Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Informazione

Tesi di laurea triennale

Sviluppo di una periferica hardware per misure di soft error su SRAM

Laureando:
Stefano Dalla Bontà
Matricola INF-591449

Relatore:
Ch.mo Prof. Alessandro Paccagnella

Correlatore:
Ing. Simone Gerardin

Anno Accademico 2011–2012

Indice

Introduzione

Il progetto presentato in questa tesi prevede la realizzazione di una periferica che rilevi la presenza di soft error su SRAM e si inserisce in un programma più ampio dedicato agli effetti delle radiazioni sui circuiti elettronici. In questo caso una memoria, di cui sono noti i valori presenti nei vari registri, viene esposta a radiazioni, successivamente si esegue una scansione per controllare l'eventuale variazione dei dati. É necessario quindi che la periferica preveda dei comandi di scrittura per poter effettuare la preparazione della SRAM prima dell'esposizione e dei comandi di lettura per il controllo finale. La periferica verrà controllata attraverso una CPU programmata in linguaggio VHDL.
Nel primo capitolo viene fornita un'analisi di come le radiazioni possano indurre delle variazioni di carica e quindi malfunzionamenti nei circuiti. Vengono quindi descritti i principali tipi di errori, quali sono le tipologie di radiazioni che influiscono sul comportamento dei circuiti e una breve analisi sulla sensibilità dei dispositivi elettronici a questi fenomeni.
Nel secondo capitolo vengono presentati i due dispositivi con cui la periferica dovrà lavorare. Il primo dispositivo è la memoria utilizzata per questo progetto, ovvero la SRAM IS61LV5128AL di ISSI®. Il secondo dispositivo è una CPU programmata in VHDL che attraverso dei comandi controllerà la periferica. In realtà la comunicazione non avviene in maniera diretta ma attraverso il bus LogiCORE^{TM} IP FSL v20 di Xilinx®. Verrà quindi analizzato il funzionamento del bus invece che quello della CPU.
Nel terzo capitolo viene descritta la struttura del codice VHDL che realizza la periferica: la macchina a stati finiti o FSM. Inoltre vengono portati dei primi esempi di codice e del funzionamento pratico della periferica.
Nel quarto capitolo vengono descritti i comandi di scrittura che permettono di preparare la SRAM all'esposizione alle radiazioni. Il codice viene commentato analizzando le funzioni dei singoli stati che realizzano il comando.
Infine nel quinto capitolo vengono analizzati i comandi di lettura, che permettono di rilevare gli eventuali errori dovuti alle radiazioni.

Capitolo 1

Radiazioni e Soft Error

1.1 Tipi di errore dovuti alle radiazioni

Man mano che le dimensioni e le tensioni operative dei circuiti integrati vengono ridotte per soddisfare il mercato, che richiede maggior densità e minor consumo di potenza, aumenta la loro sensibilità alle radiazioni. Le radiazioni sono la causa di diverse tipologie di errore nei dispositivi a semiconduttori, ognuna delle quali ha diversa importanza. Di primo interesse sono gli errori dovuti ad una singola esposizione alle radiazioni, chiamati SEE (single-event-effects), che si suddividono in hard e soft. I primi danneggiano permanentemente il circuito e vengono trattati prevalentemente in campo militare e spaziale. I secondi, invece, sono temporanei e si limitano a variare dei dati memorizzati, a invertire lo stato di un flip-flop o un latch. La successiva operazione di *storage* viene comunque eseguita con successo e quindi i soft-errors vengono anche chiamati SEU (single event upset). La categoria dei soft-errors prevede varie tipologie di errore, alcune di queste vengono qui presentate.

- Se l'evento radioattivo è ad energia molto alta è possibile che questo abbia l'effetto di cambiare lo stato di molti bit contemporaneamente. Questa tipologia di errore è conosciuta come MBU (multibit upset).
- Se invece l'evento radioattivo è a bassa energia è più probabile che produca un errore su un singolo bit. Questo errore viene chiamato SBU (single bit upset).
- É possibile che l'errore sul singolo bit riguardi un registo critico come, ad esempio, quelli presenti in una FPGA (field-programmable gate arrays) oppure nel circuito di controllo di una DRAM (dynamic random access memory). Questi errori vengono chiamati SEFI (single event interrupt) e il loro effetto è quello di introdurre un malfunzionamento del circuito in maniera diretta, al contrario degli errori sui dati memorizzati che possono essere trattati con algoritmi di correzione d'errore o risultare ininfluenti per il risultato finale delle operazioni.
- Se la radiazione cambia lo stato di un bit all'interno di una logica combinatoria è possibile che questo si propaghi fino all'uscita dando un errore sul risultato. Questo tipo di errore viene chiamato SET (single event transient)

- In una tecnologia CMOS è possibile che venga acceso il transistor BJT parassita posto tra well e substrato. A volte è possibile ripristinare il circuito spegnendo l'intero chip, in altri casi invece il danno è permanente e allora l'errore è di tipo hard. Questa situazione prende il nome di latch-up e di conseguenza si parla di SEL (single event latch-up).

1.2 Effetti delle radiazioni sul silicio

Nell'ambiente terrestre i soft errors sono dovuti a ioni energetici. L'importanza dei disturbi dipende dal trasferimento lineare di energia o LET dello ione. Questa tende ad essere maggiore quando la particella ha maggior massa e energia, e sta attraversando un materiale denso. Per quanto riguarda il silicio viene generata una coppia elettrone-lacuna ogni 3.6 eV (elettronvolt) di energia persa dallo ione.

Il punto più critico di un circuito sono le giunzioni pn polarizzate inversamente, in particolar modo se è flottante o in stato di alta impedenza. L'effetto di uno ione che attraversa una porzione di silicio drogato è quella di generare, a distanze inferiori al micron dalla sua traiettoria, delle coppie elettrone-lacuna. Quando lo ione passa vicino alla regione di carica spaziale della giunzione polarizzata inversamente i portatori, appena prodotti, vengono raccolti dal campo elettrico presente. Avviene una distorsione del potenziale a forma d'imbuto che permette alla regione di carica spaziale di estendersi nel substrato. Questa fase dura pochi nanosecondi, in seguito la diffusione inizia a contrastare il processo di raccolta. Dopo qualche centinaio di nanosecondi tutti i portatori sono stati raccolti, si sono ricombinati o sono stati allontanati dalla giunzione per effetto della diffusione. L'effetto totale è quindi quello di un impulso di corrente.

La carica così raccolta viene denominata come Q_{coll} ed è tanto più grande quanto è vicino il passaggio dello ione rispetto alla giunzione. La carica raccolta dipende anche da altri fattori quali: la grandezza del dispositivo, polarizzazione dei vari nodi del circuito, struttura del substrato, drogaggio del dispositivo, dal tipo di ione, dalla sua carica, dalla sua traiettoria e dalla sua posizione iniziale. Inoltre, in un circuito, un nodo non è mai isolato, dando origine a fenomeni di *charge sharing* che influenzano la reale carica raccolta.

Dopo aver definito Q_{crit} come la carica richiesta per cambiare lo stato del dato ci sono due situazioni possibili

1. $\mathbf{Q_{coll} < Q_{crit}}$: in questo caso la radiazione non produce alcun soft error e il circuito si comporta normalmente.

2. $\mathbf{Q_{coll} > Q_{crit}}$: l'evento radioattivo avviene sufficientemente vicino alla giunzione da indurre un soft error.

É importante notare che Q_{crit} non è costante perchè dipende dalla risposta dinamica del circuito e dall'impulso della radiazione.

Per quanto riguarda i soft error vengono utilizzate due unità di misura. La prima è la frequenza con cui questi si verificano ed è chiamata SER (soft error rate) mentre la seconda si chiama FIT (failure in time) ed è definita come un fallimento ogni 10^9 ore di funzionamento. I soft error sono attualmente tra le maggiori cause di errore dei circuiti elettronici e possono superare i 50'000 FIT/chip.

1.3 Le radiazioni nell'ambiente terrestre

In questa sezione verranno analizzati i tre principali fenomeni che danno origine alle radiazioni che inducono i soft errors nei circuiti elettronici.

