****Politecnico di Torino**

*Corso di laurea in Ingegneria Elettronica*

**Relazione Laboratorio 6 Elettronica dei sistemi digitali**

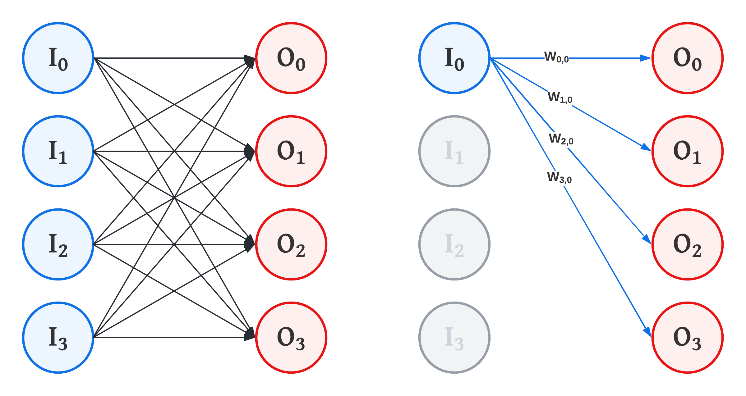
**Implementazione hardware di una rete neurale per classificare caratteri MNIST**

*M. Ghibaudo, A. Marchei, T. Terzano*

1. ***Introduzione***

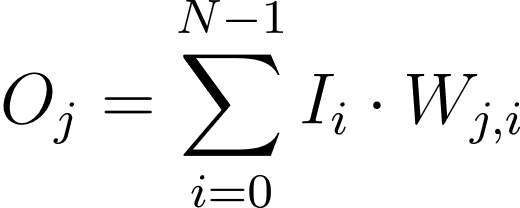
Lo scopo di questa esperienza è sviluppare una rete neurale, modello computazionale utilizzato per catalogare e analizzare dati con grande accuratezza. La rete neurale qui trattata è capace di classificare un’immagine 16 x 16, con pixel definiti a 30 bit, rappresentante una cifra da 0 a 9 scritta a mano, ricavata dal dataset MNIST[[1]](#footnote-1).

Il sistema digitale è composto da 256 neuroni di input, ovvero i pixel dell’immagine, collegati ai 10 neuroni di output attraverso delle sinapsi, ognuna con un peso differente. La realizzazione del modello capace di generare le sinapsi non è argomento di questa esperienze, il peso delle stesse viene quindi fornito in input al sistema.



**1.1:** *Rappresentazione schematica di una rete neurale. A destra, dettaglio sulle sinapsi pesate.*

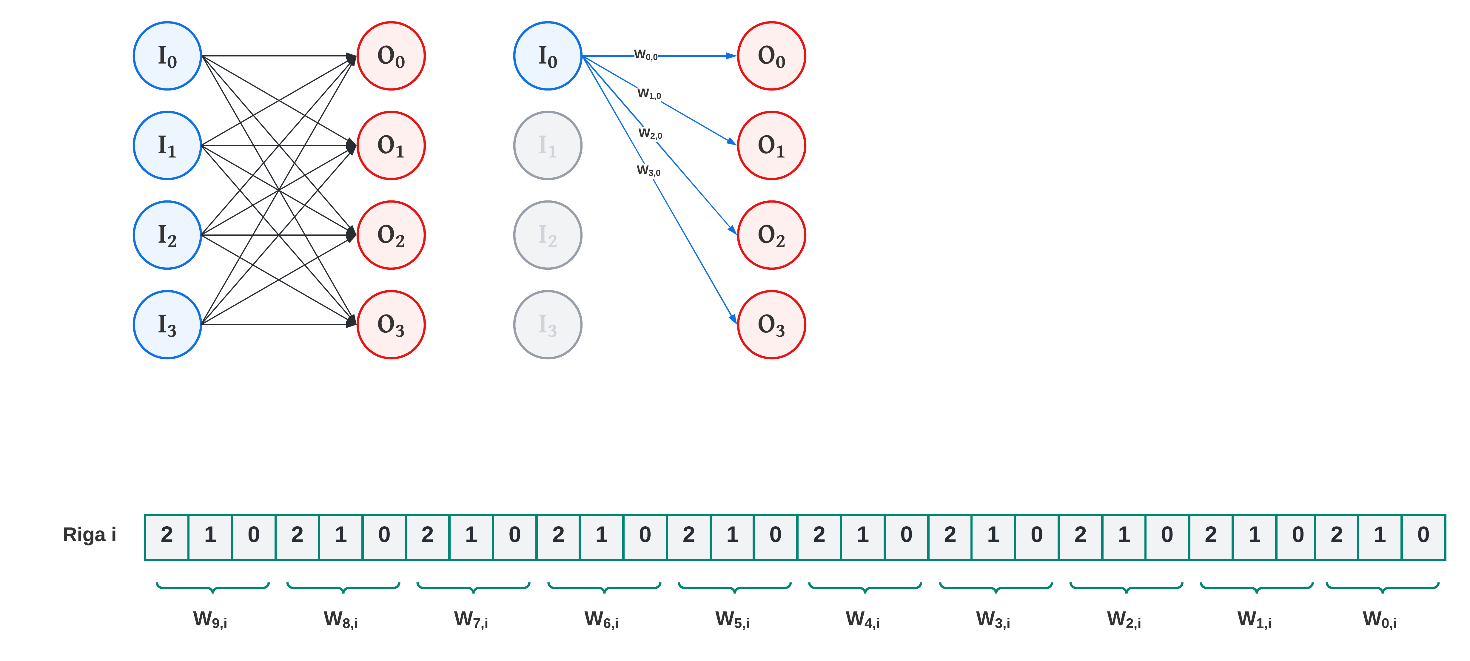
Il neurone di output con il valore più alto al termine della computazione rappresenta il risultato della classificazione compiuta dal sistema, ovvero la cifra da 0 a 9 più probabile. Si tratta di una rete neurale completamente connessa, in quanto ogni neurone di input è collegato ad ogni i neurone di output. L’algoritmo eseguito dal sistema è il seguente:



**1.2*:*** *Oj è l’output j-esimo, Ii è l’iput i-esimo e Wj,i è il peso della sinapsi*

1. ***Specifiche di progetto***
   1. ***Struttura di sistema***

Il circuito presenta un segnale di input, denominato START. Una volta che il sistema campiona START=1, al colpo di clock successivo viene caricato l’input DATA\_IN in una memoria RAM (MEM\_A) da 512 celle. E’ importante notare che i dati d’ingresso hanno parallelismo a 30bit e sono unsigned. Inoltre, vengono caricati prima i pixel fino all’indirizzo 255 e successivamente i pesi, fino all’indirizzo 511. Questi ultimi sono rappresentati con un signed a 3bit, per cui nella cella di memoria i-esima avremo 9 pesi, che costituiscono i pesi delle sinapsi fra il neurone i-esimo e i 10 neuroni di output.



**2.1*:*** *Struttura di una cella di memoria i-esima, con 255<i<512*

La memoria MEM\_A è dotata di segnali di enable (CS), di lettura (RD) e scrittura attivo basso (WR). Gli input sono DATA\_IN, ADDRESS e CLK, mentre l’output è DATA\_OUT. Le operazioni di scrittura e lettura sono sincrone ed avvengono con il rising edge del clock.

Per quanto riguarda le operazioni aritmetiche da compiere, il sistema utilizza un unico solo adder. Maggiori informazioni riguardo alle tecnologie scelte per l’adder e il parallelismo sono riportate di seguito. Una volta eseguite le operazioni, se necessario, il risultato viene saturato negativamente o positivamente al massimo valore rappresentabile su 30 bit. Infine, viene salvato in una seconda memoria RAM (MEM\_B) da 10 celle, anch’essa con parallelismo a 30 bit.

