \sigma = 287(3)$)




(b) *La gaussiana ha* ($\mu = 581(1),\ \sigma = 82(1)$)

Figura 5.3: Profili gaussiani nel conteggio degli eventi sulla cordinata x

Il fit gaussiano è ottenuto solo tenendo conto dei bin centrali, cioè quelli che hanno acquisito più eventi: si assume che le code presenti nei plot siano dovute ad effetti di scattering col materiale del collimatore. Una volta stimati i parametri delle gaussiane interpolanti, si procede al calcolo delle FWHM che fisicamente considereremo come il diametro delle aree colpite direttamente dall'emissione

di particelle. Ci attenderemo, dalla geometria del sistema (fig. 5.3), che il rapporto fra le FWHM, e di conseguenza fra le $\sigma$ stesse, relative alle due diverse prese dati sia circa $3.1 \pm 1$.

$$R = \frac{\sigma_1}{\sigma_2} = 3.5 \pm 1$$

Essendo la compatibilità discreta, le assunzioni fatte sono approssimativamente corrette. Per gate di acquisizione misuriamo una decina di hit di segnale che torna approssimativamente con la rate di emissione della sorgente.

## 5.3 Scattering

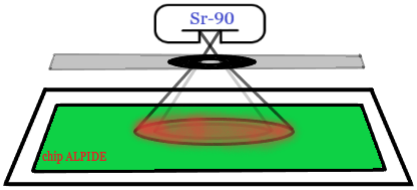
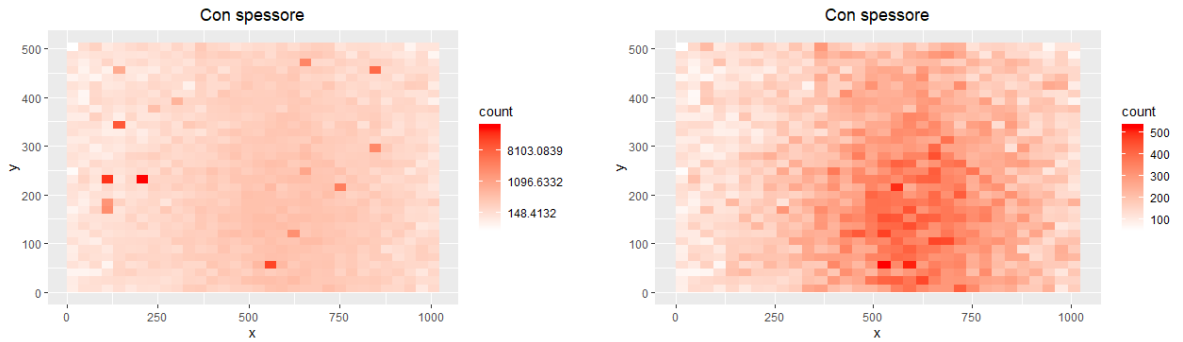



Figura 5.4: Setup in presenza di spessore (figura non in scala)

E' stato scelto inoltre di acquisire dati in presenza di uno spessore di 9(1)mm subito dopo il collimatore per verificare sperimentalmente l'effetto dello scattering. Il setup è molto simile ai precedenti, come si può notare dall'immagine a lato (fig. 5.4). Viene svolta l'analisi dati con lo stesso metodo: inizialmente si disegna un plot in scala logaritmica del conteggio degli eventi (fig. 5.5a) per evidenziare eventuali pixel rumorosi, in secondo luogo si sceglie una soglia sopra la quale considerare un pixel rumoroso. In questo caso tutti i pixel con un conteggio di eventi superiori a 100 sono stati silenziati, in quanto si vede bene nel plot a dx (fig. 5.5b) come gli hit si distribuiscano in modo più uniforme e in una superficie maggiore.




(a) *Con spessore e collimatore (log scale)*

(b) *Con spessore e collimatore (lin scale) e senza pixel rumorosi*

Figura 5.5

Seguendo il filo logico precedente consideriamo solo il conteggio di eventi lungo la coordinata $x$. Ci aspettiamo che il profilo sia gaussiano nei bin centrali con una varianza decisamente più elevata dovuta allo scattering (fig. 5.6).

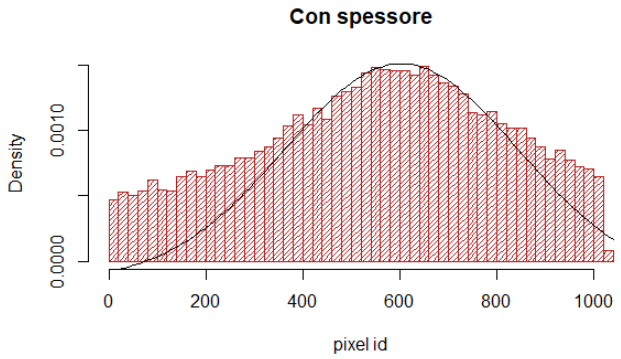



Figura 5.6: La gaussiana ha ($\mu = 603(2),\ \sigma = 240(4)$)

Verifichiamo così che le assunzioni fatte sono corrette: la varianza è quasi triplicata rispetto al setup con solo collimatore, raggiungendo quasi la stessa stima della prima presa dati. Ciò è dovuto principalmente allo scattering fra le particelle $\beta$ emesse dalla sorgente e il materiale dello spessore (polietilene): il raggio del cono di emissione della sorgente aumenta, così come diminuisce il numero del conteggio degli eventi in quanto alcune particelle si fermano durante l'attraversamento dello spessore.