1.3.1 Le particelle alfa

Le particelle alfa vengono emesse dalle impurità, uranio e torio, presenti nei materiali con cui sono costruiti i *package* dei circuiti integrati. Queste particelle attraversano il silicio ionizzandolo fino a quando non perdono tutta la loro energia cinetica. Normalmente la strada percorsa non supera i 100 μm permettendo di prendere in considerazione solo le impurità presenti nel *package*. Attualmente tutti i materiali semiconduttori sono altamente purificati, tuttavia questa è solo una condizione necessaria per avere una bassa emissione di particelle alfa. Un elemento radioattivo infatti non decade direttamente in un nucleo stabile ma dà origine ad una serie di decadimenti successivi, chiamati catena, fino ad arrivare ad un isotopo stabile. Dato che gli stati intermedi spesso emettono più radioattività dell'isotopo originale è possibile che, anche con un basso numero di impurità, vengano emesse parecchie particelle alfa.
Ci sono due metodi per ridurre la frequenza di soft error nei circuiti integrati:

1. anche se non è una condizione sufficiente per ridurre l'emissione di particelle alfa, vengono purificati tutti i materiali utilizzati per la costruzione del *package* del circuito integrato.

2. si cerca di ridurre la probabilità che le particelle alfa emesse dai materiali riescano a raggiungere i punti critici del circuito integrato.

I produttori di circuiti integrati devono sempre controllare la produzione e lavorazione dei materiali per eliminare le maggiori cause di contaminazione. Questi processi hanno permesso di ridurre la frequenza di emissione di particelle da 100 α/h cm^2 a meno di 0.001 α/h cm^2. Un materiale soddisfa le specifiche ULA (ultra low alpha) quando le sue emissioni sono inferiori a 0.002 α/h cm^2. Si ricorda però che in una catena di decadimento la particella figlio emette più particelle alfa della particella padre, percui, prima di immettere il materiale nel mercato, è necessario effettuare un controllo delle emissioni per diversi mesi. Grazie a queste accortezze i dispositivi CMOS sono sufficientemente purificati da ridurre a meno del 20% i soft error dovuti alle particelle alfa. Attualmente una riduzione di questa percentuale è eccessivamente costosa e poco redditizia dato che la SER è dominata dai raggi cosmici.

1.3.2 Raggi cosmici ad alta energia

La seconda causa di soft error è collegata ad eventi dovuti ai raggi cosmici. Questi interagiscono con l'atmosfera terrestre creando una complessa cascata di particelle secondarie. A loro volta le particelle secondarie creano una cascata di particelle terziare e così via. Solo l'1% delle particelle che arrivano al livello del mare fa parte del flusso primario e il flusso totale, che viene considerato isotropico, è composto da protoni, neutroni e muoni. I neutroni sono tra i componenti maggiori del flusso e, dato che hanno un più alto trasferimento lineare di energia, sono la causa principale dei soft error dovuti ai raggi cosmici.

L'intensità del flusso di neutroni dipende prevalentemente da due fattori:

1. l'altezza a cui si trova il dispositivo. A 3 Km di altezza il flusso del raggio cosmico può essere anche 10 volte superiore incrementando la frequenza dei soft errors.

2. la posizione geografica del territorio. L'effetto scudo del campo magnetico terrestre rende il flusso di neutroni dipendente dalla rigidità magnetica della posizione. Questo fattore è meno pronunciato rispetto al precedente.

I neutroni comunque non generano direttamente una ionizzazione del semiconduttore con cui sono composti i chip e quindi non provocano direttamente un soft error. Tuttavia possono interagire con il silicio in maniera elastica e anelastica. Nel secondo caso si ha una reazione nucleare tra il neutrone e un atomo di silicio, il cui nucleo viene frammentato e produce una carica di disturbo anche maggiore di 100 fC sufficiente a produrre un soft error. La LET prodotta da questa interazione è molto più significante rispetto a quella prodotta dalle particelle alfa rendendo i raggi cosmici più pericolosi per i circuiti integrati.

Un altro aspetto critico dei raggi cosmici è che questi, a differenza delle impurità, non possono essere ridotti in maniera efficace. Diversi decimetri di calcestruzzo forniscono un buono scudo ai raggi cosmici consentendo di minimizzare il loro effetto sui circuiti appartenenti a dispositivi fissi come stazioni base, sistemi centrali, etc. É chiaro però che per i dispositivi mobili questa soluzione non può essere presa in considerazione, rendendo necessaria una riduzione della sensibilità attraverso una modifica del design dei circuiti integrati.

1.3.3 Raggi cosmici a bassa energia

La terza sorgente di particelle ionizanti nei dispositivi elettronici è l'interazione tra i neutroni dei raggi cosmici a bassa energia e il boro. Il boro è ampiamente utilizzato come dopante di tipo p dei semiconduttori, soprattutto silicio, e nella formazione del BPSG (boron doped phospholicate glass) utilizzato come dielettrico. Il boro in natura ha due isotopi stabili, ^{11}B e ^{10}B. Quest'ultimo diventa instabile quando è esposto a neutroni e, a differenza della maggior parte degli isotopi, dopo aver assorbito un neutrone il suo nucleo si rompe e rilascia energia sottoforma di una particella alfa e di un nucleo di ^{7}Li eccitato. Queste due particelle viaggiano in direzione opposta e hanno entrambe sufficiente energia per indurre un soft error, in particolar modo nei circuiti a basso voltaggio.

La quasi totalità dell'isotopo ^{10}B nei circuiti integrati appartiene allo strato di dielettrico BPSG e non al drogaggio del silicio. Inoltre la particelle alfa e di litio hanno energia sufficiente a generare un soft error solo nei primi 0.5μm della loro traiettoria e quindi solo la parte di BPSG estremamente vicina al substrato di silicio viene trattata come una minaccia.
I sistemi per diminuire gli effetti dei raggi cosmici a bassa energia sono:

- eliminare semplicemente il BPSG dal processo di produzione.

- sostituire il BPSG vicino al substrato di silicio con un altro dielettrcio privo dell'isotopo ^{10}B.

- nel caso in cui siano necessarie le proprietà del boro è possibile sostituire il normale BPSG con un ^{11}BPSG arricchito. Le proprietà fisiche e chimiche rimangono invariate.

Ricapitolando per determinare il SER di ogni prodotto è necessario tener conto dei seguenti fenomeni:

1. Le impurità radioattive dei materiali utilizzati che emettono particelle alfa.
2. I raggi cosmici terrestri nella forma di neutroni ad alta energia che colpiscono gli atomi di silicio.
3. Le reazioni dell'isotopo ^{10}B indotte dai raggi cosmici a bassa energia.

1.4 Sensibilità ai soft error

Tra i vari circuiti elettronici integrati assumono particolare importanza le memorie e i circuiti logici sequenzali/combinatori. Viene qui di seguito presentata un'analisi della loro sensibilità ai soft errors.

1.4.1 Sensibilità delle memorie ai soft error

Attualmente il cuore di ogni sistema elettronico è un microprocessore con una grande memoria integrata, di solito DRAM, e interconnesso con varie periferiche. Per i sistemi più grandi può invece essere utilizzata una memoria discreta SRAM. Nonostante ora le memorie dinamiche siano tra i più robusti dispositivi elettronici è su queste che sono stati originariamente scoperti i soft error. Inizialmente infatti il loro SER (soft error rate) era molto alto dato che il segnale veniva salvato in una cella capacitiva che immagazzinava la carica in una giunzione dalla grande area bidimensionale, molto sensibile alle radiazioni. Con l'aumentare della densità di integrazione si è reso necessario rendere la cella tridimensionale aumentando Q_{crit} a parità di fattore di scala. Questa operazione ha permesso inoltre di ridurre l'efficienza della raccolta delle cariche.

Ci sono due fattori che influenzano il SER su un singolo bit al diminuire delle dimensioni:

1. L'efficienza della giunzione nel raccogliere le cariche si riduce al calare del volume della giunzione. Ciò si traduce in un decremento di Q_{coll}.
2. I voltaggi utilizzati diminuiscono e di conseguenza anche Q_{crit} viene ridotta.