Una volta terminato il calcolo di tutti i neuroni di output ed aver caricato i risultati su MEM\_B, viene generato un segnale di DONE. Il sistema aspetta che START sia a ‘0’ prima di poter ritornare allo stato di Idle ed eventualmente ricominciare il ciclo.

* 1. ***Parallelismi***

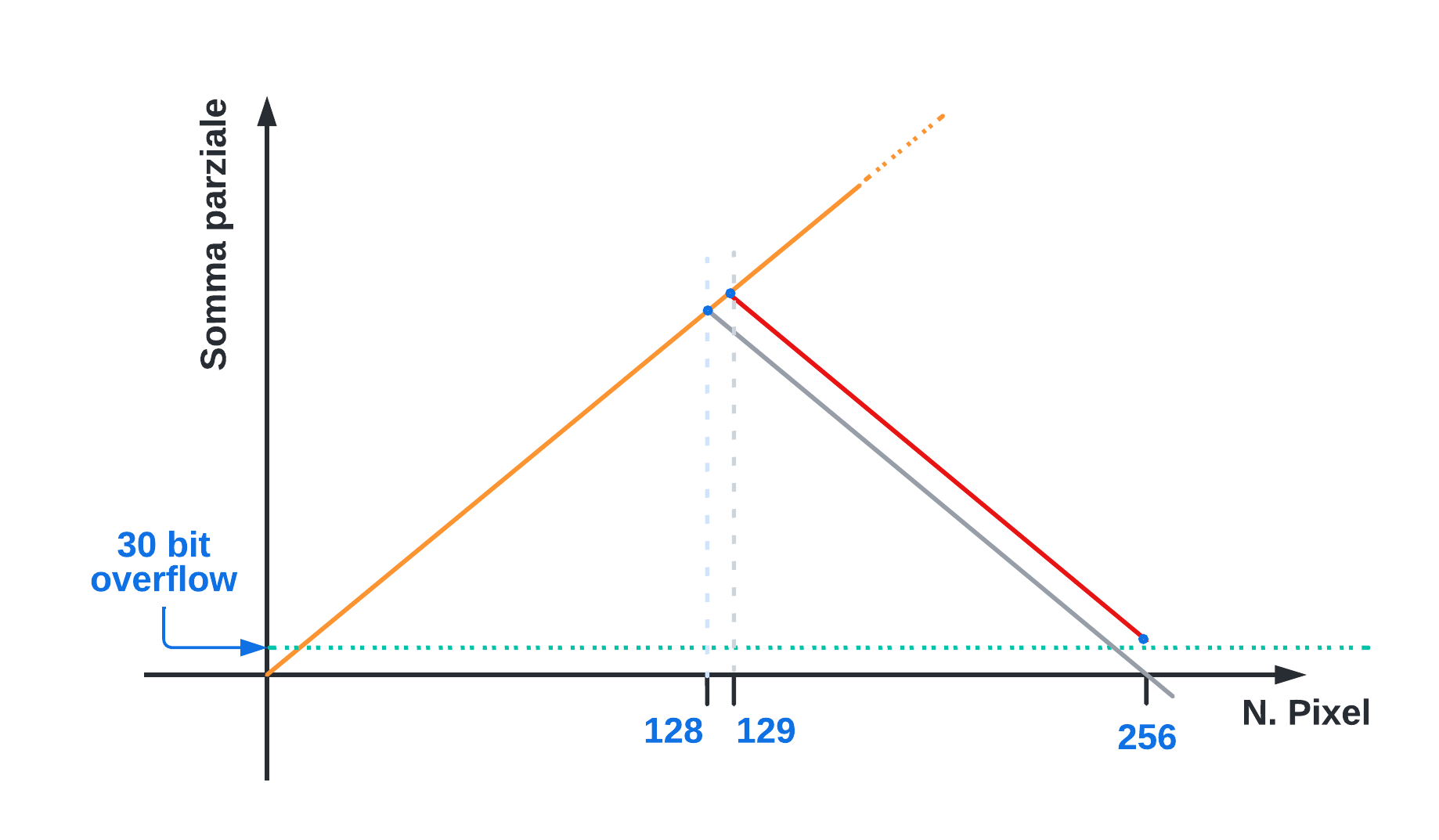
La procedura standard per la definizione del parallelismo in una serie di operazioni logiche consiste nell’individuare il massimo numero ottenibile e calcolarne il logaritmo in base due per capire quanti bit siano necessari per la sua rappresentazione binaria. In questo sistema, il worst case scenario consiste in 256 pixel con 30 bit a 1 e 256 pesi pari a 3[[2]](#footnote-2):

**log2(256 \* 3 \* (230 -1)) = 39.58 ≈ 40 bit**

In realtà, il punto critico del calcolo non si raggiunge dopo 256 pixel ma soltanto dopo 129. Nel caso in cui i primi 129 pixel di input fossero “1…1” con peso 3, si otterrebbe una somma parziale talmente alta da non essere recuperabile: se i pixel dal 130esimo al 256esimo infatti fossero “1…1” con peso -3, il risultato finale sarebbe comunque overflow su 30 bit, che è il parallelismo dei dati in output.

Questo permette di risparmiare un bit ed ottenere un parallelismo a 39 bit:

**log2(129 \* 3 \* (230 -1)) = 38.596 ≈ 39 bit**



**2.2*:*** *Rappresentazione grafica del calcolo worst case scenario. La linea rossa evidenzia come non sia possibile, una volta superato il 129° pixel, tornare ad un valore che non generi overflow su 30bit. La linea grigia evidenzia come dal 128° pixel invece sia possibile rientrare in valori accettabili*

1. ***Implementazione del sistema***

In questa sezione viene presentato e descritta l’implementazione della rete neurale descritta in precedenza. I file VHDL che descrivo il circuito e il programma in C utilizzato nella realizzazione del test bench sono allegati al presente documento, con il nome di “Allegato A – Codice C per generazione Test bench” e “Allegato B – Files VHDL”

* 1. ***Pseudocodice***

Il primo step nella progettazione del sistema digitale qui trattato consiste nella formulazione di uno pseudocodice, in modo da definire le operazioni fondamentali e l’ordine in cui eseguirle. Vengono esclusi dallo pseudocodice il segnali di start e reset.

// Inizializzazione valori della memoria a con i dati di input

for(i = 0;i<512;i++){

mem\_a = ext\_data;

