# Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)
STMicroelectronics NV




Tesi di laurea magistrale

# System level test techniques for automotive systems-on-chip


*Relatore:*
Paolo Bernardi

*Correlatore:*
Matteo Sonza Reorda

*Candidato:*
Giovambattista Gallo

*Ad Anna*

# Contents

# Abstract

La crescente complessità di integrazione nei System on Chips, e soprattutto il loro utilizzo in nuove applicazioni ha stravolto il mondo del testing industriale e la complessità degli equipaggiamenti per il test. Il lavoro di tesi ha avuto come oggetto lo sviluppo di una centralina multicore in grado di realizzare il test del Burn-in su di un SoC automotive. Questo test mira a stimolare il device aumentando la switching activity dei vari gate che lo compongono, così da ottenere una stima della sua affidabilità e del suo comportamento in situazioni limite. La procedura implementata prevede che il device entri in test-mode attraverso la corretta configurazione del boundary-scan. Tale configurazione in questa tipologia di test prevede un'unica scan-chain formata da tutti i flip-flop del SoC. Il pattern di test inserito nella scan-chain è generato in maniera pseudo-random con un algoritmo LFSR ed è parallelamente compresso in output attraverso un algoritmo SISR. In questo contesto di analisi dello stress del circuito gli sforzi si sono poi spostati dall'ambito hardware ad un ambito euristico. La valutazione dello stress realizzato mantenendo fissi i valori sui nodi dei gate del circuito in simulazione, richiede un elevato tempo di computazione per ricavare le coppie in grado di generare affinità elettromagnetica. Per sopperire a tale problematica è stato realizzato un algoritmo di machine learning dalle prestazioni più efficienti in termini di velocità d'esecuzione.

# Ringraziamenti

Riconoscere a chi dare meriti dei propri successi oltre che a se stessi è una operazione estremamente complicata, ma nel lungo percoso della mia vita ho avuto la costante fortuna di essere accompagnato da una famiglia presente ma non invasiva, e quindi devo senza dubbio ringraziare loro, per il supporto e il senso di responsabilità conferitomi.

Un altro dei cardini fondamentali di un traguardo è la cerchia di amicizie: io ho avuto la fortuna di avere tanti amici, chi da sempre, chi per poco, chi da poco,chi per sbaglio, chi per mia fortuna, chi non ho più e chi ci sarà sempre, ognuno di loro ha avuto un ruolo fondamentale nel rendermi quello che sono oggi.

L'axios del titolo accademico è ovviamente di chi mi ha istruito e di quelli con cui ho condiviso l'esperienza di apprendimento: a loro devo la mia professionalità oltre che l'affilamento delle mie doti umane.

# Chapter 1

# Introduzione

La crescente complessità di integrazione nei System on Chips, e sopratutto il loro utilizzo in nuove applicazioni ha stravolto il mondo del testing industriale e la complessità degli equipaggiamenti per il test. *Test* è il processo orientato all'identificazione dei prodotti guasti, ovvero quei prodotti che registrano dei fallimenti di funzionamento rispetto a quanto viene garantito dalle specifiche del produttore. Esistono varie classi di test:

- Verification Testing
- Manufacturing Testing
- Burn-in
- System Level Test
- Incoming Inspection
- On-line Testing

ed in questo lavoro di tesi l'oggetto di interesse è stato proporre una architettura semplice, economica e versatile per il **Burn-in** sul SoC di proprietà STMicroelectronics soprannominato *Bernina*. Il Burn-in ha l'obiettivo di individuare i componenti del SoC soggetti alla mortalità infantile, in modo da riuscire ad eliminare dalla popolazione i chip che falliscono un numero ingente di volte nella fase iniziale, garantendo una maggiore affidabilità del prodotto in esame.

Il BI applica 2 tipologie di stress : *stress esterno*, basata su temperature e voltaggi superiori a quelli di utilizzo, e *stress interno*, quella realizzata, che attiva le funzionalità del device durante la fase di test. Il BI è stato realizzato utilizzando un LFSR software in input per generare i pattern per il testing e un SISR software in output per comprimere i pattern generati in una firma, così da permettere una diagnostica sui vari componenti del SoC, prima in una versione single-core, sia utilizzando una board NUCLEO sia una board EIGER, ed in seguito parallelizzando i processi di LSFR e SISR sfruttando la capacità multi-core della board EIGER.

Il caso di studio ha presentato la necessità di approfondire un ulteriore processo di testing, di forte interesse in ambito industriale, basato sulla valutazione dello stress statico del SoC: l'idea è quella di mantenere in simulazione per un certo intervallo di tempo dei valori fissi su ogni nodo del circuito e valutare la copertura tra nodi vicini, dove è più probabile nascano fenomeni di affinità elettromagnetica. L'ostacolo principale di queste computazioni nasce dalla ingente mole di dati, Bernina è composta da circa 20 milioni di nodi, e l'approccio proposto è stato quello di implementare un algoritmo di machine learning in grado di ottimizzare il tempo di elaborazione del vicinato, con la ulteriore utile capacità di considerare le diverse densità delle varie aree del SoC.

# Chapter 2

# Burn-in

## 2.1 LFSR

Il **L**inear **F**eedback **S**hift **R**egister ( Registro a scorrimento a retrazione lineare) produce dei dati sfruttando una funzione lineare ( lo XOR *ndr*) dello stato interno al registro, ed è stato applicato per generare dei pattern random per stimolare il DUT: ovvero LFSR che sfruttiamo produce un numero di output pari al numero di Flip-Flops da cui è composta la catena di scan di Bernina ( 661477 bits ).

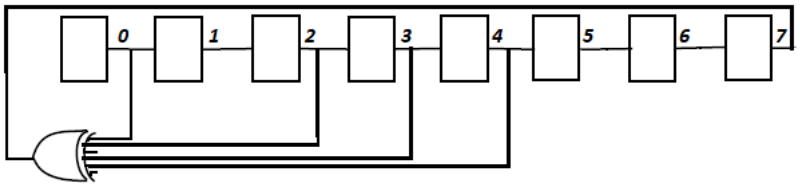



Figure 2.1: Scan-chain di un lfsr a 8 bit

Ad ogni shift corrisponde un cambiamento di stato e quindi un nuovo bit per la sequenza del pattern, e questo stato successivo è influenzato da una lista di posizioni di bit, detta *sequenza di tap*, che è rappresentabile sotto forma di un polinomio avente come coefficienti 1 o 0, noto come *poliniomio di retroazione*; la nostra sequenza di tap è data dal polinomio:

$$x^1 + x^2 + x^{22} + x^{32} = 0$$

## 2.2 SISR

In una tecnica di **B**uilt-**I**n **S**elf-**T**est come quello che stiamo adottando sarebbe impossibile memorizzare tutti gli output a causa dall'eccessivo consumo di memoria che richiederebbe, ed è quindi possibile comprimere in una **firma** il pattern che generiamo in input, che potrà essere in seguito confrontata con una *firma golden* per rilevare i fallimenti.

La compressione è realizzata con il **SISR**, Single Input Shift Register, che è una tipologia particolare di LFSR: condividono, infatti, la struttura a catena e l'utilizzo della sequenza di tap, con la differenza che nel SISR si aggiungono un EXOR e un bit di input.

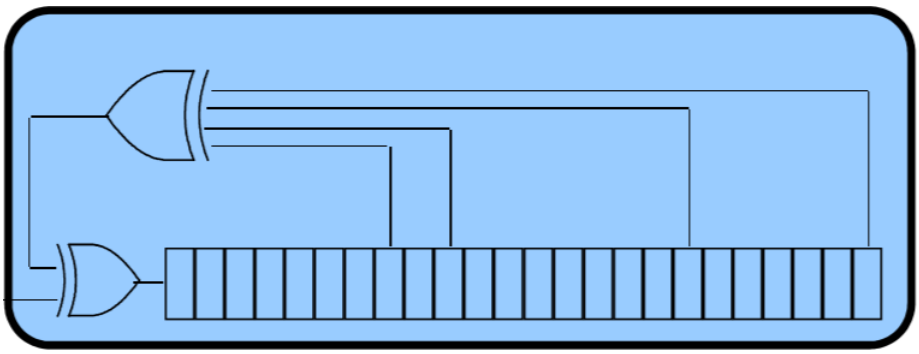


Figure 2.2: Signature analyzer

Poichè l'obiettivo è avere una firma univoca per ogni pattern generato, è essenziale ridurre al minimo la probabilità di generare due firme uguali partendo da due pattern diversi.