La prima componente risulta predominante in quanto lo *scaling* del volume delle giunzioni decresce più in fretta di quano non facciano i voltaggi a cui operano le DRAM. Il SER sul bit quindi decresce mano a mano che la densità di integrazione aumenta mentre quello sull'intera memoria rimane pressochè costante.

Al contrario di quanto visto per le memorie dinamiche le SRAM erano inizialmente più sicure ma con l'aumentare della densità di integrazione il SER sul bit non è diminuito. Come per le DRAM i fattori che incidono sulla frequenza di soft error della SRAM sono dovuti alla decremento della Q_{col} e della Q_{crit}. I voltaggi usati però sono stati ridotti in maniera molto più aggressiva rispetto alle memorie dinamiche. Il risultato è che il SER sul bit, in un primo periodo, non solo non è calato ma è addirittura cresciuto. Eliminando il BPSG dai materiali utilizzati si è arrivati prima ad una saturazione dell'errore su bit

e successivamente, a tecnologie inferiori al micron, ad un decremento. In ogni caso il SER per l'intera memoria, che all'aumentare della densità di integrazione contiene un numero maggiore di bit, sta continuando ad aumentare senza sosta.

1.4.2 Sensibilità dei circuiti logici sequenziali/combinatori ai soft error

anche i circuiti logici combinatori e sequenziali possono essere affetti dai soft errors. I circuiti logici includono, tra i propri elementi, latch e flip-flop che sono molto simili alla cella di una memoria. Tuttavia risultano più robusti perchè ad ogni nodo sono collegati più transistor, inoltre questi dispositivi sono progettati per lavorare con correnti più alte che possono compensare facilmente alle cariche spurie dovute alle radiazioni. Bisogna inoltre tener conti del tempo in cui i circuiti sequenziali sono vulnerabili, ovvero quando stanno effettuando operazioni critiche: un dispositivo in semplice attesa non è influenzato dai soft errors.

1.5 Impatto sulla sensibilità dei prodotti

Dopo aver analizzato come i soft errors influenzano le memorie e i circuiti sequenziali/combinatori vediamo il loro impatto sull'affidabilità dei prodotti e come sia possibile, ove necessario, limitarli.

Un processore con una memoria incorporata può facilmente superare i 50.000 FIT che equivalgono a circa un soft error ogni due anni. In un dispositivo con un singolo chip non è un problema, considerando che può essere spento o che il soft error potrebbe colpire un bit non critico, e non è quindi necessario implementare tecniche di correzione d'errore.

In caso di sisitemi avanzati, ad esempio un mainframe, le cose si complicano. I 50.000 FIT devono essere moltiplicati per il numero di chip e la frequenza di soft error risultante può passare facilmente da uno ogni due anni ad uno a settimana. Risulta quindi obbligatorio mitigare le probabilità d'errore.

Non bisogna tuttavia sovrastimare la probabilità di errore. Ecco alcuni fatti che devono essere presi in considerazione:

- **Sensibilità dei dati**: non tutti i bit hanno la stessa importanza. Ad esempio in una memoria potrebbero esserci dei dati già letti. Il soft error potrebbe variare il valore di questi dati senza alterare il funzionamento del sistema. Infatti i dati corrotti potrebbero essere sovrascritti senza essere letti una seconda volta.

- **Sensibilità temporale**: in un circuito sincrono le radiazioni potrebbero indurre all'accumulo di carica in una giunzione poco prima della commutazione del clock. In questo caso il nodo viene pilotato dai circuiti esterni e non avviene un soft error.

- **Mascheramento logico**: in un circuito logico se l'errore avviene in un degli ingressi non c'è necessariamente un errore in uscita. Ad esempio un errore in un ingresso di una porta logica OR in cui l'altro input è a livello logico basso non ha alcun effetto sull'output.

Nel caso in cui il SER di un prodotto si riveli troppo elevato ci sono varie strategie utilizzabili. Come già visto è possibile limitare i soft error dovuti ai raggi cosmici a bassa energia eliminando il BPSG dal processo di costruzione. Per quanto riguarda le particelle alfa è possibile ridurre le impurità dei materiali utilizzati per il package o tenere queste distanti dalle giunzioni critiche del circuito. Modificando la truttura e il doping del substrato è possibile limitare la profondità alla quale i portatori vengono raccolti, diminuendo di conseguenza la Q_{crit}. Cambiando il design e i layout è possibile ridurre la Q_{crit} mantenendo invariata la Q_{coll}. Nei circuiti sequenziali/combinatori è accettabile memorizzare i dati critici in più nodi separati fisicamente, in modo che un unico evento radiattivo non influisca su tutti contemporaneamente. Purtroppo attuare quest'ultima tecnica sulle memorie risulta molto costoso e inefficiente dal punto di vista di area e, in maniera minore, di velocità e consumo.

Con le memorie è tuttavia possibile utilizzare dei circuiti o degli algoritmi di rilevazione e correzione d'errore. Il più semplice metodo di rilevazione consiste nell'aggiungere un bit di parità per ogni parola di dati rendendo il numero di 1 della nuova parola pari (in questo caso si parla di parità-pari) o dispari (si parla di parità-dispari). Durante la lettura del dato è quindi possibile controllare se questa situazione è variata, rilevando in questo caso un errore. Risulta però chiaro che con questo sistema non è possibile rilevare una variazione di un numero pari di bit e che il dato non può essere corretto automaticamente. Tecniche avanzate di codifica permettono, aumentando la ridondanza, la rilevazione e la correzione automatica di più bit.

Capitolo 2

Descrizione SRAM e bus FSL

Il progetto prevede lo sviluppo di una periferica che permette ad una CPU di interfacciarsi con la memoria statica CMOS ISSI® IS61LV5128AL. La periferica dovrà dialogare sia con la SRAM sia con il processore. Nel primo caso il dialogo avviene direttamente ed è quindi necessaria un'analisi degli ingressi e delle uscite della memoria, per comprendere il suo funzionamento e di conseguenza come strutturare la periferica. Il dialogo con la CPU invece non avviene direttamente: viene utilizzata una struttura intermedia sviluppata da Xilinx®che permette al processore e alla periferica di lavorare a frequenze differenti. Non è quindi necessario analizzare il funzionamento e gli I/O della CPU ma si deve invece procedere ad un'analisi del dispositivo intermedio, il LogiCORE^{TM} IP FSL v20 di Xilinx®.

2.1 SRAM

In questa prima sezione verrà fornita un'analisi del funzionamento della SRAM ISSI® IS61LV5128AL e delle tipologie di I/O che essa presenta. La funzione della SRAM è, ovviamente, quella di memorizzare dei dati di nostro interesse a cui possiamo successivamente accedere. Queste due funzioni sono possibili seguendo il protocollo definito nel datasheet della memoria e prendono il nome di ciclo Write, per la scrittura del dato, e ciclo Read, per la lettura. La ISSI® IS61LV5128AL ha una capacità di 512KB ed è struttrata nel seguente modo:

- i registi che contengono i valori memorizzati sono di 1 Byte (8 bit).
- ogni indirizzo è identificato univocamente da un indirizzo a 19 bit.

Nella figura 2.1 , tratta dal datasheet della memoria, troviamo tutti gli inputs e outputs della SRAM. Vdd e GND non sono da considerare degli ingressi veri e propri in quanto servono semplicemente per alimentare la memoria. I rimanenti ingressi/uscite sono invece suddivisi in tre tipologie: Enable Inputs, Address Inputs e Bidirectional Ports.