}

for ( i = 0;i <10 ; i++){

for( j =0;j<256;j++){

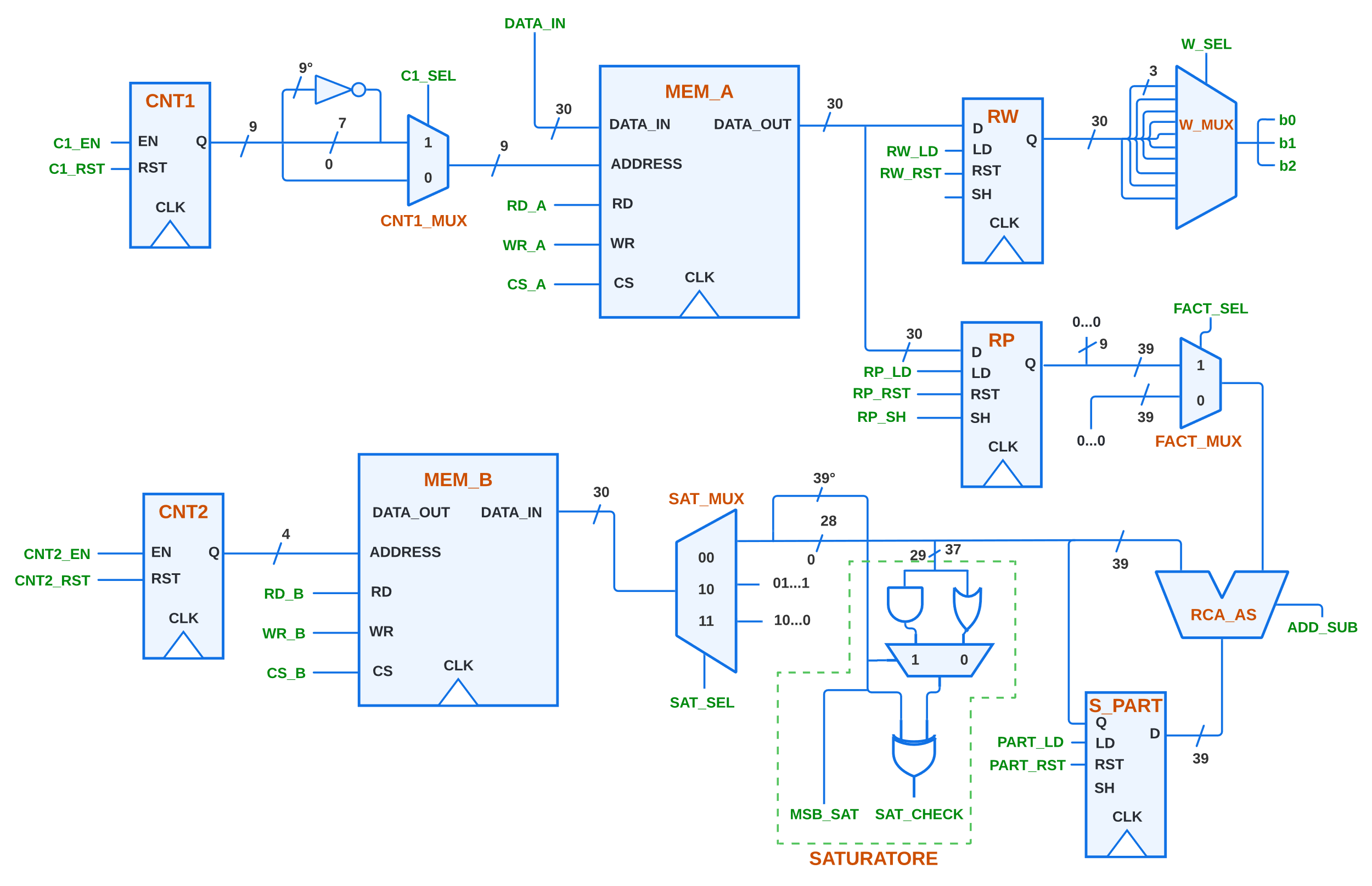
peso = mem\_a[j + 256][i, i + 1, i + 2]; // Estrazione del peso dalla memoria A

mem\_b[i] += mem\_a[j] \* peso; // Calcolo dell' uscita salvata nella memoria B

}

}

* 1. ***Data path***

In questa sezione viene descritta la struttura hardware del circuito, come illustrata nell’immagine 3.1. In seguito sono descritti in dettaglio i singoli componenti, il loro funzionamento e il loro ruolo all’interno del sistema. 

**3.1*:*** *Data path della rete neurale. Sono rappresentati con tag in nero i segnali interni dei componenti,*

*in verde i segnali utilizzati dalla Control Unit per gestire il sistema e in rosso il nome dei componenti*

1. ***Contatori***

I contatori hanno il compito di sincronizzare tutte le operazioni del sistema, dal caricamento dei dati in ingresso e uscita alle operazioni di somma. I contatori sono realizzati con toggle flip-flop e hanno reset RST e enable EN sincrono. Nel sistema sono presenti due counter:

* **CNT1:** Ha uscita con parallelismo a 9 bit, in quanto esprime gli indirizzi delle locazioni di memoria della MEM\_A. Per questo motivo deve essere capace di ciclare da 0 a 511.
* **CNT2:** Ha uscita con parallelismo a 4 bit, in quanto esprime gli indirizzi delle locazioni di memoria della MEM\_B. Per questo motivo deve essere capace di ciclare da 0 a 10.

1. ***Memorie RAM***

Le memorie hanno il compito di salvare i dati di ingresso e il risultato dell’algoritmo. Entrambe le memorie utilizzate hanno i segnali di read RD, write WR, e chip select CS sincroni e hanno parallelismo dei dati a 30 bit. MEM\_A e MEM\_B differiscono soltanto nel numero di celle: la prima ne ha 512, mentre la seconda solo 10.

1. ***Registri***

I registri utilizzati sono PIPO con reset RST, enable EN e shift-enable SH sincroni. I due registri RW e RP hanno lo scopo di memorizzare temporaneamente il pixel e il peso

i-esimo, per cui hanno parallelismo a 30 bit. S\_PART invece memorizza la somma parziale, necessaria per calcolare i prodotti fra pixel e pesi, per cui ha parallelismo a 39 bit.

1. ***Saturatore***

Si tratta di tre porte logiche e di un multiplexer, con lo scopo di creare dei segnali utilizzati dalla Control Unit per determinare se il risultato dell’algoritmo necessiti di saturazione prima di essere salvato nel neurone di output corrispondente. Se il numero da analizzare è positivo (MSB\_SAT = ‘0’), si realizza una OR ai bit 29-37, input di una XOR oltre a MSB\_SAT che genera il segnale SAT\_CHECK.

Viceversa se il numero da analizzare è negativo (MSB\_SAT = ‘1’), si realizza una AND ai bit 29-37. Anche in questo caso, SAT\_CHECK è una XOR fra la AND e MSB\_SAT. Di seguito è riportata la tabella della logica utilizzata:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Saturazione positiva | Saturazione negativa | Nessuna saturazione |
| MSB\_SAT: | 0 | 1 | -[[3]](#footnote-3) |
| SAT\_CHECK: | 1 | 1 | 0 |