# 6. Conclusioni

Inizialmente in questo lavoro di tesi si è studiato cosa sono i rivelatori pixel a Si ed alcuni loro possibili utilizzi, ad esempio la costruzione di un telescopio per la caratterizzazione di un sensore innovativo.
E' stato necessario tuttavia uno studio preliminare per acquisire i fondamenti di elettronica digitale: funzionamento ed importanza del segnale di clock, utilizzo di registri, possibili impieghi di microprocessori, macchine a stati finiti e comunicazione mediante protocolli standard (master/slave). Senza queste conoscenze, non sarebbe stato possibile impostare il readout di ALPIDE via microcontrollore prima, utilizzando il linguaggio di programmazione C++, ed FPGA poi, sfruttando la potenza e la versatilità del VHDL appreso per lo scopo.
Per concludere l'esperienza sono stati acquisiti dati con ALPIDE sfruttando l'Arduino shield in presenza di sorgente di tipo $\beta$, in modo tale da verificare l'effetto dello scattering utilizzando diversi setup sperimentali.
In prospettiva restano ancora in sospeso il completamento e i test del readout lento di ALPIDE attraverso il Bus di Controllo utilizzando una scheda FPGA, il readout veloce sempre con FPGA, ed infine il setup del telescopio per la caratterizzazione di APiX utilizzando 4 stazioni ALPIDE.

# Appendice

## .1 FSM1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity master is
 Port (
    power:    in std_logic;
    rst:      in std_logic;
    clk_in:   in std_logic;
--    data_in:  in std_logic_vector(7 downto 0); --word to be sent
    strt:     in std_logic;

    ready:   out std_logic;
    clk_out: out std_logic;
    data_out:out std_logic;
    started: out std_logic
  );
end master;

architecture Behavioral of master is

COMPONENT vio_0
  PORT (
    clk : IN STD_LOGIC;
    probe_out0 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;

signal start, ready_signal: std_logic;
signal vio_word, word:  std_logic_vector(7 downto 0);

signal counter: unsigned (27 downto 0);
signal slow_clk: std_logic;

type state is (s_idle, s_low, s_send, s_high);
signal state_fsm: state;
```

```
begin

slow_counter: process (clk_in) is
begin
  if rising_edge(clk_in) then
    counter <= counter+1;
  end if;
end process;

p_sendword: process (clk_in, start)
variable word_cnt: integer := 0;
variable SL_CLOCK: integer := 0;
variable WTIME:    integer := 50000;
begin
if rising_edge(clk_in) then
   case state_fsm is

   when s_idle =>
    if start = '1' then       --if start button pressed
       state_fsm <= s_low;    --change state
       slow_clk <= '0';
    else state_fsm <= s_idle; --otherwise remains in idle
         data_out  <= '0';
         slow_clk  <= '0';
         ready_signal <= '0';
    end if;

   when s_low  =>
        data_out <= '0';
        if SL_CLOCK = WTIME/2 then
           slow_clk <= '1';
        end if;
        if SL_CLOCK = WTIME then
            state_fsm <= s_send;
            slow_clk <= '0';
            SL_CLOCK := 0;
        end if;
        SL_CLOCK := SL_CLOCK + 1;

   when s_send =>
        data_out <= vio_word(word_cnt);
        if SL_CLOCK = WTIME/2 then
           slow_clk <= '1'; --here the slave updates
           end if;
        if SL_CLOCK = WTIME then
            if word_cnt = 7 then --whole word sent!
               state_fsm <= s_high;
               slow_clk <= '0';
               word_cnt := 0;    --reset counter word
               SL_CLOCK := 0;
            else word_cnt := word_cnt + 1;
               slow_clk <= '0';
```

```
                SL_CLOCK := 0;
            end if;
        end if;
        SL_CLOCK := SL_CLOCK + 1;

   when s_high =>
        data_out <= '1';
        if SL_CLOCK = WTIME/2 then
            slow_clk <= '1';
        end if;
        if SL_CLOCK = WTIME then
            state_fsm <= s_idle;
            ready_signal <= '1';
            slow_clk <= '0';
            SL_CLOCK := 0;
        end if;
        SL_CLOCK := SL_CLOCK + 1;

   when others =>
        state_fsm <= s_idle;
  end case;
 end if;
end process;

vio: vio_0 port map (
    clk => clk_in,
    probe_out0 => vio_word
);

clk_out  <= slow_clk;
start    <= strt; --input for fsm, strt is the button pressed
started  <= start;--input for receiver
word     <= vio_word;
ready    <= ready_signal;