Enable Inputs Gli Enable Inputs sono i tre ingressi collegati al Control Circuit e permettono di controllare le funzioni principali della SRAM. Come si può notare dalla figura i tre ingressi funzionano in logica negata, cioè se viene

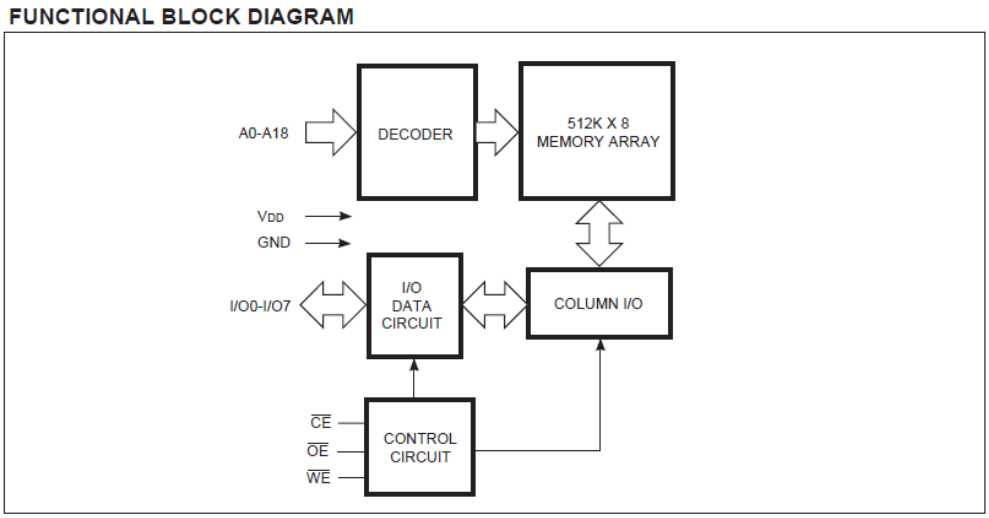

Figura 2.1: Schema Funzionale della SRAM.

presentato un livello logico basso la funzione è abilitata altrimenti, se il livello logico è alto, la funzione è disabilitata. Per questo motivo nel codice e nel resto di questa tesi si fa riferimento a questi ingressi come a nCE, nOE e nWE.

- **nCE**: quando nCE è a livello logico alto la SRAM è in stantby e il suo consumo viene ridotto a circa 250 μW. Se il livello logico è basso la SRAM è operativa.
- **nOE**: permette di controllare la direzione del bus I/O. Quando nOE è a livello logico basso il bus I/O è da considerare un bus di output mentre a livello logico alto diventa un bus di input.
- **nWE**: portando a livello logico basso questo pin si abilita la modalità scrittura. Si rende quindi necessaria l'abilitazione di questa funzione durante un ciclo Write.

Address Inputs L'Address Inputs è un bus a 19 bit e comprende gli ingressi da A0 fino ad A18. Con questo bus possiamo specificare l'indirizzo del registro su cui vogliamo operare. Questa funzione è indispensabile sia nei cicli di scrittura, sia nei cicli di lettura che verranno spiegati nel dettaglio successivamente.

Bidirectional Ports Il bus I/O è un bus a 8 bit bidirezionale che può quindi presentarsi sia come input, sia come output. La direzione viene decisa dall' Enable Input nOE come precedentemente descritto. Con questo bus possiamo presentare il valore che vogliamo scrivere in un registro di memoria durante un ciclo Write oppure leggere il valore memorizzato durante un ciclo Read.

2.2 Bus FSL

La comunicazione tra Periferica e CPU, al contrario di quella con la SRAM, non avviene direttamente. La CPU normalmente lavora a frequenze molto più eleva-

te rispetto a quelle della memoria, rendendo necessario l'utilizzo di un canale di comunicazione che consenta di collegare due dispositivi che lavorano in maniera asincrona. Il LogiCORE^{TM} IP FSL v20 di Xilinx® soddisfa tutte le richieste. É infatti un bus unidirezionale punto-punto che permette la trasmissione di dati da un dispositivo, detto Master, ad un altro, detto Slave. I due dispositivi devono essere due elementi progettati nell'FPGA. É possibile descrivere il bus come una coda FIFO (First In First Out) prevista di vari ingressi e uscite presentati in figura 2.2.

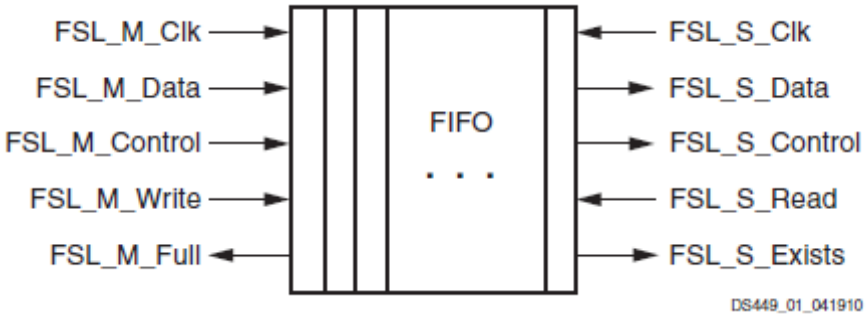

Figura 2.2: I/O di una singola struttura FIFO.

Ecco le funzioni di tutti gli I/O presenti, ad eccezione di FSL_M_Control e FSL_S_Control che non verranno utilizzati in questo progetto:

- **FSL_M_Clk**: questa porta fornisce il clock all'interfaccia Master del bus FSL quando la struttura FIFO è utilizzata in modalità asincrona.
- **FSL_M_Data**: l'ingresso dati dell'interfaccia Master del bus FSL.
- **FSL_M_Write**: segnale di input che controlla il segnale di abilitazione della scrittura dell'interfaccia Master della FIFO. Quando è ad un livello logico alto il valore presente su FSL_M_Data viene inserito nella struttura FIFO appena si ha un fronte di salita su FSL_M_Clk.
- **FSL_M_Full**: segnale di uscita nell'interfaccia Master che indica che la FIFO è piena.
- **FSL_S_Clk**: questa porta fornisce il clock alla interfaccia Slave del bus FSL quando si utilizza la struttura FIFO in modalità asincrona.
- **FSL_S_Data**: il bus di uscita dei dati nell'interfaccia Slave del bus FSL.
- **FSL_S_Read**: segnale di ingresso dell'interfaccia Slave che controlla il segnale di *acknowledge* della FIFO. Quando viene portato a valore logico alto, il valore di FSL_S_Data viene tolto dalla FIFO appena si ha un fronte di salita su FSL_S_Clk.
- **FSL_S_Exists**: segnale di uscita dell'interfaccia Slave che indica se la FIFO contiene dati validi.

- **FSL_Rst**: segnale di output generato dalla logica di reset dell'FSL. Ogni dispositivo collegato al bus FSL può utilizzare questo sengale come segnale di reset.
- **FSL_Full**: segnale di uscita che indica che la FIFO è piena e non può accettare altri dati.

Dato che il progetto richiede una comunicazione bidirezionale tra CPU e perifetica sono necessarie due strutture FIFO. Nella prima la CPU è il Master e la Periferica lo Slave, viene utilizzata per trasportare i comandi da eseguire, nella seconda il ruolo dei due dispositivi è invertito e la sua funzione è quella di trasmettere gli eventuali risultati delle operazioni eseguite sulla memoria. Gli ingressi e le uscite delle due strutture FIFO di nostro interesse sono esclusivamente quelle da collegare alla Periferica.

Capitolo 3

La macchina a stati

Si è deciso di sviluppare il codice VHDL utilizzando il modello della macchina a stati finiti o FSM (finite state machine). La periferica utilizza come base dei tempi il clock e ad ogni fronte di salita valuta, in base allo stato attuale e agli ingressi, quale sarà lo stato futuro. É quindi necessario definire per ogni dato che si intende salvare una versione attuale e una futura. Nel codice tutti i dati futuri hanno il nome che comincia per **new_** mentre quelli attuali iniziano per **new_**, ad eccezione dello stato che verrà chiamato semplicemente **state**. Bisogna prestare particolare attenzione al fatto che tutte le uscite modificate da un certo stato verranno in realtà aggiornate durante l'esecuzione dello stato successivo. In questo modo è possibile ottenere la sincronizzazione di tutti i segnali. A livello di codice la macchina a stati prevede lo sviluppo di due processi:

1. Il primo processo è sensibile alle variazioni del clock, più precisamente al fronte di salita. La sua funzione è quella di aggiornare la macchina a stati: tutti i dati futuri diventano quindi i dati attuali, compreso lo stato.

2. Il secondo processo è sensibile alla variazione di qualunque ingresso e di qualunque stato attuale. La sua funzione è quella di definire il comportamento di **tutti** gli stati della macchina. A livello di codice questo si traduce in una serie di costrutti if-then ognuno dei quali descrive uno stato.