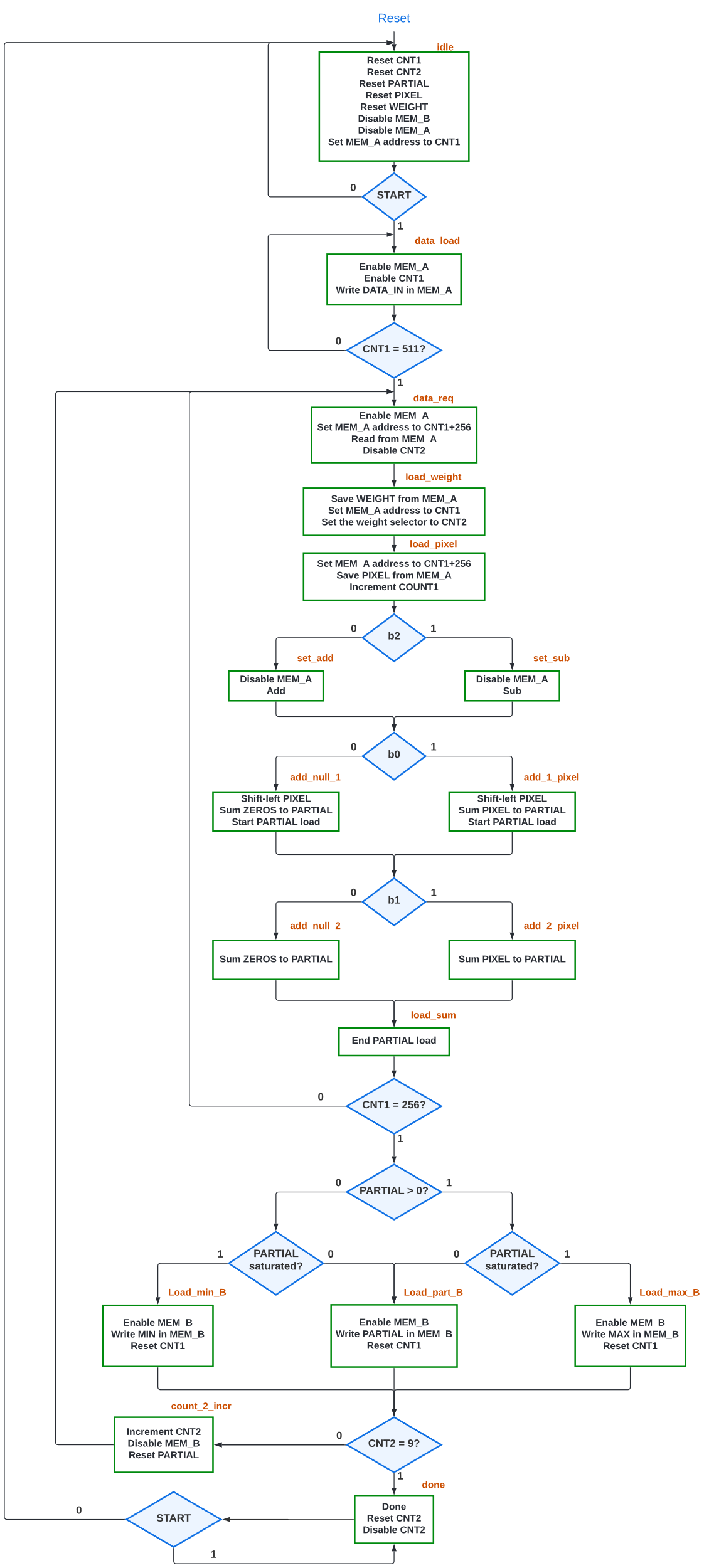
1. ***Multiplexer***

I MUX utilizzati hanno diversi scopi:

* **CNT1\_MUX:** 9 bit 2to1, permette di selezionare l’address della MEM\_A fra l’output del CNT1 e quest’ultimo con MSB negato, realizzando la somma con 256
* **W\_MUX**: 3bit 10to1, permette di selezionare all’interno dei 30bit di peso del pixel il peso specifico di un singolo neurone di output
* **FACT\_MUX:** 39bit 2to1, fondamentale nelle operazioni di prodotto e somma
* **SAT\_MUX:** 30bit 3to1, realizza la saturazione del neurone di output su 30 bit.

1. ***Adder/Subtracter:***

Per quanto riguarda le operazioni aritmetiche, si è scelto di utilizzare un adder-subtracter realizzato a partire da un ripple carry adder. Il bit di selezione ADD\_SUB realizza l’XOR bit a bit del secondo fattore e setta il carry d’ingresso a ‘1’. Così facendo si ottiene la somma del fattore A e del complemento a 2 del fattore B, realizzando di fatto una sottrazione. La frequenza di clock massima nel worst-case scenario non deve superare circa 152 MHz[[4]](#footnote-4), per permettere all’RCA di svolgere le operazioni in un singolo colpo di clock.

* 1. ***ASM chart dell’algoritmo***

**3.2*:*** *ASM algoritmico della rete neurale con stati e controlli*

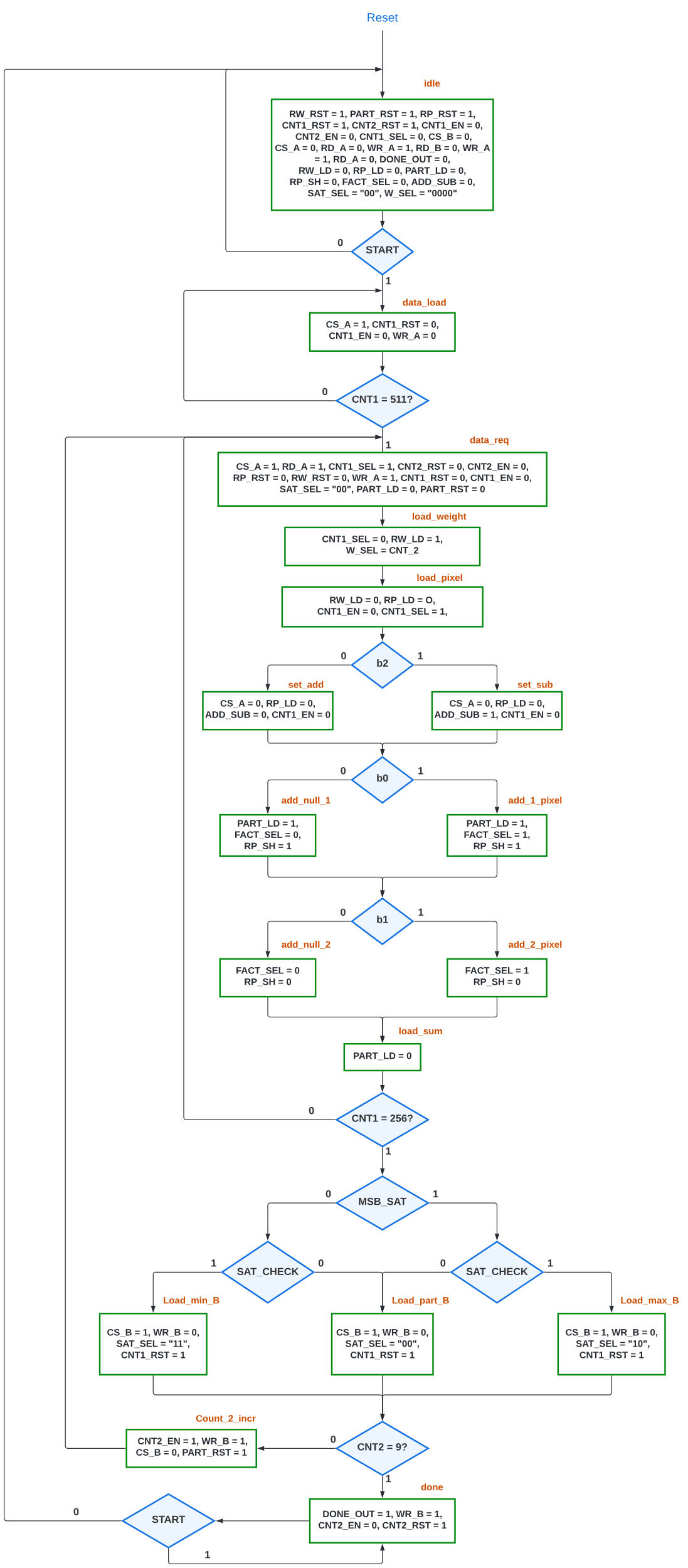
Lo schema riportato nella figura 3.2 descrive l’algoritmo utilizzato nell’implementazione della rete neurale. Come descritto nelle specifiche di progetto, le operazioni da svolgere sono due:

* Calcolare il prodotto fra pixel e pesi:
* Sommare tali prodotti per ottenere il valore del neurone d’uscita corrispondente:

Entrambe le operazioni sono realizzate da un unico adder grazie ad una temporizzazione accurata dei registri di pixel, pesi e somme parziali. La procedura è la seguente:

|  |  |  |
| --- | --- | --- |
| Peso | 1° passaggio | 2° passaggio |
| 0 | Somma 0 a parziale | Somma 0 a parziale |
| 1 | Somma 1 a parziale | Somma 0 a parziale |
| 2 | Somma 0 a parziale | Somma 2 a parziale |
| 3 | Somma 1 a parziale | Somma 2 a parziale |

Per sommare alla parziale “2” viene eseguito uno shift del registro di pixel RP, mentre per sommare “0” si controlla il FACT\_MUX. Una volta eseguita l’operazione a) di cui sopra, il sistema carica la successiva coppia pixel-peso e così via, fino alla lettura completa di MEM\_A. Successivamente, il risultato passa attraverso un processo di saturazione e viene caricato in MEM\_B.