end Behavioral;
```

## .2 FSM2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity master is port (
    clk_in:   in std_logic;
    strt:     in std_logic;
    data_in:  in std_logic;
    rst:      in std_logic;

    word:    out std_logic_vector(7 downto 0);
    ready:   out std_logic;
```

```
    clk_out: out std_logic;
    started: out std_logic
    );
end master;

architecture Behavioral of master is

COMPONENT vio_word
  PORT (
    clk : IN STD_LOGIC;
    probe_in0 : IN STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;

signal start, slow_clk, data: std_logic;
signal ready_signal, rst_signal: std_logic;

signal word_temp: std_logic_vector(7 downto 0);

type state is (s_idle, s_low, s_read, s_high, s_error);
signal state_fsm: state;

begin

p_sendword: process (clk_in, start)
variable word_cnt: integer := 0;
variable SL_CLOCK: integer := 0;
variable WTIME:    integer := 50000;
begin
if rising_edge(clk_in) then
   case state_fsm is

    when s_idle =>
       if start = '1' then    --if start button pressed
       state_fsm <= s_low;    --change state
       slow_clk <= '0';
       SL_CLOCK := 0;
       ready_signal <= '0';
    else state_fsm <= s_idle; --otherwise remains in idle
         slow_clk <= '0';
         SL_CLOCK := 0;
         ready_signal <= '0';
    end if;

    when s_low  =>
        if SL_CLOCK = WTIME/2 then
           slow_clk <= '1';
        end if;
        if SL_CLOCK = WTIME then
            if  data = '0' then
                state_fsm <= s_read;
                slow_clk <= '0';
                SL_CLOCK := 0;
```

```
          else if data = '1' then
               state_fsm <= s_error;
               end if;
          end if;
       end if;
       SL_CLOCK := SL_CLOCK + 1;

  when s_read =>
       if SL_CLOCK = WTIME/2 then
          slow_clk <= '1';
       end if;
       if SL_CLOCK = WTIME then
          word_temp(word_cnt) <= data;
          if word_cnt = 7 then
             state_fsm <= s_high;
             slow_clk <= '0';
             word_cnt := 0;    --reset counter word
             SL_CLOCK := 0;
          else
             word_cnt := word_cnt + 1;    --reset counter word
             SL_CLOCK := 0;
             slow_clk <= '0';
          end if;
       end if;
       SL_CLOCK := SL_CLOCK + 1;

  when s_high  =>
       if SL_CLOCK = WTIME/2 then
          slow_clk <= '1';
       end if;
       if  SL_CLOCK = WTIME then
           if  data = '1' then
               state_fsm <= s_idle;
               ready_signal <= '1';
               slow_clk <= '0';
               SL_CLOCK := 0;
           else if data = '0' then
               state_fsm <= s_error;
               end if;
           end if;
       end if;
       SL_CLOCK := SL_CLOCK + 1;

  when s_error =>
       if rst_signal = '1' then
           state_fsm <= s_idle;
       end if;

  when others =>
       state_fsm <= s_idle;

 end case;
end if;
```

```
end process;

vio_out : vio_word port map (
    clk => clk_in,
    probe_in0 => word_temp
);

clk_out   <= slow_clk;
start     <= strt;            --input for fsm, strt is the button pressed
started   <= start;        --input for receiver
data      <= data_in;
ready     <= ready_signal;
word      <= word_temp;
rst_signal<= rst;

end Behavioral;
```

# Bibliografia


[1] M. Suljic, *Study of Monolithic Active Pixel Sensors for the Upgrade of the ALICE Inner Tracking System*, Università degli studi di Trieste, a.a. 2016-2017

[2] S. Kwan, CM Lei, D. Menasce, L. Moroni, J. Ngadiuba, A. Prosser, R. Rivera, S. Terzo, M. Turqueti, L. Uplegger, L. Vigani, *The CMS Pixel Tracking Telescope at the Fermilab Test Beam Facility*, Fermi National Accelerator Laboratory, Batavia, IL, USA; INFN sezione Milano Bicocca & Università Studi di Milano Bicocca

[3] ALICE ITS ALPIDE development team, *ALPIDE Operations Manual, DRAFT Version: 0.3*, CERN, Genève; July 25, 2016

[4] Gianmaria Collazuol, *Slides del corso Management and Analysis of Physical Data – part I. FPGA and VHDL* , Università degli Studi di Padova, a.a. 2018-19.

[5] Stefano Pavinato, *Slides del corso Management and Analysis of Physics Datasets, Part. 1 e codici, Lezioni 1-10*, Università degli Studi di Padova, a.a. 2018-19.

[6] L. Pancheri, A. Ficorella, P. Brogi, G. Collazuol, G. Dalla Betta, P.S. Marrocchesi, F. Morsani, L. Ratti, A. Savoy-Navarro, A. Sulaj, *First Demonstration of a Two-Tier Pixelated Avalanche Sensor for Charged Particle Detection*, INFN, Italy, CNRS, Paris, 23 August, 2017.