3.1 Lo stato idle

Seguendo il modello della macchina a stati è necessario definire uno stato iniziale: lo stato idle. Questo stato ha una funzione molto semplice e la sua analisi è utile per capire il funzionamento della MSF e la sua interazione con i dispositivi collegati alla periferica. In figura 3.1 viene riportato il codice che realizza lo stato idle. Inizialmente la periferica rimane in attesa che la CPU le comunichi qualcosa attraverso il bus FSL CPU->Periferica. Si ricorda che se nella struttura FIFO è presente almeno un comando FSL_S_Exists è a livello logico alto e in queste condizioni la periferica legge la testa della coda, ovvero FSL_S_Data. Durante lo stato idle si aspetta che la CPU comunichi quale comando vuol far eseguire alla periferica. Sono stati arbitrariamente associati i valori 0, 1, 2, 3, 4

```
185 if state = idle then
186    if FSL_S_Exists = '1' then
187       case ( FSL_S_Data(28 to 31) ) is
188          when X"0" => new_state <= chip_enable;
189          when X"1" => new_state <= write_v;
190          when X"2" => new_state <= set_byte_v;
191          when X"3" => new_state <= errors_p;
192          when X"4" => new_state <= check_p;
193          when X"5" => new_state <= chip_disable;
194          when others => new_state <= idle;
195       end case;
196       FSL_S_Read <= '1';
197    end if;
198 end if;
```

Figura 3.1: Codice VHDL dello stato idle

e 5 rispettivamente ai comandi chip enable, write, set byte, errors, check e chip disable. Il primo e l'ultimo comando permettono di abilitare e disabilitare la memoria, write e set byte sono dei comandi di scrittura e errors e check sono dei comandi di lettura. Una volta che il comando viene identificato attraverso il costrutto **case** la FSM provvederà a impostare correttamente lo stato futuro **new_state**. Infine viene portato a livello logico alto il pin FSL_S_Read per permettere la rimozione del comando dalla testa della coda da parte della struttura FIFO CPU->Periferica.

3.2 Esecuzione del comando chip enable

```
204 if state = chip_enable then
205    new_nCE <= '0';
206    new_state <= idle;
207 end if;
208
209
210 if state = chip_disable then
211    new_nCE <= '1';
212    new_state <= idle;
213 end if;
```

Figura 3.2: Codice VHDL dello stato chip_enable

Dopo aver visto lo stato iniziale vediamo come viene eseguito il più semplice comando: chip enable. La sua funzione è quella di rendere operativa la memoria

portando a livello logico basso il pin nCE, come descritto nella sezione relativa agli ingressi/uscite della SRAM. Lo stato che realizza il comando chip enable è riportato in figura 3.2 ed è autoesplicativo. In figura 3.3 è riportata una

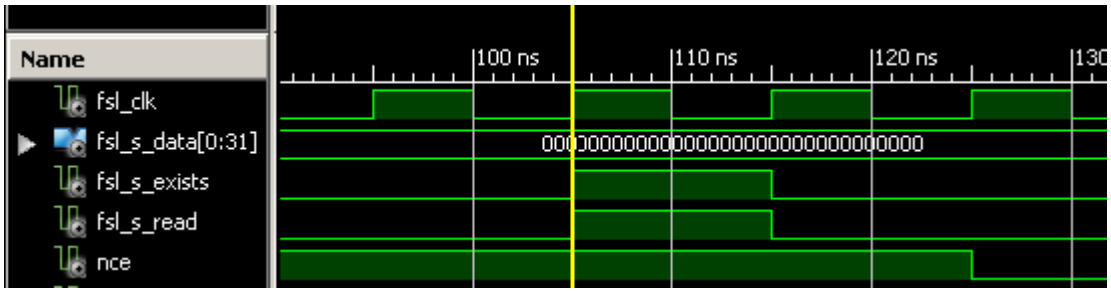

Figura 3.3: Codice VHDL dello stato chip_enable

simulazione, effettuata tramite lo strumento ISim fornito nel pacchetto Xilinx ISE Design Suite di Xilinx®, della ricezione del comando chip enable e della sua esecuzione.

- Nel primo periodo la macchina a stati è in idle e sul bus non è presentato alcun dato.
- Nel secondo periodo viene presentato nel bus FSL_S_Data la parola con tutti i 32 bit a '0'. FSL_S_Exists va a livello logico alto per avvertire la periferica che è presente un comando da eseguire, in questo caso Chip Enable. La FSM assegna quindi allo stato futuro il valore chip_enable grazie al costrutto case e manda FSL_S_Read a livello logico alto per comunicare alla struttura FIFO la rimozione del comando.
- Nel terzo periodo la macchina a stati provvede a portare a livello logico basso new_nCE.
- Nell'ultimo periodo la FSM torna nello stato di idle in attesa di un nuovo comando. Inoltre quello che nel periodo precedente era lo stato futuro di nCE è diventato lo stato attuale: il pin nCE viene quindi portato a livello logico basso rendendo la SRAM operativa.
  Il comando chip disable funziona in maniera analoga ma porta il pn nCe a livello logico alto.

3.3 Lo stato di fine istruzione

Per tutti i comandi che verranno descritti successivamente, ad eccezione quindi dei comandi Chip Enable e Chip Disable, è prevista una comunicazione della Periferica alla CPU per avvertirla che il comando è stato eseguito. Si è deciso di utilizzare la parola con tutti i 32 bit a '1' per svogere questa funzione. A livello di codice, presentato in figura 3.4, è previsto uno stato che, nel caso in cui la coda Periferica->CPU non sia piena, inserisca la parola scelta nel bus FSL_M_Data. Successivamente la FSM passa allo stato idle per poter eseguire

altri comandi. É comunque necessario che la parola di fine istruzione non possa essere confusa con altri dati inviati dalla periferica alla CPU in modo che non ci siano ambiguità.

```
469  if state = end_inst then
470     if FSL_M_Full = '1' then
471        new_state <= end_inst;
472     else
473        new_FSL_M_Data <= end_instruction;
474        new_FSL_M_Write <= '1';
475        new_state <= idle;
476     end if;
477  end if;
```

Figura 3.4: Codice VHDL dello stato end_inst

Capitolo 4

Comandi di scrittura

In questo capitolo verranno descritti i comandi di scrittura. Per l'esperimento abbiamo bisogno di un comando che permetta di scrivere un valore su tutti i registri della memoria. Questa operazione prepara la SRAM all'esposizione alle radiazioni. Successivamente si effettuerà, una scansione della memoria per rilevare eventuali errori. In fase di collaudo è però utile avere a disposizione un comando che simuli un errore su uno specifico registro per controllare il corretto funzionamento dei comandi di lettura. Il comando write, che si occuperà di scrivere lo stesso valore su tutti i registri viene eseguito un'unica volta, successivamente viene utilizzato per più volte il comando Set Byte per simulare vari errori all'interno della memoria. Il primo comando analizzato sarà set byte perchè è più semplice e permette di comprendere come la periferica riesce a soddisfare il protocollo richiesto per l'operazione di scrittura sulla SRAM. Successivamente verrà analizzato il comando write che prevede tanti cicli di scrittura quanti sono i registri disponibili.

4.1 Set Byte

Il comando Set Byte richiede che vengano specificati un valore e l'indirizzo del registro su cui si vuole memorizzarlo. Sono necessarie quindi 3 parole di 32 bit per l'esecuzione di questo comando:

1. La prima parola è il codice del comando, grazie a questa la periferica sa qual è l'operazione che la CPU richiede. Il codice previsto per il comando Set Byte è 2 quindi la parola a 32 bit in esadecimale è X0000_0002.

2. La seconda parola deve contenere il valore che si vuole memorizzare. Dato che i registri della SRAM sono da 1Byte sono necessari solamente 8 dei 32 bit della parola. Si è scelto di utilizzare gli 8 bit meno significativi, i restanti bit non saranno presi in considerazione.

3. L'ultima parola deve contenere l'indirizzo del registro su cui si vuole salvare il valore. Si è scelto di utilizzare i 19 bit meno significativi della parola per contenerlo. I rimanenti bit non saranno presi in considerazione.