* 1. ***ASM chart della control unit***

1. ***Time analysis***

In questa sezione viene riportata la timing analysis del sistema, comparando la versione realizzata a mano e basata su ASM Control Unit con i risultati della simulazione con ModelSim.

* 1. ***Generazione di dataset casuali***

Realizzare un dataset adeguato è fondamentale per testare qualunque tipo di sistema di elaborazione dati. Per questo motivo e data la mole di dati in entrata[[5]](#footnote-5) si è optato per l’automazione completa del processo, realizzando un codice in C.

Quest’ultimo è in grado di generare dataset differenti ogni volta che viene eseguito, calcolare il valore dei singoli neuroni di output e salvare i dati in un file .txt denominato “rand\_data.txt”. Per testare efficacemente ogni funzionalità del sistema, incluso il processo di saturazione, sono stati fissati i pesi relativi all’output 0 a -3, in modo da avere sempre una saturazione negativa.

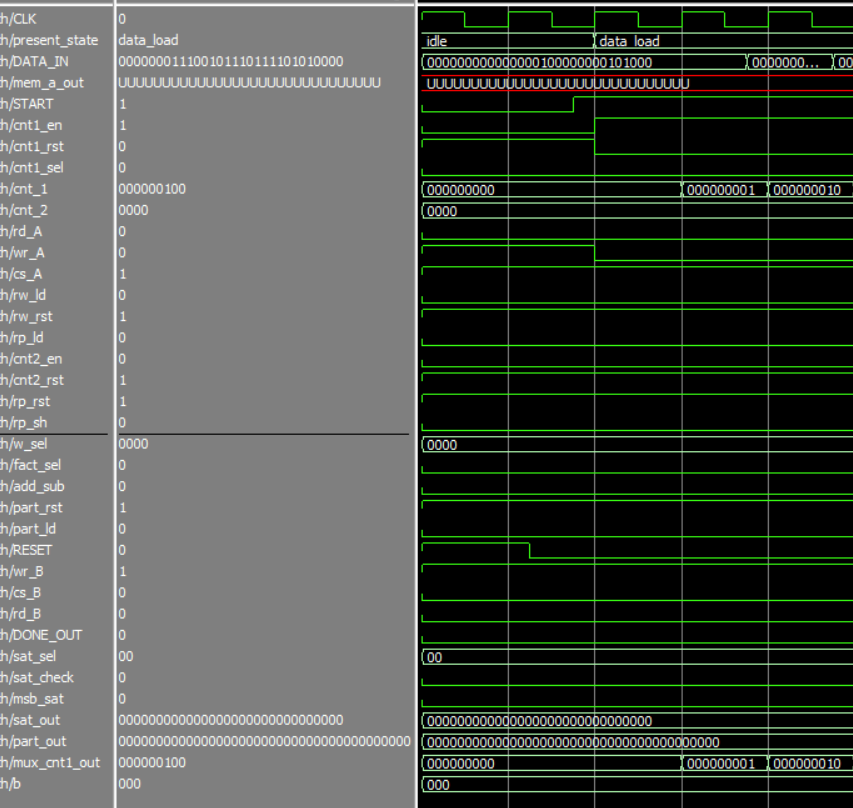
Una volta ottenuto “rand\_data.txt”, il testbench viene completato inserendo la dichiarazione dell’entity, architecture e del componente “neural\_network”, il VHDL principale del progetto. L’intero codice C è riportato nell’allegato “Allegato A – Codice C per generazione Test bench” al presente documento, così come il testbench utilizzato per la simulazione ModelSim successivamente riportata, “Allegato B – Files VHDL”.

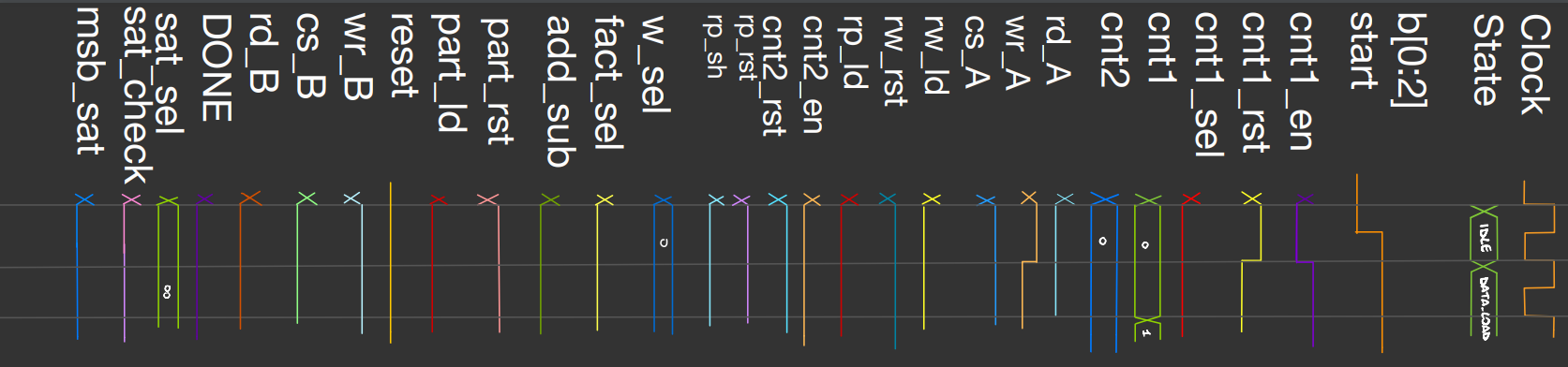
* 1. ***Risultati attesi e simulazione***

Di seguito si riportano alcuni passaggi fondamentali nello svolgimento dell’algoritmo, con focus sui segnali principali, confrontando i risultati aspettati e ciò che si evince dalla simulazione del test bench su ModelSim.

* **Fase di avviamento del sistema**

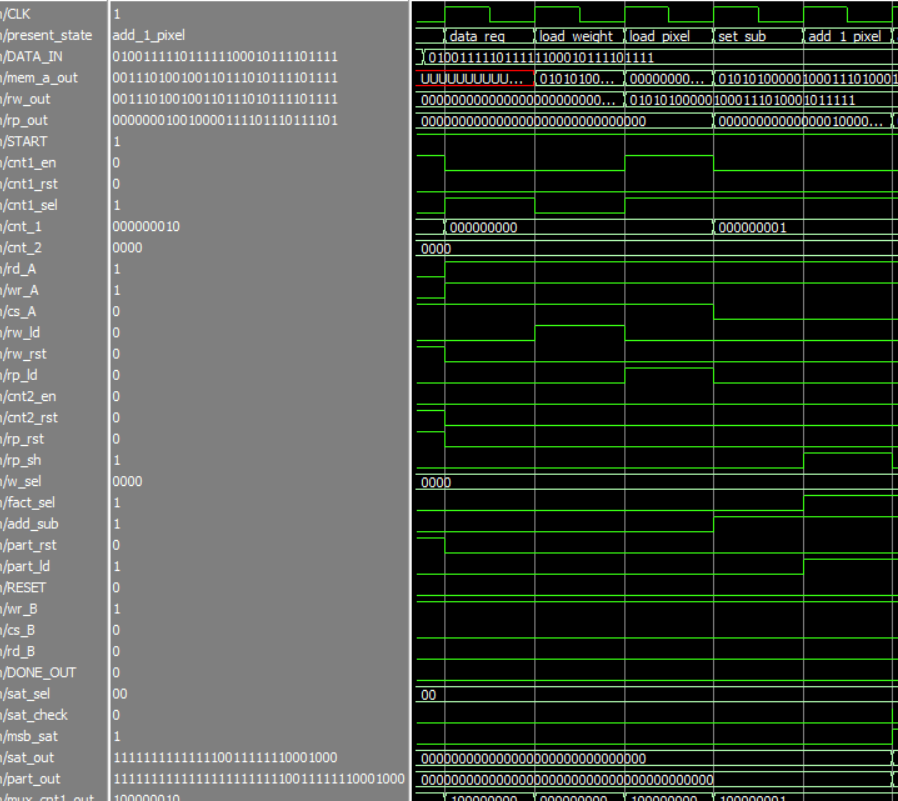
Inizialmente si ha RESET alto e START basso, molti segnali ancora non sono stati inizializzati e il sistema è in stato di *idle*. Quando lo START viene campionato a ‘1’, il processo inizia, il sistema entra in *load\_data* e al colpo di clock successivo viene salvato il primo dato nella prima cella di memoria di MEM\_A. Notare come le uscite di MEM\_A siano undefined nella simulazione in quanto essa sia in modalità di scrittura. L’address della memoria è il segnale “MUX\_CNT1\_OUT”:





* **Caricamento pixel e pesi**

Una volta terminato il caricamento della memoria MEM\_A, inizia l’esecuzione dell’algoritmo vero e proprio. Lo stato del sistema passa prima in *load\_weight* in cui viene caricato il peso nel registro RW, come si può vedere dall’uscita dellostesso al clock successivo (rw\_out). Successivamente si passa in *load\_pixel*, stato analogo al precedente per quanto riguarda i pixel. In questa fase è importante la temporizzazione dell’address di MEM\_A, in quanto deve variare fra CNT1 e CNT1+256 a seconda se si stia caricando pesi o pixel. Ciò avviene grazie al segnale cnt1\_sel.



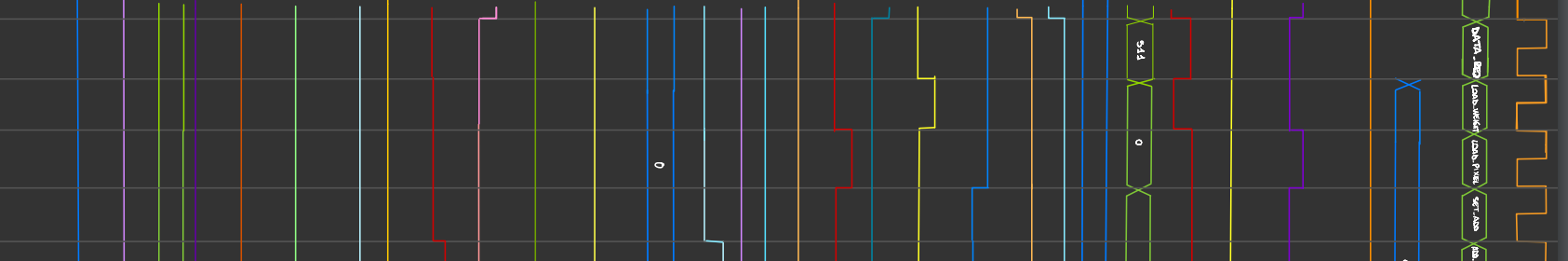
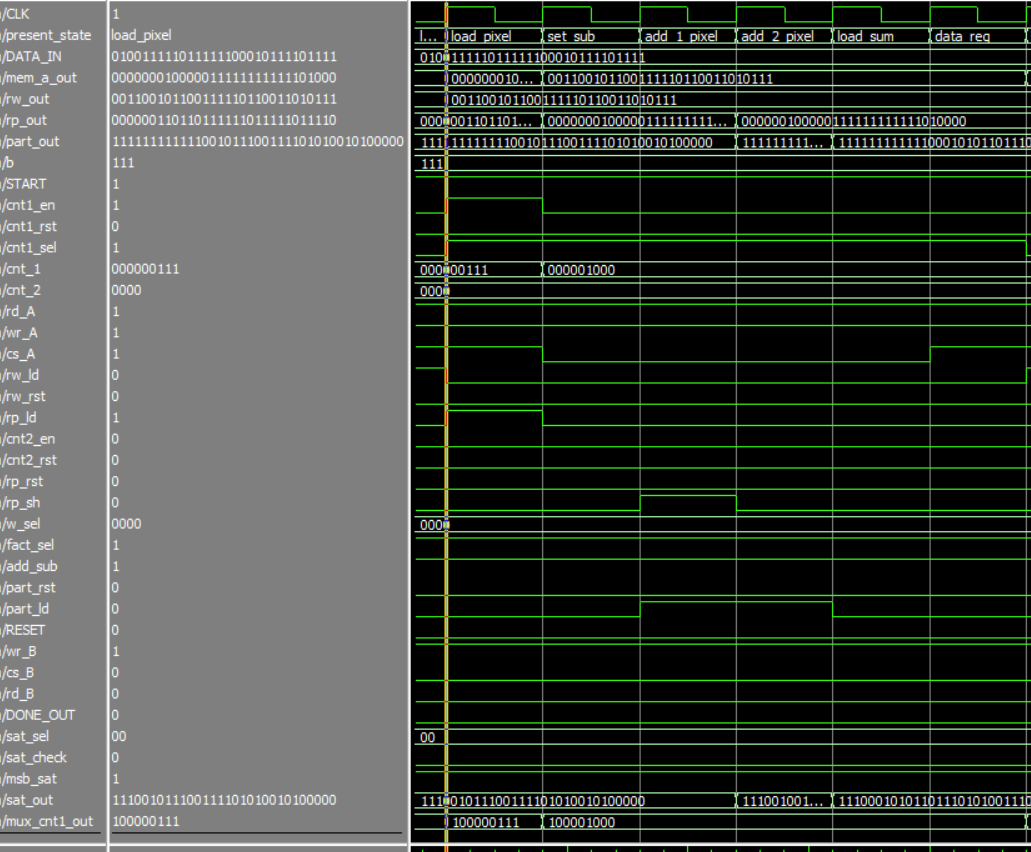


Immagine che contiene testo, elettronico

Descrizione generata automaticamente

* **Calcolo del prodotto fra pesi \* pixel e somma parziale**

In questa fase si concretizza l’algoritmo del sistema. Nell’esempio della simulazione sotto riportato, il segnale “b” rappresenta il peso ed è “111”, -3. Per questo motivo il sistema inizialmente è nello stato *set\_sub* e imposta l’adder in modalità subtracter con il bit add\_sub a ‘1’, successivamente passa in *add\_1\_pixel* e *add\_2\_pixel*, seguendo gli stati alla tabella del paragrafo 3.3. Sia nell’esempio simulato che nella timing analysis manuale negli stati *add\_1\_pixel* viene dato il segnale per realizzare lo shift del pixel in ingresso. Tale processo nella simulazione è constatabile dall’uscita del registro RW. Una volta terminato il prodotto il sistema entra in *load\_sum* per poter salvare la somma parziale in S\_PART.