L'idea alla base del SISR è trasformare i bit del pattern in un polinomio, dividerlo per un altro polinomio e considerare come firma il resto della divisione, e considerando tutte sequenze di lunghezza $m$ bisogna minimizzare:

$$P = \frac{2^{m-n}-1}{2^{m-1}-1}$$

In pratica vengono utilizzati i polinomi irriducibili per garantire la minor probabilità di collisione possibile.

## 2.3 Architettura Tester

Le fondamenta del progetto risiedono in una iniziale architettura composta da un microcontroller denonimato NUCLEO di proprietà STMicroelectronics NV.

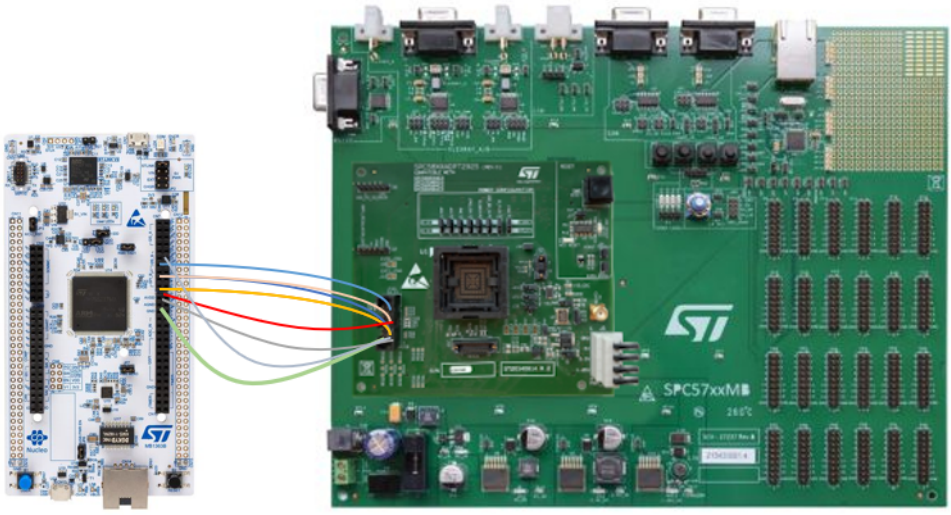



Figure 2.3: Connessione tra tester e DUT

Con questa architettura low-cost sono iniziati gli esperimenti di compressione del pattern, sfruttando la strategia double-buffer, sia nell'invio che nella compressione.
Il buffer in ricezione è diviso in due aree: una utilizzata per depositare i valori letti dal DUT, e l'altra in cui accede la CPU per applicare il SISR.

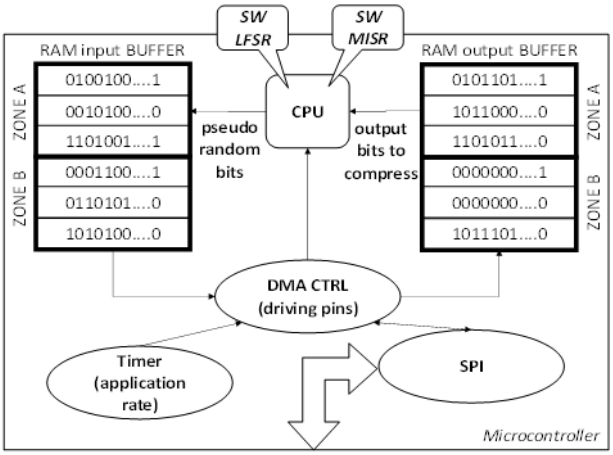



Figure 2.4: Architettura Nucleo single-core

A questo punto dello studio si è inserita la necessità di aumentare la frequenza a cui riusciva a lavorare il tester per compiere le due operazioni ed è nata l'idea di fare il Porting dell'architettura su un altro SoC automotive, denominato **Eiger**, sempre di proprietà STMicroelectronics:
è un device sofisticato relativamente low-cost con cui si sono sviluppate due tipologie di architettura:

- Architettura single-core
- Architettura multi-core

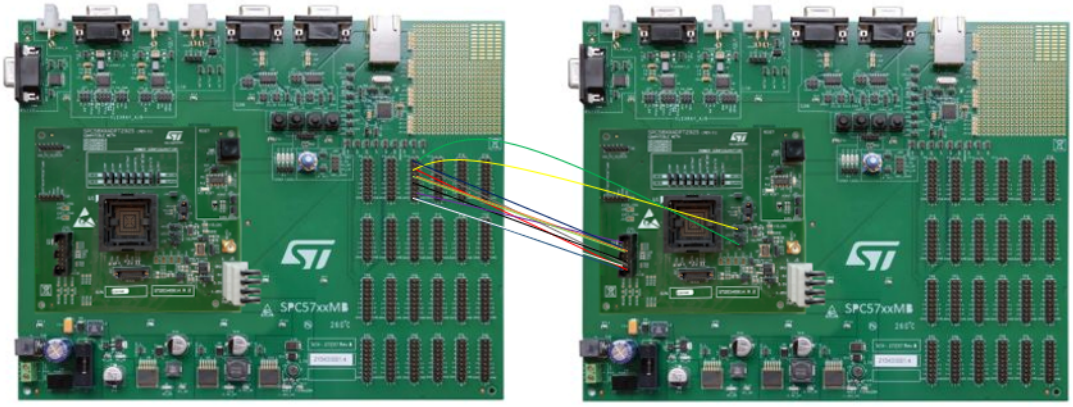



Figure 2.5: Connessione tra tester e DUT

### 2.3.1 Architettura single-core

Il System-On-Chip automotive soprannominato **Eiger** di proprietà STMicroelectronics NV è stato utilizzato come tester per il BI: il Core-0 genera la sequenza di ingresso in test mode e la inoltra al DUT attraverso la **S**ystem **I**ntegration **U**nit **L**ite **2**, a questo punto parte la generazione del pattern pseudo-random derivato dalla strategia LFSR, di cui viene fatto upload e download in scan-chain attraverso un **S**erial **P**eripheral **I**nterface.
Una volta scaricato tutto il pattern in RAM di Eiger viene compresso con il SISR e la signature è salvata in una predisposta struct allocata staticamente in una determinata zona di memoria.

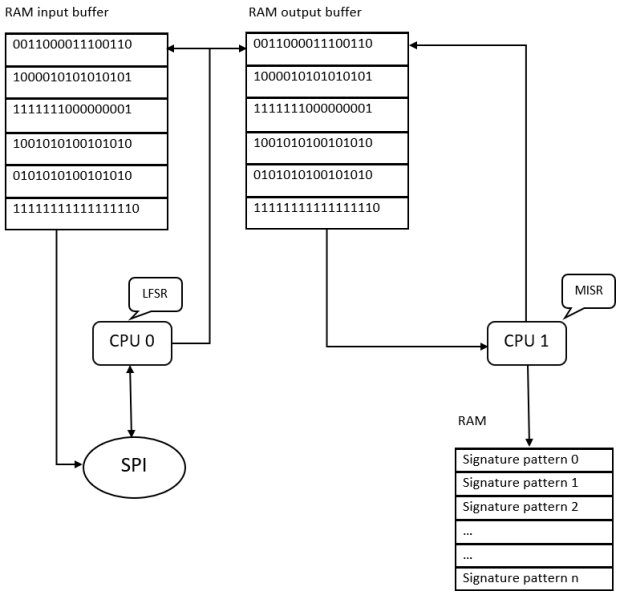



Figure 2.6: Architettura Eiger single-core

### 2.3.2 Ingresso in Test Mode

Per realizzare lo shift completo della catena di scan ed effetture il test di Burn-In è necessario inviare una specifica sequenza di segnali alla JTAG del DUT. I segnali coinvolto in questo processo sono i seguenti nove:

- TCK
- EXTAL
- JCOMP
- TMS
- TDI
- TESTMODE
- PORST
- ESR0
- TDO

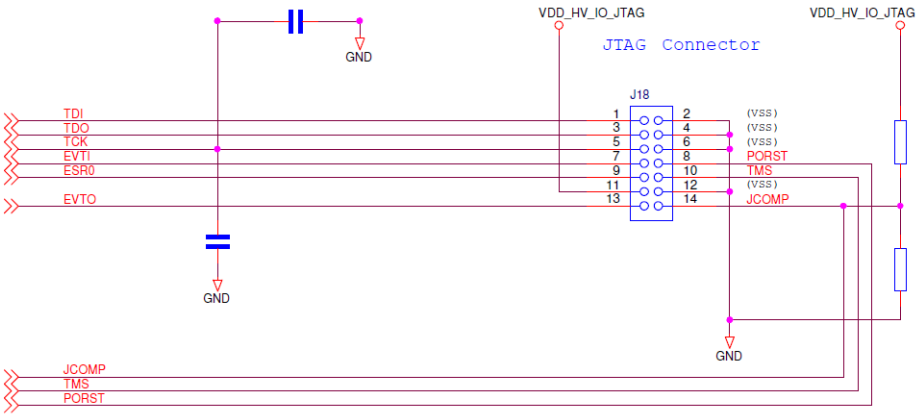



Figure 2.7: interfaccia JTAG

Tale sequenza è generata emulando il meccanismo del *TAP controller*, indispensabile per controllare l'*Istruction Register*, i quali valori determinano il comportamento del Boundary Scan in accordo con la modalità opeartiva definita, e il *Data Register*, registro dove vengono caricati serialmente i test pattern e dove possiamo osservarne i risultati:

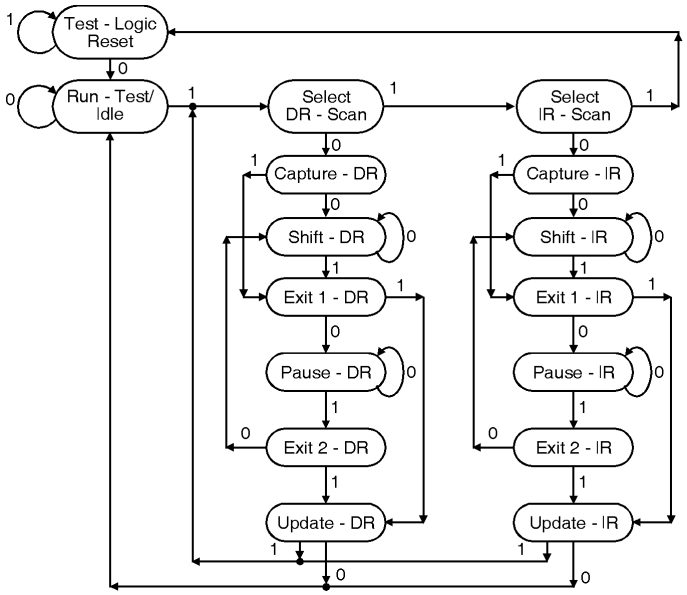



Figure 2.8: Architettura di un Tap Controller

ed è scambiata tra Eiger e Bernina utilizzando il periferico SIUL2, che consente di controllare tutte le porte di I/O di Eiger e supporta la comunicazione bi-direzionale di 16 bit general purpose in parallelo:

in questa architettura i pin necessari della porta C di Eiger sono configurati in I/O per scambiare i segnali descritti.

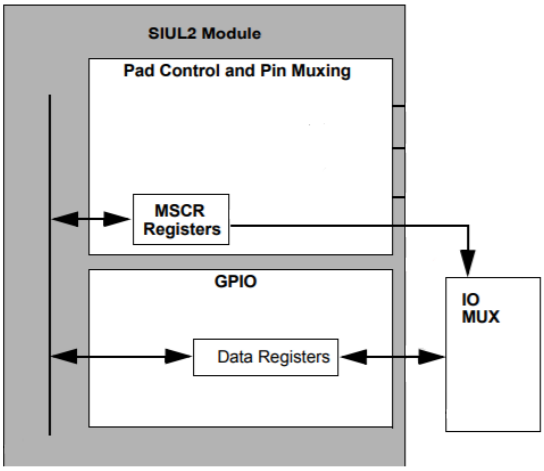



Figure 2.9: Archietettura SIUL2

Questa la sequenza che permette l'ingresso in test mode del DUT:


Figure 2.10: Cattura della squenza di test mode ottenuta con un Logic Analyzer

### 2.3.3 Intergity Check

Per verificare l'effettivo ingresso in Test Mode del DUT portiamo Bernina in Scan Mode, ovvero i Flip-Flops sono interconnessi tra di solo in una catena attraverso i SerialIn pin e il SerialOut pin, la prima cella della catena è collegata al TDI della JTAG, mentre l'ultima al TDO, e le operazioni sono temporizzate dal TCK.

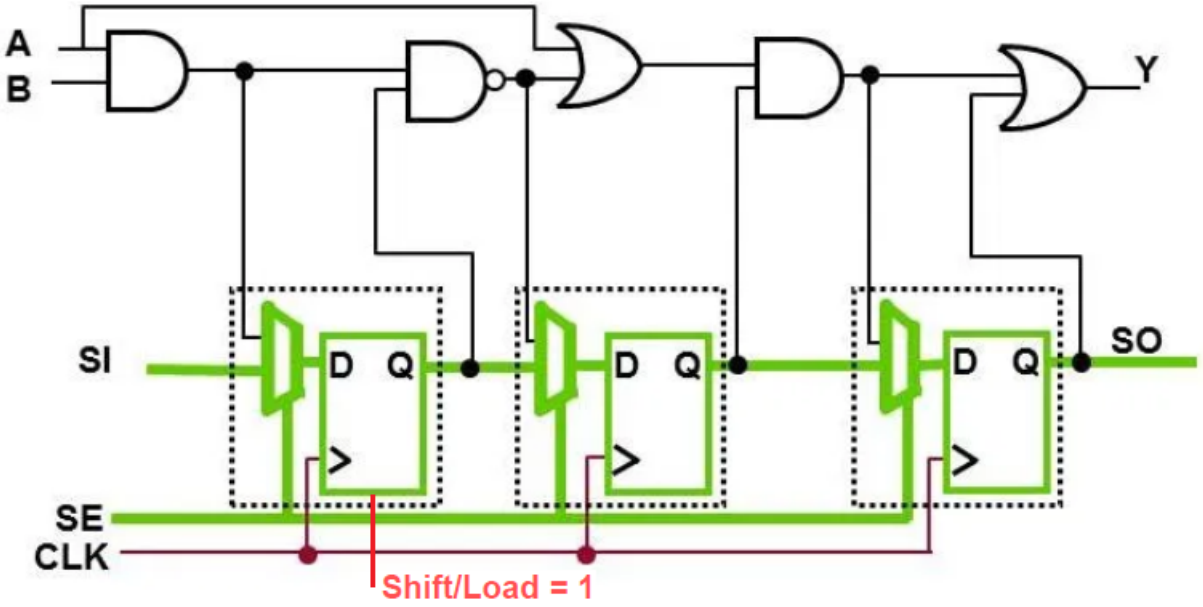



Figure 2.11: Scan Mode

Nella architettura predisposta questi 3 segnali essenziali derivano dal driver SPI del tester e i pin della porta C vengono risettati per implementare le funzionalità necessarie: SOUT dell' SPI di Eiger è connesso al TDI di Bernina ed emette in output dei bit di dato,che vengono shiftati ad ogni colpo di clock (SCK) dell'SPI, connesso al TCK di JTAG, e ogni output dal TDO della scan-chain è dato in input al SIN dell'SPI.