Gli stati che permettono la realizzazione del comando sono suddivisi in tre gruppi: lo stato set_byte_v si occupa di ricevere correttamente il valore da

memorizzare; lo stato nella forma set_byte_a si occupa di ricevere l'indirizzo; infine gli stati nella forma set_byte_#**stato** si occupano di interagire con la memoria e di rispettare il protocollo del ciclo Write descritto in figura 4.1 presa direttamente dal datasheet della SRAM. Vediamo ora nel dettaglio i singoli stati

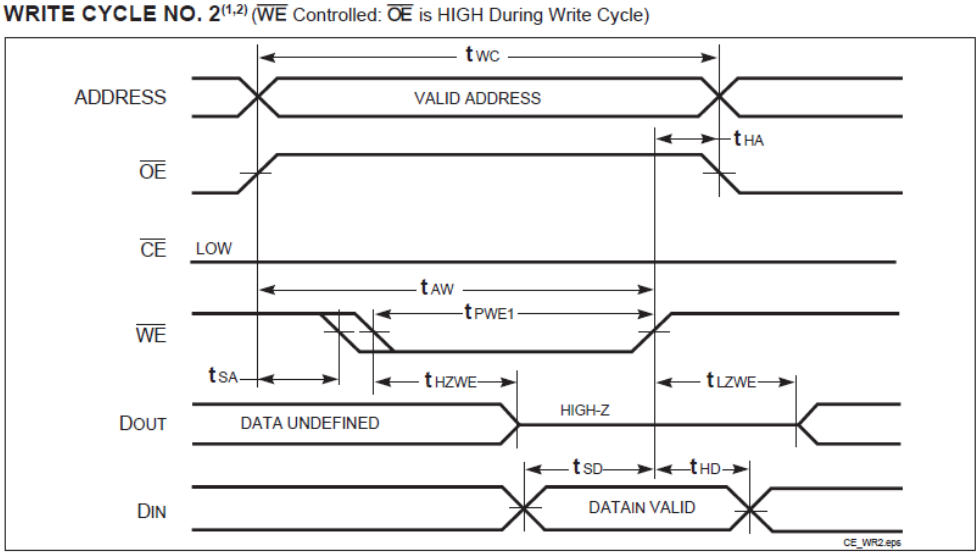

Figura 4.1: Protocollo del ciclo di scrittura della SRAM

e la loro funzione.

set_byte_v Questo stato si occupa della ricezione e memorizzazione del valore da scrivere nella SRAM. La periferica controlla innanzitutto che il bit FSL_S_Exists sia a livello logico alto. Nel caso in cui non lo sia vuol dire che la CPU deve ancora inserire il valore nella struttura FIFO CPU->Periferica e di conseguenza la FSM rimane in attesa impostando set_byte_v come stato futuro. Se invece FSL_S_Exists è a '1' la Periferica memorizza gli 8 bit meno significativi di FSL_S_Data come valore da scrivere. Viene inoltre portato FSL_S_Read a '1' per comunicare alla struttura FIFO CPU->Periferica che il dato presente sulla testa della coda può essere eliminato. Infine viene impostato come stato futuro set_byte_a.

set_byte_a Questo stato si occupa di ricevere e memorizzare l'indirizzo su cui si vuole eseguire il ciclo di scrittura. Il procedimento è lo stesso dello stato precedente con la differenza che vengono salvati i 19 bit meno significativi di FSL_S_Data. La Periferica ha ora tutti i parametri per poter iniziare il ciclo di scrittura e imposta il valore dello stato futuro a set_byte_0.

```
277 if state = set_byte_v then
278    if FSL_S_Exists = '1' then
279       new_value <= FSL_S_Data(24 to 31);
280       FSL_S_Read <= '1';
281       new_state <= set_byte_a;
282    else
283       new_state <= set_byte_v;
284    end if;
285 end if;
286
287
288 if state = set_byte_a then
289    if FSL_S_Exists = '1' then
290       new_address <= FSL_S_Data(13 to 31);
291       FSL_S_Read <= '1';
292       new_state <= set_byte_0;
293    else
294       new_state <= set_byte_a;
295    end if;
296 end if;
297
298
299 if state = set_byte_0 then
300       new_A <= s_address;
301       new_DATA_T <= '0';
302       new_nOE <= '1';
303       new_nWE <= '1';
304       new_state <= set_byte_1;
305 end if;
306
307
308 if state = set_byte_1 then
309       new_nWE <= '0';
310       new_state <= set_byte_2;
311 end if;
312
313
314 if state = set_byte_2 then
315       new_O <= s_value;
316       new_state <= set_byte_3;
317 end if;
```

Figura 4.2: Codice VHDL del comando Set Byte.

```
320  if state = set_byte_3 then
321        new_nWE <= '1';
322        new_state <= set_byte_4;
323  end if;
324
325
326  if state = set_byte_4 then
327        new_DATA_T <= '1';
328        new_nOE <= '0';
329        new_state <= end_inst;
330  end if;
```

Figura 4.3: Codice VHDL del comando Set Byte (continua).

set_byte_0 Seguendo il protocollo di scrittura riportato nella figura 4.1 la periferica si assicura che i due pin di enable nOE e nWE siano a livello logico alto, ovvero disabilitati. In questo modo il bus I/O è impostato nella direzione Periferica->SRAM mentre la modalità scrittura è attualmente disabilitata. Il pin nOE indica alla SRAM qual è la direzione del bus I/O mentre nella Periferica questa funzione viene svolta da DATA_T. Nel resto di questo capitolo e del successivo verrà quindi preso in considerazione solamente il pin nOE dando per scontato che ad una sua variazione è associata anche la variazione su DATA_T Viene inoltre impostato il valore dell'indirizzo su cui vogliamo scrivere su new_A in modo che venga presentato nel corrispondente bus durante l'esecuzione del prossimo stato.

set_byte_1 Il pin nWE viene abilitato per preparare la SRAM alla scrittura del valore nell'opportuno registro identificato dall'indirizzo presentato sul bus A. Viene impostato set_byte_2 come stato futuro.

set_byte_2 Questo stato si occupa di presentare il valore da memorizzare sul bus I/O. A livello di circuito si attende ora che il valore presentato sul bus si stabilizzi per poter essere letto correttamente.

set_byte_3 Ora che il valore si è stabilizzato la Periferica comanda alla SRAM di memorizzarlo nel registro di memoria portando il pin nWE a livello logico alto.

set_byte_4 L'ultima operazione da eseguire per completare il protocollo riportato in figura 4.1 è quello di portare il pin nOE a livello logico basso. Questo imposta il bus I/O nella direzione SRAM->Periferica. Infine viene assegnato end_inst allo stato futuro per comunicare alla CPU che il comando è stato eseguito.

4.2 Write

Il comando write permette di scrivere lo stesso valore su tutti i registri di memoria. Per la sua esecuzione sono quindi necessarie due parole a 32 bit.

1. La prima parola a 32 bit richiesta è il codice del comando. É stato deciso di assegnare al comando write il numero 2 e quindi la parola esadecimale risultante è X0000_0002.

2. La seconda parola a 32 bit è il valore da memorizzare nei registri della SRAM. Essendo i registri da 1 byte si è scelto di utilizzare gli 8 bit meno significativi.

Una volta memorizzato il valore la Periferica provvederà ad eseguire tanti cicli di scrittura quanti sono i registri della memoria, in questo caso 2^{19}. Questa sembra la situazione ideale per usare il costrutto for, che purtroppo non può essere implementato direttamente a causa della struttura della macchina a stati. Si procede quindi ad una simulazione di tale costrutto utilizzando gli stati per inizializzare il ciclo for e controllare la condizione finale. La variabile utilizzata come indice del ciclo è l'indirizzo. Si è scelto di utilizzare come registro iniziale quello identificato dall'indirizzo con i bit tutti a 0. L'indirizzo viene incrementato ad ogni ciclo finchè non si raggiunge il valore con tutti i bit a 1, dopodiché la macchina a stati procederà ad inviare alla CPU il messaggio di fine istruzione e, successivamente, tornerà nello stato idle.