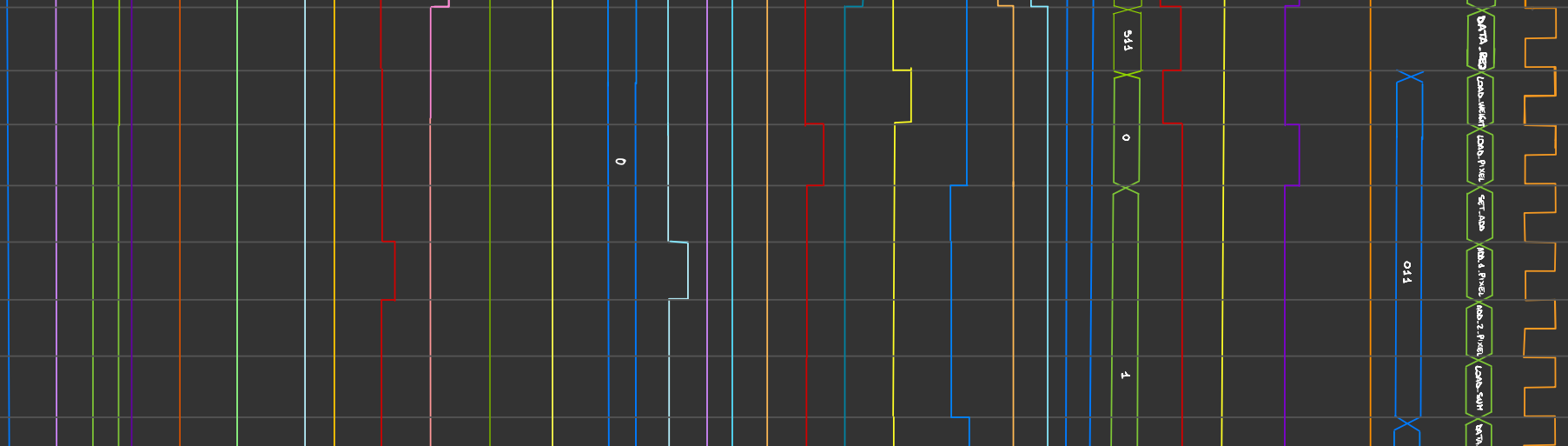


Immagine che contiene testo, elettronico

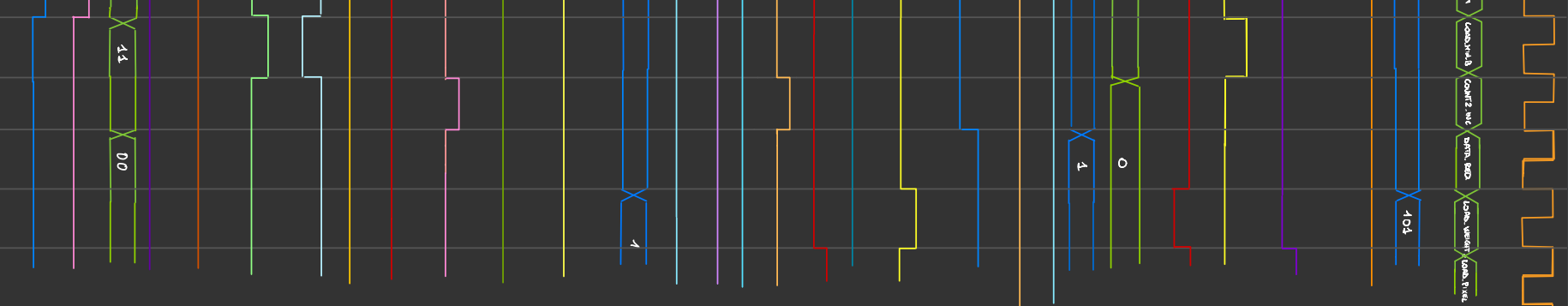
Descrizione generata automaticamente

* **Output ed eventuale saturazione**

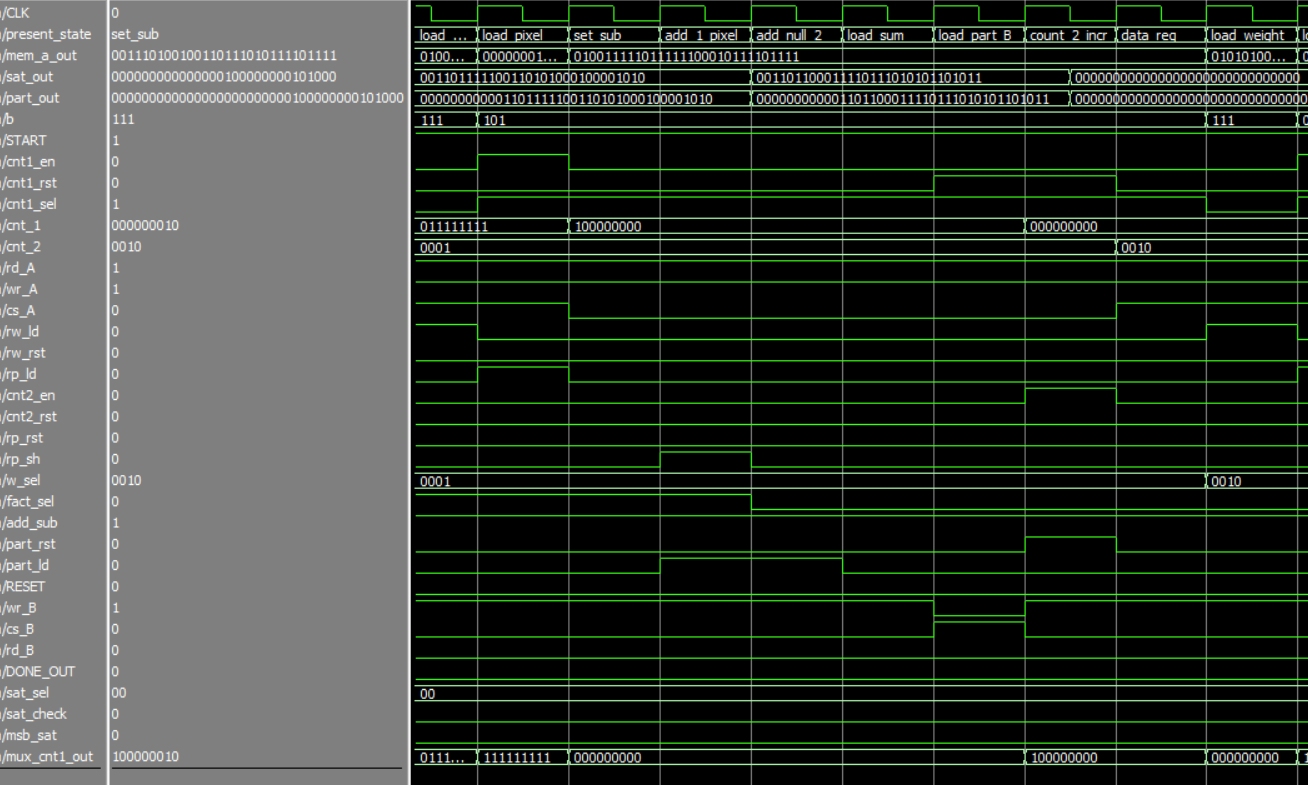
In questa fase viene concluso il calcolo del valore del neurone di output e se supera il massimo valore rappresentabile su 30bit subisce un processo di saturazione, che può essere positiva o negativa.Immagine che contiene tavolo