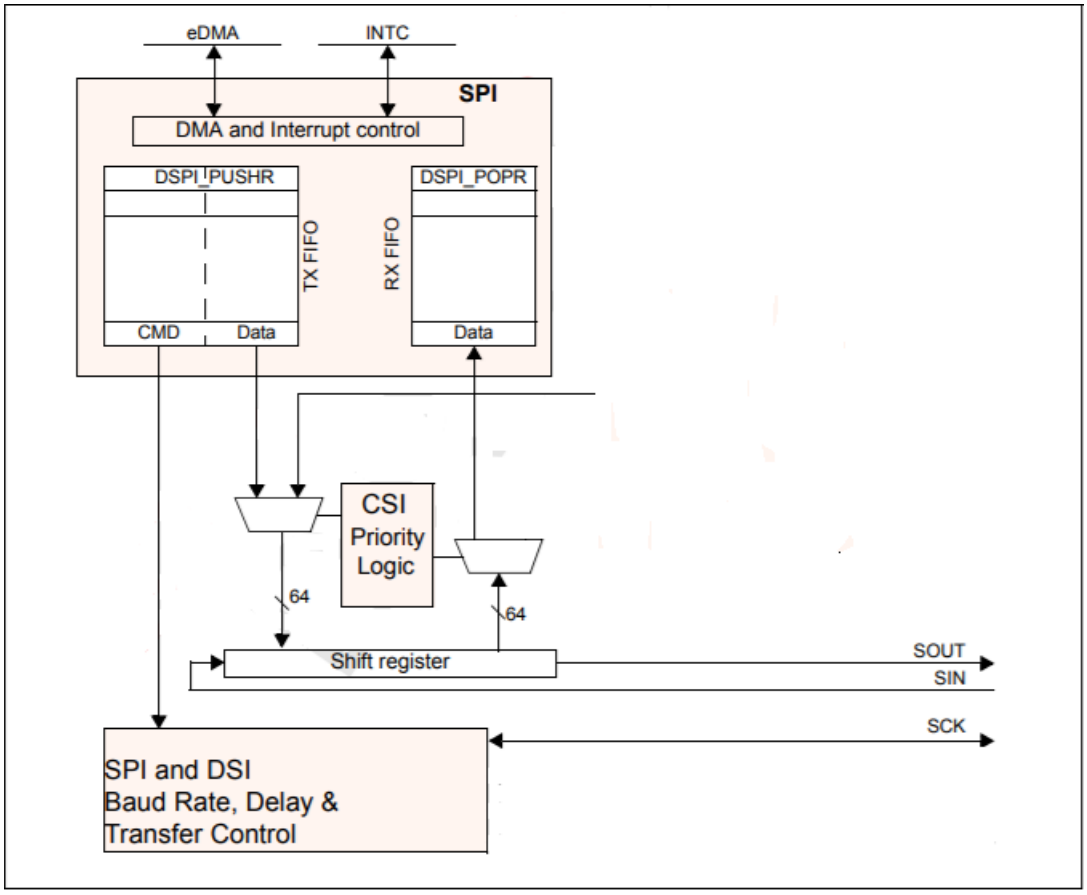



Figure 2.12: Architettura SPI

Il test effettuato di shift è denominato *Integrity Check* e come suggerisce il nome serve a verificare l'integrità della catena di scan:
viene inviata la stessa sequenza di bit per tutta la lunghezza della catena e si verifca che in output non ci siano errori.

La sequenza di bit inviata è 0001010011001111


Figure 2.13: Cattura pattern per integrity check ottenuta con un Logic Analyzer

e l'output viene immagazzinato in RAM di Eiger, dove possiamo verificarne l'integrità:

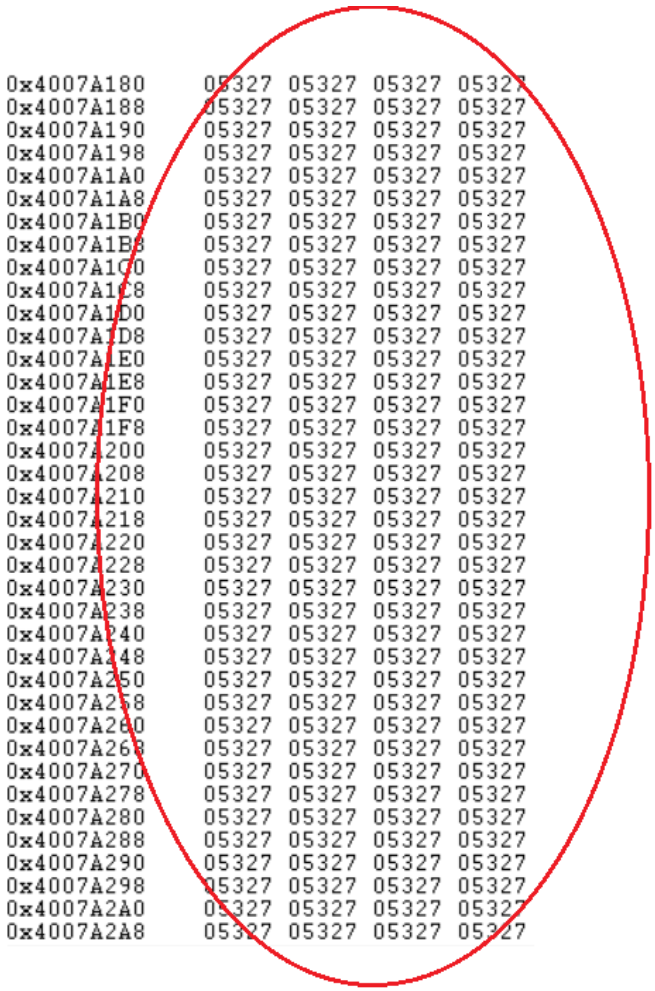



Figure 2.14: RAM contenente il download del pattern shiftato

```c
void integrity_check(uint16_t *rxbuf){

    /* Fill SPI TX buffer */
    uint16_t i;
    for(i=0;i<41343;i++){
        txbuf[i]=0xAAAA;//0x14CF;
     }
    /* Switch to a configuration of board for SPI communication */
    SPIDMAPinInit();
    /* Start SPI Driver */
    spi_lld_start(&SPID4,&spi_config_9Mhz);

    /* Chip Select Enable */
    spi_lld_select(&SPID4);
    /* Upload txbuf in scan chain */
    spi_lld_exchange(&SPID4, 41343, txbuf,rxbuf);
    while(!seq_ack);
    seq_ack=0;
    /* Download txbuf from scan chain */
    spi_lld_exchange(&SPID4, 41343, txbuf,rxbuf);
    while(!seq_ack);
    seq_ack=0;

    /* Chip Select Disable */
    spi_lld_unselect(&SPID4);
    /* Stop SPI Driver */
    spi_lld_stop(&SPID4);
    /* Print scan chain output in decimal format */
    UARTPinInit();
    printBuffer(&SD1,rxbuf,41343,0,2);
    printBuffer(&SD1,rxbuf,41343,41341,41343);
}
```