Gli stati da write_1 a write_4 eseguono il ciclo di scrittura previsto dal protocollo di figura 4.1 come già è stato descritto nella descrizione del comando Set Byte. Vediamo invece una descrizione degli altri stati.

write_v Questo stato si preoccupa di ricevere il valore da memorizzare nei vari registri della SRAM. Quando FSL_S_Existis è a livello logico alto è presente un dato nella struttura FIFO CPU-Periferica: vengono quindi memorizzati gli 8 bit meno significativi e FSL_S_Read viene portato a '1' per eliminare il dato dalla FIFO. Se il valore non è ancora presente nella struttura FIFO la Periferica rimane in attesa rimanendo nello stato write_v.

write_0 Il ciclo for viene inizializzato. Tutti i bit della variabile, che contiene l'indirizzo su cui si vuole operare, vengono portati a '0'.

write_5 Ponendo il pin nOE a livello logico basso si pone fine al ciclo di scrittura descritto dal protocollo. Successivamente si controlla l'indirizzo del registro su cui si è appena memorizzato il dato per vedere se bisogna procedere con un'altra iterazione o se il ciclo for è terminato. Nel caso in cui l'indirizzo sia identico alla costante last_address il comando write è stato eseguito e la FSM passa allo stato end_inst per avvisare la CPU. Se invece è necessaria almeno un'altra iterazione l'indirizzo futuro viene posto pari a quello attuale più una unità. Assegnando a new_state il valore write_1 la macchina a stati procede con un altro ciclo di scrittura.

```
218 if state = write_v then
219    if FSL_S_Exists = '1' then
220       new_value <= FSL_S_Data(24 to 31);
221       FSL_S_Read <= '1';
222       new_state <= write_0;
223    else
224       new_state <= write_v;
225    end if;
226 end if;
227
228
229 if state = write_0 then
230    new_address <= (others => '0');
231    new_state <= write_1;
232 end if;
233
234
235 if state = write_1 then
236    new_A <= s_address;
237    new_DATA_T <= '0';
238    new_nOE <= '1';
239    new_nWE <= '1';
240    new_state <= write_2;
241 end if;
242
243
244 if state = write_2 then
245    new_nWE <= '0';
246    new_state <= write_3;
247 end if;
248
249
250 if state = write_3 then
251    new_O <= s_value;
252    new_state <= write_4;
253 end if;
```

Figura 4.4: Codice VHDL del comando Write.

```
256  if state = write_4 then
257     new_nWE <= '1';
258     new_state <= write_5;
259  end if;
260
261
262  if state = write_5 then
263     new_DATA_T <= '1';
264     new_nOE <= '0';
265     if ( s_address /= last_address) then
266           new_address <= s_address +1;
267           new_state <= write_1;
268        else
269           new_state <= end_inst;
270     end if;
271  end if;
```

Figura 4.5: Codice VHDL del comando Write (continua).

Capitolo 5

Comandi di lettura

Come abbiamo visto nel capitolo precedente grazie al comando di scrittura write è possibile scrivere su tutti i registri di memoria lo stesso valore. La SRAM viene quindi sottoposta alle radiazioni che potrebbero indurre dei soft errors. É quindi ora necessario implementare dei comandi che rendano possibile la lettura della memoria e che rilevino eventuali errori presenti in essa.
I comandi realizzati sono Errors e Check e hanno le seguenti funzioni

- **Errors**: la sua funzione è quella di rilevare il **numero** di errori. Dopo aver selezionato il comando si invia alla Periferica un valore a 8bit che verrà utilizzato come pattern nella scansione della memoria. Verranno quindi letti tutti i registri della SRAM e il loro valore verrà confrontato con il pattern: in caso i due valori siano differenti si parlerà di *mismatch*. Alla fine della scansione viene inviato alla CPU, attraverso l'apposita struttura FIFO, il numero di *mismatch* riscontrati nella scansione.

- **Check**: questo comando ha la funzione di rilevare in quali registri della memoria è avvenuto un errore e qual è il valore attualmente salvato. Dopo aver selezionato il comando è quindi necessario inviare alla Periferica anche un valore che, anche in questo caso, viene utilizzato come pattern. La Periferica attuerà una scansione e appena troverà un *mismatch* tra il pattern e un valore invierà prima quest'ultimo alla CPU e subito dopo l'indirizzo del registro che lo contiene.

Chiaramente per ottenere dei risultati coerenti è necessario che il valore inviato alla periferica come parametro per i due comandi sia lo stesso utilizzato precedentemente nella fase di scrittura.

5.1 Errors

Il comando Errors prevede che vengano inviate due parole a 32 bit:

1. la prima è la parola che specifica la scelta del comando Errors. É stato deciso di assegnargli il numero 3 e quindi la parola a 32 bit scritta in esadecimale diventa X0000_0003.

2. la seconda parola deve contenere il valore a 8 bit che verrà utilizzato come pattern nella scansione dei registri della SRAM. Si è scelto di utilizzare gli 8 bit meno significativi della parola a 32 bit.

Una volta ottenuto il pattern attraverso la struttura FIFO la Periferica procede alla scansione partendo dal registro identificato dall'indirizzo con tutti i bit a '0'. La FMS a questo punto simula il costrutto for incrementando dopo ogni lettura l'indirizzo di una unità, fino ad arrivare a quello composto da tutti i bit a '1'.
In figura 5.1 vediamo il protocollo del ciclo di lettura preso direttamente dal datasheet della memoria ISSI® IS61LV5128AL.

AC WAVEFORMS
READ CYCLE NO. 1[1,2] (Address Controlled) ($\overline{CE}$ = $\overline{OE}$ = V_{IL})

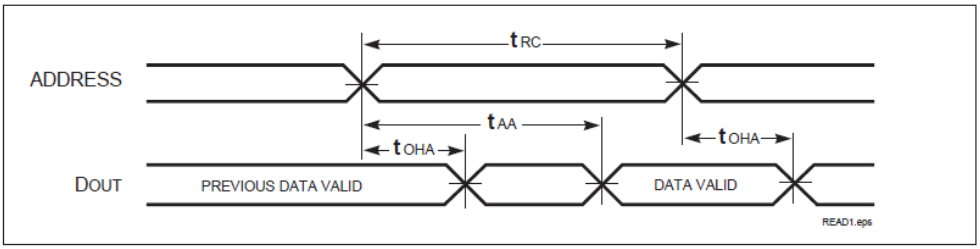

Figura 5.1: Protocollo del ciclo di lettura della SRAM

Vediamo ora come la FMS permette l'esecuzione del comando Errors rispettando il protocollo del ciclo di lettura analizzando i vari stati.

errors_p Questo stato si occupa di ricevere il valore ad 8 bit che verrà utilizzato come pattern. Come già visto la Periferica attende che FSL_S_Exists sia a livello logico alto per leggere il dato presentato su FSL_S_Data, memorizzarlo e, attraverso FSL_S_Read, rimuovere la testa della coda alla struttura FIFO CPU->Periferica. Finchè il dato non è presente nella coda la FSM rimane nello stato errors_p, mentre dopo la lettura procede verso lo stato errors_0.

errors_0 Per prima cosa viene portato il pin di controllo nOE a livello logico basso per rispettare il protocollo di lettura. In questo modo la direzione del bus I/O è SRAM->Periferica.
Inoltre viene inizializzato il ciclo for eseguendo le seguenti operazioni:

- i **signal** s_errors e new_errors servono come contatori per il numero di errori riscontrati (il primo è il valore dello stato attuale mentre il secondo quello che verrà preso in considerazione nello stato futuro). Si inizializza quindi il numero di errori a 0.
- viene assegnato il primo indirizzo del registro su cui viene eseguito il ciclo di lettura.

```
336 if state = errors_p then
337    if FSL_S_Exists = '1' then
338       new_pattern <= FSL_S_Data(24 to 31);
339       FSL_S_Read <= '1';
340       new_state <= errors_0;
341    else
342       new_state <= errors_p;
343    end if;
344 end if;
345
346
347 if state = errors_0 then
348       new_DATA_T <= '1';
349       new_nOE <= '0';
350       new_address <= (others => '0');
351       new_state <= errors_1;
352       new_errors <= (others => '0');
353 end if;
354
355
356 if state = errors_1 then
357       new_A <= s_address;
358       new_state <= errors_2;
359 end if;
360
361
362 if state = errors_2 then
363       new_state <= errors_3;
364 end if;
365
366 if state = errors_3 then
367       if s_pattern /= DATA_I then
368          new_errors <= s_errors + 1;
369       end if;
370       if s_address = last_address then
371          new_state <= errors_4;
372       else
373          new_state <= errors_1;
374          new_address <= s_address+1;
375       end if;
376 end if;
```