Descrizione generata automaticamenteImmagine che contiene testo, elettronico

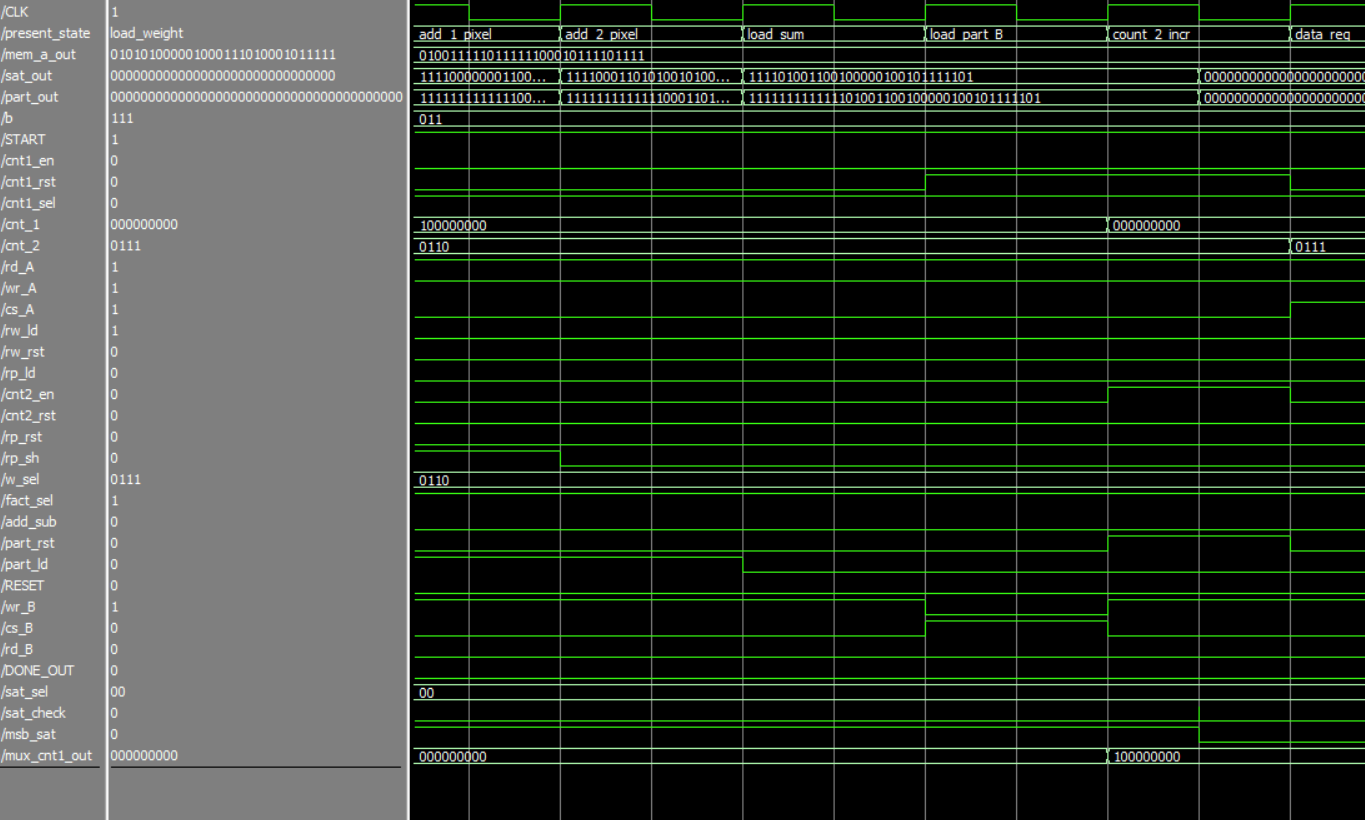
Descrizione generata automaticamente Come riportato nel testbench, l’output del primo neurone è saturo negativamente, ed è esattamente ciò che succede nella simulazione: il sistema va nello stato *load\_min\_b* e come si può vedere dal segnale “sat\_out” carica nella MEM\_B il valore saturo negativamente “1000…000”. Una volta fatto ciò, CNT2 viene incrementato, CNT1 e i registri si resettano e il ciclo ricomincia



Di seguito si riporta il risultato del secondo output dato dal segnale sat\_out, il quale corrisponde al valore atteso, ovvero “001101100011110111010101101011”:



Si riporta anche il valore del settimo output, ovvero “111101001100100000100101111101”



* **Completamento algoritmo**

Una volta terminato il calcolo del decimo neurone di output, il sistema produce un segnale di DONE e, una volta che START è campionato a ‘0’, torna nello stato di *idle*

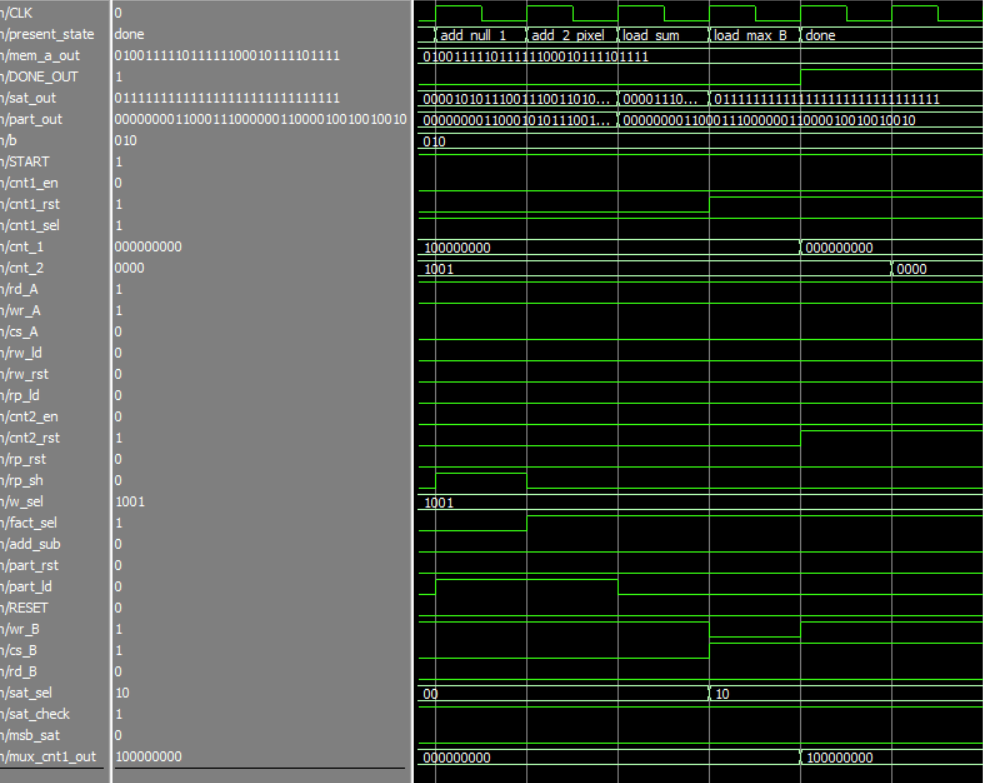
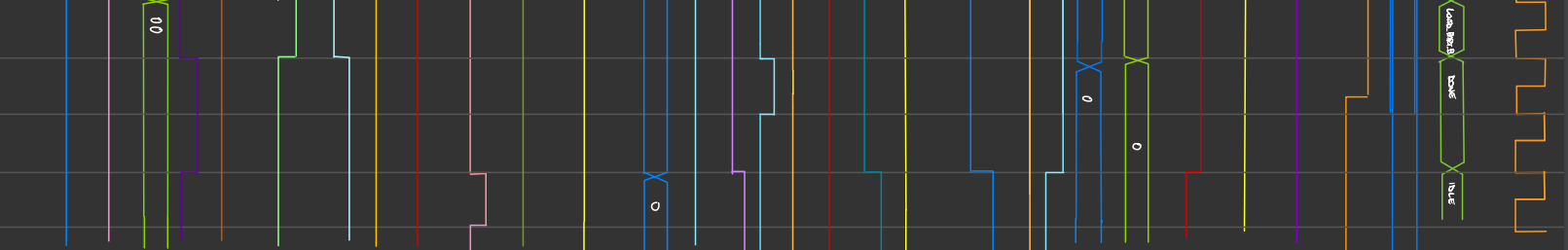


Immagine che contiene testo, elettronico

Descrizione generata automaticamente**

1. *Modified National Institute of Standards and Tecnology* [↑](#footnote-ref-1)
2. *Vale anche nel caso in cui i pesi fossero tutti a -3* [↑](#footnote-ref-2)
3. *Il simbolo “-“ significa “Don’t Care”, in quanto ai fini della CU il valore del bit in tale configurazione è irrilevante* [↑](#footnote-ref-3)
4. *Questo è quanto risulta dalla simulazione dell’RCA su ModelSim* [↑](#footnote-ref-4)
5. *512 word da 30 bit* [↑](#footnote-ref-5)