Figure 2.15: Codice C per l'Integrity Test

### 2.3.4 Capture

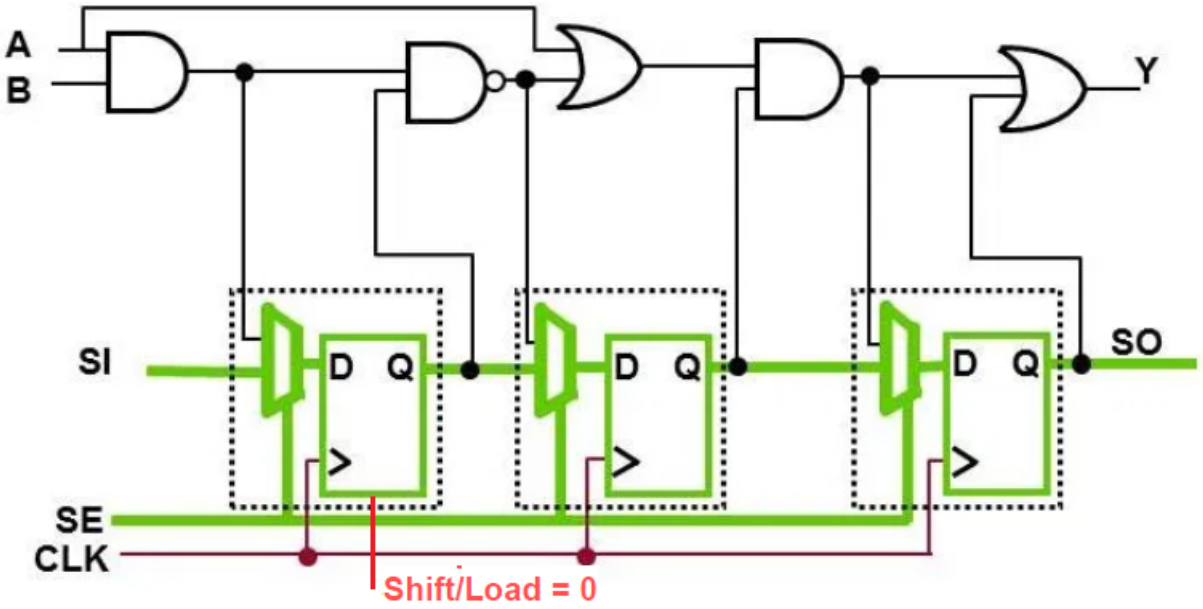



Figure 2.16: Capture Mode

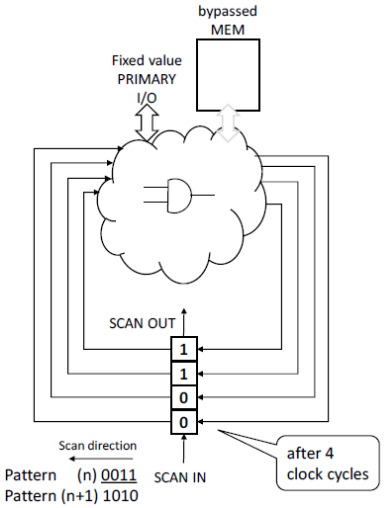
Una seconda tipologia di test condotta è quella denominata *Capture*:
il pattern è ricavato dall'**ATPG** e attraverso uno script in Python il file *STIL* viene convertito in una struct C e immaginazzinato nella RAM di Eiger. Il pattern viene a questo punto uplodato in scan-chain, viene negato lo Scan Enable, e avviene la cattura in un ciclo di clock ( EXTAL ) . in seguito viene asserito lo Scan Enable e viene downloadato il pattern cosi da verificarne la correttezza.

```c
void capture_test(void){
    /* Littardi-Restifo pattern */
    readPattern(pattern);
    readPattern(expected);
    /* Switch to a configuration of board for SPI communication */
    SPIDMAPinInit();
    /* Start SPI Driver */
    spi_lld_start(&SPID4,&spi_config_9Mhz);
    spi_lld_exchange(&SPID4, 41343, pattern,result);
    while(!seq_ack);
    seq_ack=0;
    spi_lld_exchange(&SPID4, 41343, result,pattern);
    while(!seq_ack);
    seq_ack=0;
    /* Configuration for TMS and EXTAL pins use */
    TMSExtalPinReset();
    /* Negate Scan Enable */
    pal_writepad(PORT_GPIO,PIN_GPIO_2,0);
    /* Perform a Capture through an EXTAL cycle */
    pal_writepad(PORT_GPIO,PIN_GPIO_4,1);
    pal_writepad(PORT_GPIO,PIN_GPIO_4,0);
    /* Assert Scan Enable */
    pal_writepad(PORT_GPIO,PIN_GPIO_2,1);
    /* Re-switch to configuration board for SPI */
    SPIDMAPinInit();
    spi_lld_exchange(&SPID4, 41343, pattern,expected);
    while(!seq_ack);
        seq_ack=0;
    /* Stop SPI Driver */
    spi_lld_stop(&SPID4);
    /* Check equality of expected and result*/
    uint8_t check = check_buffer(result,expected,41343);
    if(check==1){
        /* Equals */
    }
    else{
        /* Not Equals */
    }
}
```

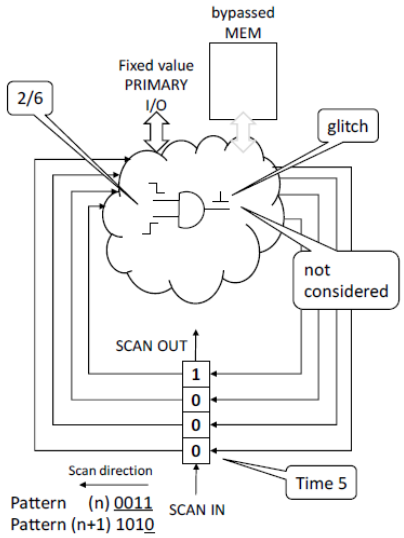
Figure 2.17: Codice C per il Capture Test

### 2.3.5 Burn-In completo

L'ultima tipologia di test è l'obiettivo della tesi: un pattern pseudo-random è generato dalla CPU e viene inviato alla catena di scan del DUT. La catena di scan viene shiftata completamente, viene anche fatto il capture dei valori, e raccolta in input nel Tester. I risultati di tali operazioni sono stati confrontati con i risultati ottenunti con gli stessi esperimenti sia con l'architettura nucelo che in simulazione, ottenendo dei risultati concordi.




(a)




(b)

Figure 2.18: Esempio di shift su 4 Flip-Flops

### 2.3.6 Compressione

Durante il Burn-in completo viene introdotta la compressione: ogni volta che un pattern viene scaricato dalla catena di scan del DUT la generazione del pattern successivo è sospesa fino a quando non è completata la creazione delle firma.

```
uint16_t MISRcompaction(uint16_t *buffer){
    //("\r\nCompacting...\r\n");
    unsigned index=0;
    int i;
    for(index=0;index<41343;index++){
         input = buffer[index];
         if(index<41342){
             for(i=0;i<16;i++){
                 result = ((input >> 15)^((state >> 1)^((state >> 21)^(state >> 31)))) &(1);
                 state = ( state << 1) | result;
                 input = (input << 1) ;
             }
         }
         else{
             for(i=0;i<5;i++){
                result = ((input >> 15)^((state >> 1)^((state >> 21)^(state >> 31)))) &(1);
                state = ( state << 1) | result;
                input = (input << 1) ;
            }
         }
    }

    /* The last state is the signature of the pattern passed with buffer */
    return state;
}
```

Figure 2.19: Codice C per comprimere i pattern

## 2.4 Architettura multi-core

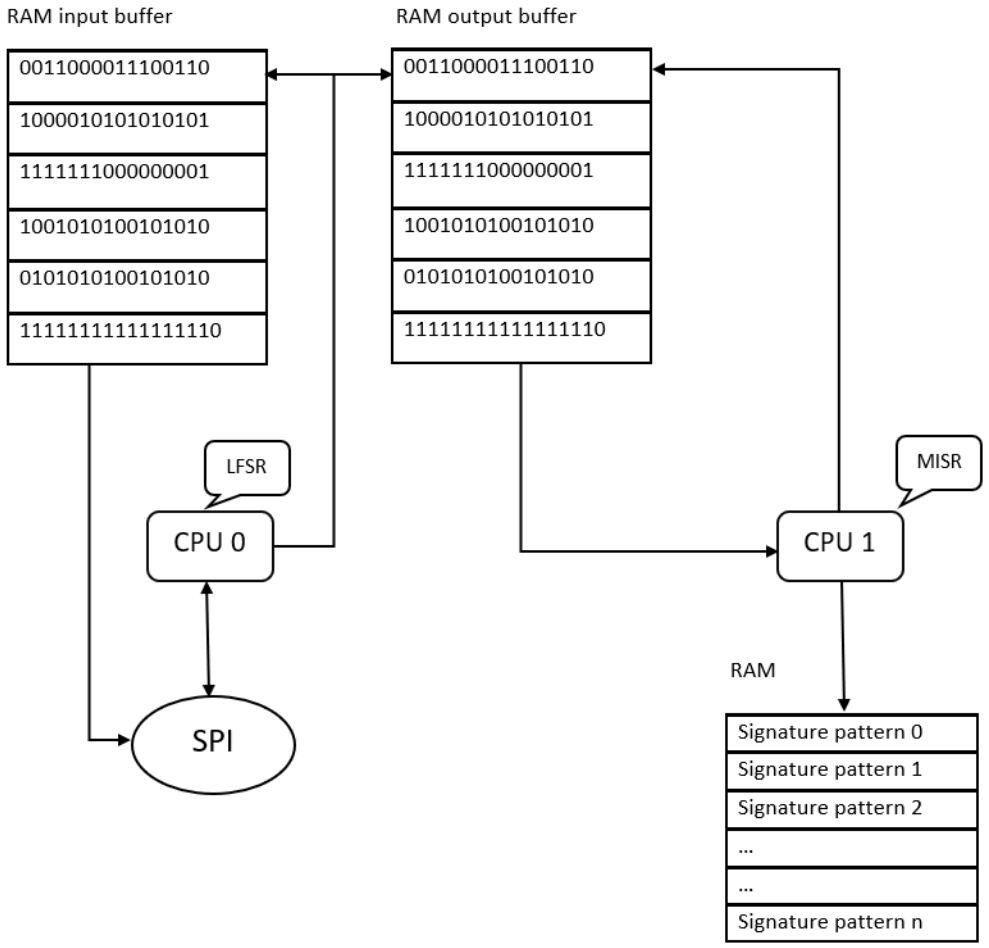



Figure 2.20: Architettura Eiger multi-core

Un upgrade dell'architettura precedentemente descritta è possibile avvalendosi della capacità **multicore** di Eiger : durante la fase di inizializzazione dei periferici viene quindi inclusa la direttiva per avviare un altro dei 3 cores disponibili.

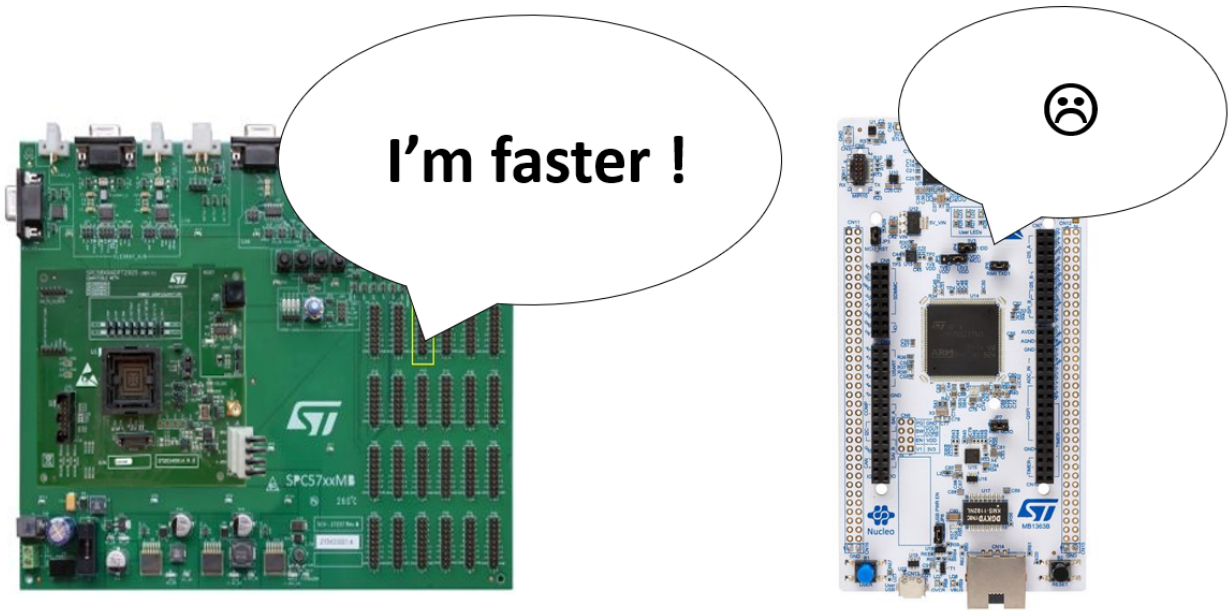
### 2.4.1 Parallelizzazione LFSR-SISR

L'introduzione di un secondo core permette di **parallelizzare** i processi di *generazione* e *compressione* dei pattern per il BI: un core si occupa di generare l'i-esimo pattern e scambiarlo con il DUT, intanto il secondo core comprime il pattern i-1, grazie ad una sincronizzazione favorita dall'inserimento di un semaforo che coordina le sezioni critiche del codice,aumentando considerevolmente le prestazioni dell'architettura e quindi la qualità del testing.

## 2.5 Risultati

L'avvento di un MCU multicore ha reso possibile generare e comprimere i pattern per il BI ad una frequenza di circa 9MHz, triplicando la velocità di un MCU single-core, ed inoltre l'utilizzo di Eiger consentirebbe di avere più DUT in parallelo grazie all'ingente numero di porte di GPIO disponbili.

L'introduzione di Eiger ha portato in beneficio anche una memoria più capiente per salvare le firme ottenute dalla compressione e permettere quindi una diagnosi sul sistema, rendendo accessibile la funzionalità di confrontare per ogni istante di tempo il funzionamento del DUT rispetto al funzionamento di un chip etichettato *GOOD*, ovvero funzionante, aggiungendo quindi ulteriore capacità di testing.

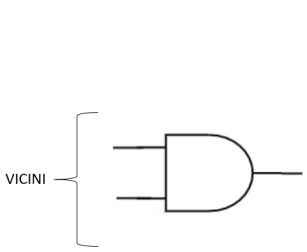


# Chapter 3

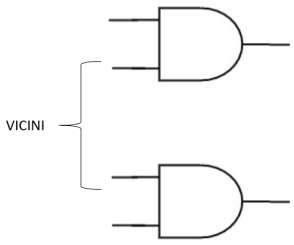
# Stress Statico

## 3.1 Modello di stress Topologico

Nel corso del progetto di ricerca è nata l'idea di valutare la toggle activity sfruttando una tipologia differente di stimolazione, da noi ribattezzata *topologica*, in quanto strettamente correlata alla struttra fisica del DUT.
Esplorando il posizionamento dei gates è possibile estrarre una lista di coppie di nodi definibili vicine ogni qual volta rispettano una distanza di soglia:




(a)




(b)

Il modello di stress proposto prende ogni coppia della lista ricavata, confronta questi nodi con i dati di un EVCD ricavato da una simulazione per verificare se hanno assunto due valori logici opposti allo stesso istante di tempo, comportamento che scatena dei fenomeni di affinità elettromagnetica.

Una coppia è catalogata totalmente togglata quando entrambe le possiblità di valori logici opposti (1|0 e 0|1) emergono dall'analisi del pattern simulato, e in questo caso di studio la distanza massima che possono avere due nodi per essere considerati vicini è di 6 micrometri e il tempo di stasi dei valori logici è di 500 ns.
Determinare le coppie di nodi vicine è il processo più oneroso computazionalmente in questa tipologia di stimolo, e il mio lavoro si è concentrato principalmente nel ridurre questo tempo di computazione cercando di ottenere il minor errore possibile rispetto all'approccio esaustivo preesistente attraverso un approccio *deduttivo* basato

sul machine learning.

### 3.1.1 Approccio esaustivo

L'approccio deduttivo è basato su elaborare l'intera lista dei nodi del circuito, circa 20 milioni di entries, presi singolarmente e confrontati con tutti gli altri per determinare la vicinanza: ovviamente su una mole di dati così grande il tempo di computazione è costoso, impiegando all'incirca 20 giorni per l'estrazione delle coppie vicine.

### 3.1.2 Approccio deduttivo

La mia idea è stata quella di considerare i nodi dei gate che compongono il SoC come coordinate (x,y) nello spazio cartesiano, e possiamo dunque classificarli per vicinanza usando come metrica la distanza euclidea.

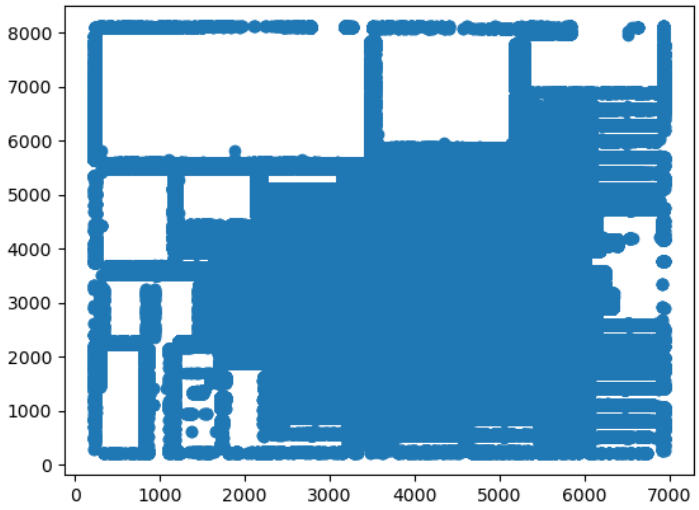



Figure 3.1: SoC nello spazio cartesiano

### 3.1.3 Density-based scan

La tecnica di classificazione utillizzata è basata sull'algoritmo DBSCAN, un metodo di clustering basato sulla densità, che connette regioni di punti con densità sufficientemente alta.

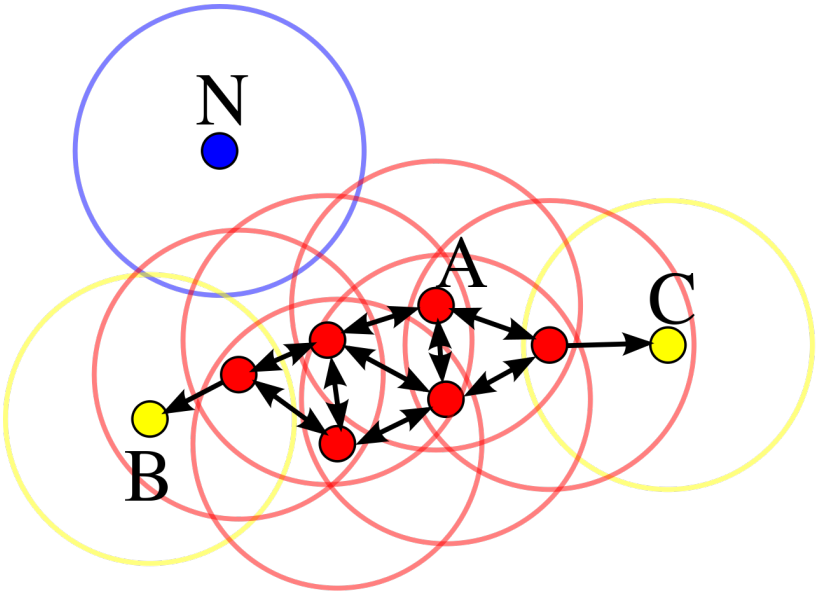



Figure 3.2: dbscan background

Questo algoritmo con densità intende il numero di punti entro un prefissato raggio *Eps*. Per descrivere il processo di classificazione è indispensabile introdurre alcuni concetti:

- **Core Point** : un punto che entro il raggio eps ha vicino uno specificato numero di punti, il parametro *MinPts* ; questi sono i punti interni al cluster;

- **Border Point**: nel suo intorno eps ha un numero di punti inferiore a MinPts ma è a una distanza ravvicinata ad un core point;

- **Noise Point**: ogni punto che non rientra nelle altre due categorie.

La scelta del DBSCAN deriva dalla considerazione che il device possiede diverse densità di gate nelle sue varie zone e si è ipotizzato si prestasse bene all'obiettivo prefissato.

Un ambiente Conda scritto in Python denominato **ClassificatorEnv** è stato implementato:
-per generare i cluster dei nodi vicini viene sfruttata la libreria openSource *skleарn*, settando i parametri **Eps** e **MinPts**
-per determinare i nodi che rispettano la distanza soglia che desideriamo e per calcolare la toggling activity del circuito ricavata dall'analisi delle simulazioni di shift completo della catena di scan uno script in Python multi-processo viene eseguito.

### 3.1.4 Prestazioni

Il vantaggio principale di utilizzare il machine learning per generare la fault list dei vicini è nella velocità di esecuzione:

```
(ClassificatorEnv) [g.gallo@cattivo ~]$ python dbscan.py
read_time: 74.4061758518219
Start DBSCAN algorithm at: 110.884938955307
Classification end at: 203.147522687912
Estimated number of cluster with 1 point:462774
Estimated number of cluster with 2 points:647305
Most populed cluster is: 317
Less populed cluster is: 1
Average population in cluster is: 5.811174598193596
Estimated number of couple: 105514852.0
Estimated clusters: 3441645
Estimated noise points: 0

Execution time: 543.2523376941681
```

Figure 3.3: Esecuzione script Python

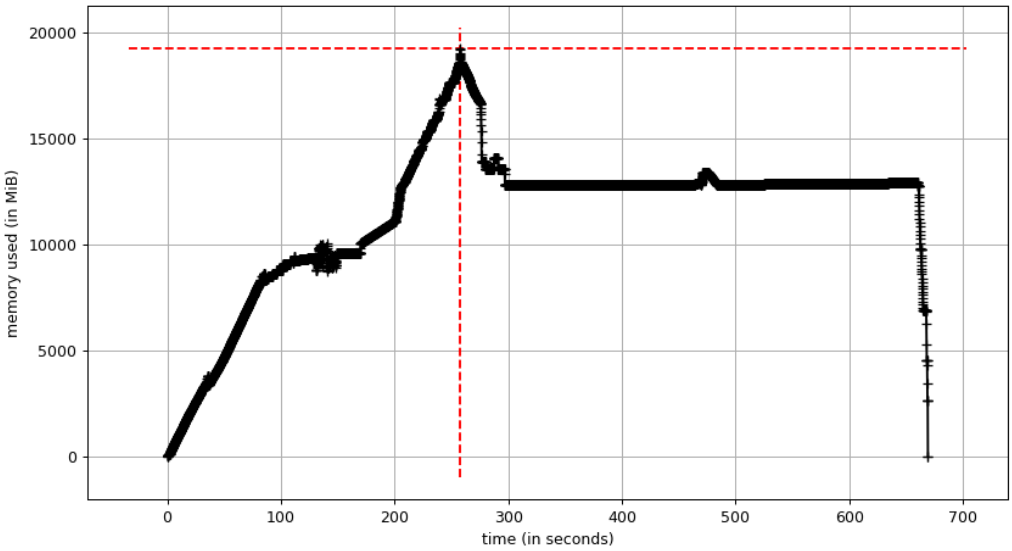



Figure 3.4: Andamento consumo di memoria durante la generazione della fault list

Dalla valutazione delle simulazioni è possibile calcolare una coverage statica partendo da una fault list creata dalla combinazione di tutte le possibili coppie di nodi appartenenti ad ogni cluster che rispettano una distanza di soglia massima (nel nostro caso 0,8 nm), ottendo cosi una idea di copertura del SoC nelle sue varie aree:

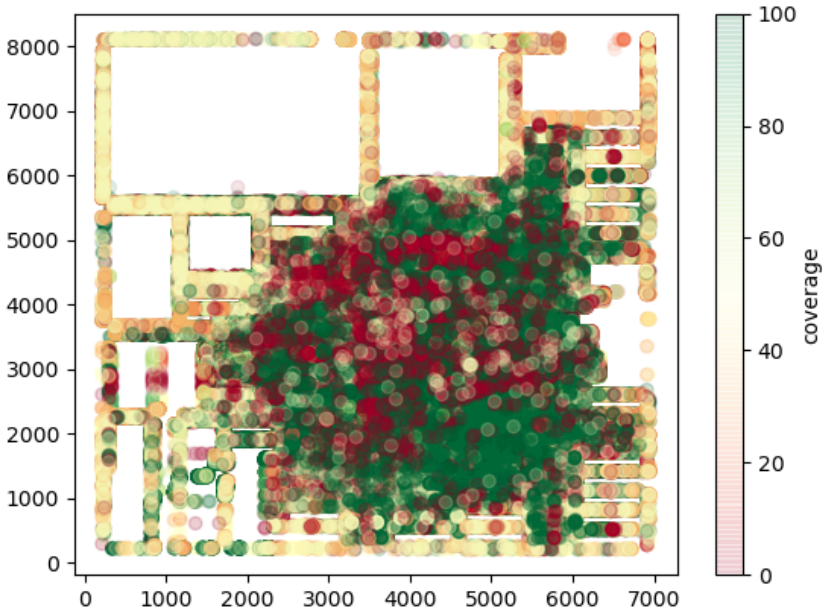



Figure 3.5: Cluster coverage 32 pattern random

### 3.1.5 Calcolo multiprocesso toggling activity

Con l'avvento dei cluster risulta estremamente semplice parallelizzare su più processi il lavoro di calcolo di toggling activity del circuito: l'ID di ogni cluster serve a distribuire i processi sui vari core dell'unità di elaborazione e in questo modo è possibile ottimizzare ancora di più la quantità di tempo necessario ad ottenere i risultati del modello di guasto proposto.

```
def distance(x1,y1,x2,y2):
     dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
     return dist

def analisi_cluster(indice):
  group = grouped.get_group(indice) #prendo i-esimo cluster e salvo su una lista
  n_elements=len(group)
  lista=[]
  data={}
  if(n_elements>1):
    for index,rows in group.iterrows():
      my_list = [rows.node,rows.x,rows.y,rows.res]
      lista.append(my_list)
      data[rows.node]=rows.res

    count=0
    fault_coperti=0
    for k in range(len(lista)):
      for z in range(k,len(lista)):
        if(lista[k]!=lista[z]):
          if(distance(lista[k][1],lista[k][2],lista[z][1],lista[z][2])<0.8):
             count=count+1
             nodoa=data[lista[k][0]]
             nodob=data[lista[z][0]]

             res_nodoa=nodoa.split('|')
             res_nodob=nodob.split('|')

             last_a=None
             last_b=None
             flag_a=0
             flag_b=0
             for pttrn in range(32):
               d1= res_nodoa[pttrn]
               d2= res_nodob[pttrn]

               if(d1[-1]!='-'):
                 last_a = d1[-1]
               if(d2[-1]!='-'):
                 last_b = d2[-1]
```

```
                if(d1=='-'):
                  d1=last_a

                if(d1=='-'):
                  d2=last_b

                if(flag_a==0):
                  if(d1[-1]=='0'):
                    if(d2[-1]=='1'):
                      fault_coperti=fault_coperti+1
                      flag_a=1
                if(flag_b==0):
                  if(d1[-1]=='1'):
                    if(d2[-1]=='0'):
                      fault_coperti=fault_coperti+1
                      flag_b=1
            fault_coverage=float(100*fault_coperti/(count*2))
    n_coppie=n_coppie+count
  #insertimento nella lista gestibile in parallelo
  df_tmp.append([indice,fault_coverage])

start_time=time.time()
df = pd.read_csv("ResultsOFClusteringATPG.csv")
print("Read file end at:"+str(time.time()-start_time))

n_coppie=0
n_cluster = max(df.cluster)
utile={}

grouped = df.groupby('cluster')

# per l'accesso concorrente non potevo usare un DataFrame ->
# usato una lista di liste dove la prima contiene i nomi delle colonne
df_tmp=Manager().list([["cluster", "coverage"]])

print("Start compute coverage at:" +str(time.time()-start_time))

i = 0
hmm = 0
sganciati = 0
processo = None
```

```
    while i < int(n_cluster)+1:

      # sganciati = il numero di processi che sono riuscito ad avviare
      sganciati = 0

      # fili è una lista di processi
      fili = []

      # ciclo che cerca di sganciare n processi
      for hmmm in range(10):

            if(i < int(n_cluster) + 1):

              # creo processo
              processo = multiprocessing.Process(target=analisi_cluster, args=(i, ))

              # appendo processo a lista processi
              fili.append(processo)

              # incrementa indice di cluster
              i = i + 1

              # incrementa numero di processi avviati
              sganciati = sganciati + 1

      # se sono riuscito a creare dei processi
      if(sganciati != 0):

        # esegui la start di ogni elemnto nella lista "fili" -> avvio i processi
        for finito_i_nomi in range(sganciati):
          fili[finito_i_nomi].start()

            # esegui la join su tutti i processi avviati
        for finito_i_nomi in range(sganciati):
          fili[finito_i_nomi].join()


    # estraggo nomi delle collonne nella lista riempita in parrallelo
    column_names = df_tmp.pop(0)

    # converto lista in dataframe
    df_tmp = pd.DataFrame(df_tmp, columns=column_names)

    # join eseguita come in precedenza
    df=pd.merge(df,df_tmp,on='cluster',how='right')


print("Compute coverage ends, and plot creation start, at:" +str(time.time()-start_time))
print("Number of couple:" + str(n_coppie))
plt.figure('Bernina nodes')
plt.scatter(df['x'],df['y'],c=df['coverage'],alpha=0.1,cmap='RdYlGn')
#plt.scatter(df['x'],df['y'])
cbar=plt.colorbar()
cbar.set_label("coverage",labelpad=+1)
#plt.show()
plt.savefig('Coverage32atpg.png')
print('Execution time' +str(time.time()-start_time))
```

### 3.1.6 Risultati

Questa analisi *deduttiva* individua 29644712 coppie, a fronte del risultato *esaustivo* pari a 30920599, e dunque possiamo parlare di una **accuratezza** del 95,9% per l'individuazione delle coppie.

```
Read file end at:52.127930879592896
Start compute coverage at:54.651451110839844
Compute coverage ends, and plot creation start, at:27865.646602630615
Number of couple:29644712
```

Figure 3.6: Toggling activity rispetto ai clusters

A seguire la matrice di confusione:

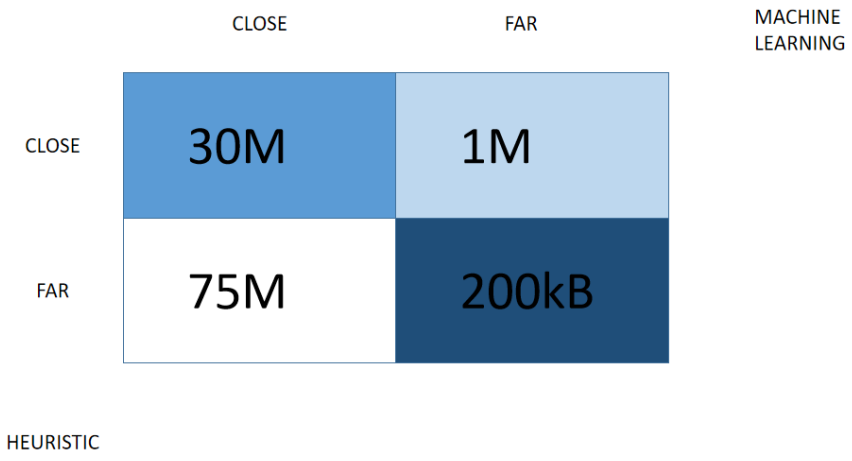



Figure 3.7: Matrice di confusione

# Chapter 4

# Conclusioni

Questo lavoro di tesi rientra in un ampio progetto di ricerca nell'ambito delle tecniche di System Level Test e ci sono tantissime possibilità di affinamento degli ottimi risultati ottenuti.

### 4.0.1 Conclusioni architettura

L'architettura è gia predisposta per implementare altre modalità di test, unica azione necessaria è modificare l'emulazione della sequenza di tap ( vedi 2.3.2 )
Può essere di esempio la modalità **Delay**:

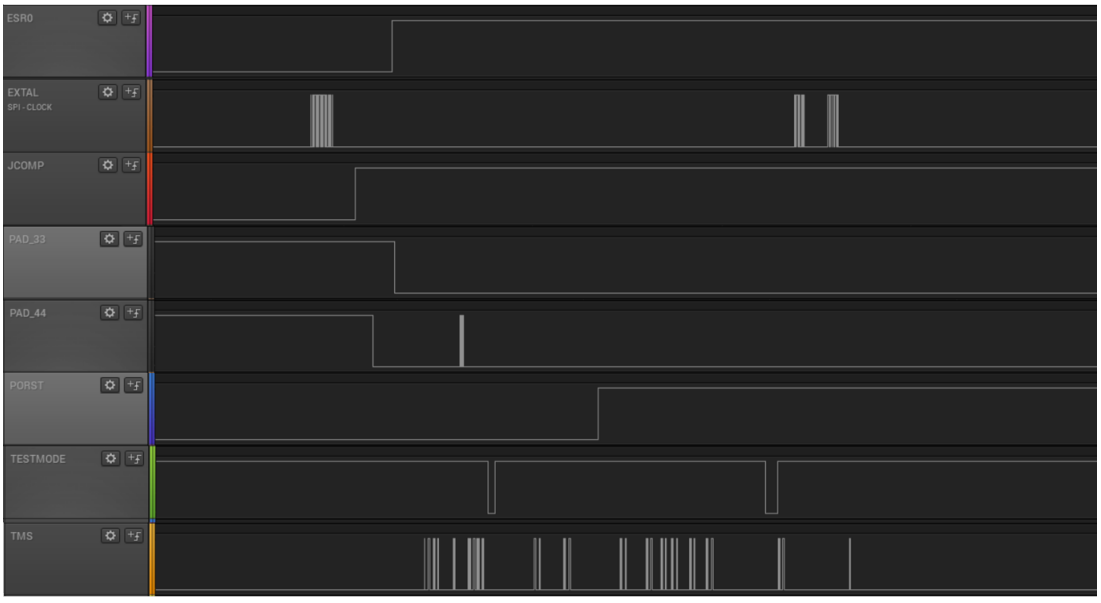



Figure 4.1: Cattura della squenza di delay mode ottenuta con un Logic Analyzer

L'architettura per il BI potrebbe essere migliorata con l'inserimento di una FPGA per demandare alcune delle operazioni delegate alla MCU-Tester in modo da incrementare in maniera ancora più sensibile le prestazioni, in termini di frequenza e memoria disponibile

### 4.0.2 Conclusioni stress topologico

L'approccio *machine learning* può essere investigato maggiormente utilizzando altre metodologie di clustering o impelmentando degli algoritmi di classificazione di altro genere e in seguito confrontare i risultati per ottenere una precisione ancora maggiore.

# Bibliography


[1] Sounil Biswas; Bruce Cory, “An Industrial Study of System-Level Test”, IEEE Design & Test of Computers, 2012, vol. 29, no. 1, pp. 19 – 27

[2] Dilip Kumar Reddy Tipparthi; Karthik Krishna Kumar, “Concurrent system level test (CSLT) methodology for complex system-on-chip”, 2014 IEEE 16th Electronics Packaging Technology Conference (EPTC)

[3] A. Birolini, “Reliability Engineering Theory and Practice,” Heidelberg: Springer, 2017

[4] International Standard - IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems, International Electrotechnical Commission, 2010.

[5] ISO 26262-[1-10], Road vehicles – Functional safety, 2011.

[6] M. F. Zakaria et al., "Reducing burn-in time through high-voltage stress test and Weibull statistical analysis," IEEE Design & Test of Computers, vol. 23, no. 2, pp. 88-98, March-April 2006.

[7] D. Appello et al., "A comprehensive methodology for stress procedures evaluation and comparison for Burn-In of automotive SoC," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, 2017, pp. 646-649.

[8] X. Guo, W. Burleson and M. Stan, "Modeling and experimental demonstration of accelerated self-healing techniques," 2014 ACM/EDAC/IEEE Design Automation Conference (DAC), 2014, pp. 1-6.

[9] A. Benso, A. Bosio, S. D. Carlo, G. D. Natale and P. Prinetto, "ATPG for Dynamic Burn-In Test in Full-Scan Circuits," 2006 15th Asian Test Symposium, Fukuoka, 2006, pp. 75-82.

[10] Chen He, “Advanced Burn-In - An Optimized Product Stress and Test Flow for Automotive Microcontrollers”, IEEE International Test Conference, 2019

[11] F. Almeida et al., "Effective Screening of Automotive SoCs by Combining Burn-In and System Level Test," 2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Cluj-Napoca, Romania, 2019, pp. 1-6

[12] https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html

[13] https://it.wikipedia.org/wiki/Dbscan

[14] https://www.electronics-tutorial.net

[15] https://semiengineering.com

[16] Matteo Sonza Reoda, Materiale del corso "Testing and fault tolerance"

[17] Maria Elena Baralis, Materiale del corso "Data science e tecnologie per le basi di dati"

[18] https://www.advantest.com/documents/11348/f717e957-0326-42d1-a13c-921059722fc6