Figura 5.2: Codice VHDL del comando Errors.

```
378 if state = errors_4 then
379     if FSL_M_Full = '1' then
380         new_state <= errors_4;
381     else
382         new_FSL_M_Data <= s_errors;
383         new_FSL_M_Write <= '1';
384         new_state <= end_inst;
385     end if;
386 end if;
```

Figura 5.3: Codice VHDL del comando Errors (continua).

errors_1 Il protocollo ora prevede che venga presentato sul bus A l'indirizzo del registro su cui vogliamo eseguire il ciclo di scrittura. Si ricorda che, per rispettare il modello della macchina a stati, il valore verrà presentato realmente sul bus A durante l'esecuzione dello stato successivo errors_2.

errors_2 Questo stato è un semplice stato di attesa durante la quale il valore dell'indirizzo viene presentato sul bus A. In questo modo è possibile rispettare il protocollo che prevede l'attesa del tempo t_{RC} prima di poter leggere il valore sul bus I/O.

errors_3 Il valore contenuto nel registro è ora disponibile sul bus I/O ed è quindi possibile confrontarlo con il pattern. In caso di *mismatch* il contatore degli errori viene incrementato di una unità. Dopodiché si controlla se l'indirizzo del registro appena controllato è l'ultimo oppure c'è bisogno di almeno un'altra iterazione del ciclo for. Nel primo caso si passa allo stato errors_4, nel secondo si incrementa l'indirizzo di una unità e si torna allo stato errors_1.

errors_4 Questo stato provvede a comunicare alla CPU il numero di errori rilevato nella scansione della memoria. Per farlo viene utilizzata la struttura FIFO Periferica->CPU. Per poter inserire il dato è necessario che la coda non sia piena e che quindi FSL_M_Full non sia a livello logico alto. Se la coda è piena la FSM rimane in attesa rimanendo nello stato errors_4. In caso contrario si provvede a presentare il dato su FSL_M_Data e a portare FSL_M_Write a livello logico alto.
Successivamente si passa allo stato end_inst per informare la CPU che l'esecuzione del comando è stata completata.

5.2 Check

Il comando Check richiede che venga specificato un valore che verrà utilizzato come pattern nella scansione della memoria.
Le parole a 32 bit ricevute dalla periferica sono quindi:

1. Si è deciso di assegnare al comando Check il numero 4. La parola 32 bit in esadecimale diventa dunque X0000_0004.

2. La seconda parola a 32 bit deve contenere il valore da 1byte che viene utilizzato come pattern. Si è deciso di utilizzare gli 8 bit meno significativi.

Anche in questo caso la FSM, dopo aver acquisito il valore da usare come pattern, deve simulare il costrutto for per eseguire la scansione di tutta la memoria. Come è possibile notare leggendo il codice riportato nelle figure 5.4 e 5.5 alcuni degli stati sono analoghi a stati del comando Errors.
Verrano qui di seguito analizzati gli stati che permettono alla Periferica di eseguire le specifiche richieste dal comando Check.

check_p Questo stato è analogo a errors_p e la sua funzione è quella di acquisire e memorizzare il parametro da usare come pattern.

check_0 Vengono impostate le condizioni iniziali del ciclo for. A differenza di quanto avviene nel comando Errors non viene inizializzato nessuno contatore degli errori.

check_1 Come nello stato errors_1 si impone che nello stato futuro sia presentato l'indirizzo sul bus A.

check_2 Questo stato permette che il valore dell'indirizzo venga presentato realmente sul bus A come nello stato errors_2.

check_3 Il valore del registro selezionato è ora presente sul bus I/O e viene confrontato con il pattern. Se i due valori sono uguali si controlla se il ciclo for è finito, e quindi si passa allo stato di fine istruzione, o se è necessaria almeno un'altra iterazione, incrementando l'indirizzo e tornando allo stato check_1. In caso di *mismatch* il valore presente nel registro viene salvato sul **signal** pattern_e e si imposta come stato futuro check_4.

check_4 Questo stato si occupa di inserire nella struttura FIFO Periferica->CPU il valore del registro in cui è avvenuto l'errore. Si è deciso di utilizzare gli 8 bit meno significativi per contenerlo. Gli altri bit vengono portati a '0' grazie alla costante v31downto19_0 per evitare che questa parola coincida con la parola di fine istruzione.

check_5 La FSM inserisce nella FIFO il valore dell'indirizzo del registro in cui è avvenuto l'errore utilizzando i 19 bit meno significativi della parola. Anche in questo caso i rimanenti bit vengono posti a '0', grazie alla costante v31downto19_0, per evitare coincidenze con la parola di fine istruzione. La FSM controlla infine se il ciclo for è finito o è necessaria almeno un'altra iterazione.

```
392  if state = check_p then
393     if FSL_S_Exists = '1' then
394        new_pattern <= FSL_S_Data (24 to 31);
395        FSL_S_Read <= '1';
396        new_state <=check_0;
397     else
398        new_state <= check_p;
399     end if;
400  end if;
401
402
403  if state = check_0 then
404        new_DATA_T <= '1';
405        new_nOE <= '0';
406        new_address <= (others => '0');
407        new_state <= check_1;
408  end if;
409
410  if state = check_1 then
411        new_A <= s_address;
412        new_state <= check_2;
413  end if;
414
415
416  if state = check_2 then
417        new_state <= check_3;
418  end if;
419
420
421  if state = check_3 then
422        if s_pattern = DATA_I then
423           if s_address = last_address then
424              new_state <= end_inst;
425           else
426              new_address <= s_address+1;
427              new_state <= check_1;
428           end if;
429        else
430           new_pattern_e <= DATA_I;
431           new_state <= check_4;
432        end if;
433  end if;
```

Figura 5.4: Codice VHDL del comando Check.

```
436 if state = check_4 then
437    if FSL_M_Full = '1' then
438       new_state <= check_4;
439    else
440       new_FSL_M_Data (31 downto 8) <= v31downto8_0;
441       new_FSL_M_Data (7 downto 0) <= s_pattern_e;
442       new_FSL_M_Write <= '1';
443       new_state <= check_5;
444    end if;
445 end if;
446
447
448 if state = check_5 then
449    if FSL_M_Full = '1' then
450       new_state <= check_5;
451    else
452       new_FSL_M_Data(31 downto 19) <= v31downto19_0;
453       new_FSL_M_Data(18 downto 0) <= s_address;
454       new_FSL_M_Write <= '1';
455       if s_address = last_address then
456          new_state <= end_inst;
457       else
458          new_address <= s_address+1;
459          new_state <= check_1;
460       end if;
461    end if;
462 end if;
```

Figura 5.5: Codice VHDL del comando Check (continua).

Conclusioni

La periferica per rilevare soft errors sulla SRAM è stata realizzata programmando in VHDL una FPGA, utilizzando il modello della macchina a stati che permette di rendere il circuito sincrono. I segnali di uscita della periferica cambiano infatti di livello solamente durante il fronte di salita del Clock. Oltre ai comandi di abilitazione e disabilitazione della memoria sono stati implementati dei comandi di scrittura, per preparare la SRAM all'esposizione alle radiazioni, e due comandi di lettura per rilevare eventuali errori. I comandi di lettura permettono di conoscere il numero di errori presenti nella memoria, il valore dei dati corrotti e l'indirizzo del registro in cui si trovano.

Bibliografia

[1] Robert C. Baumann, Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, IEEE TRANSACTION ON DEVICE AND MATERIALS RELIABILITY, VOL. 5, NO. 3, SEPTEMBER 2005 305, Fellow, IEEE

[2] IS61LV5128AL 512K x 8 HIGH-SPEED CMOS STATIC RAM, datasheet http://www.datasheetcatalog.org/datasheets2/10/101483_1.pdf

[3] LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c), datasheet, April 2010
http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf