



Politecnico di Torino  
Corso di laurea in Ingegneria Elettronica

# Relazione Laboratorio 6 Elettronica dei sistemi digitali

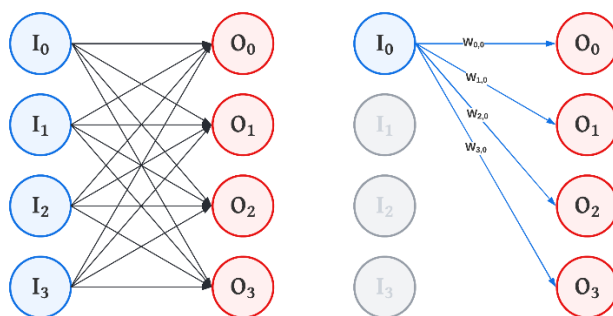
## Implementazione hardware di una rete neurale per classificare caratteri MNIST

*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  $16 \times 16$ , con pixel definiti a 30 bit, rappresentante una cifra da 0 a 9 scritta a mano, ricavata dal dataset MNIST<sup>1</sup>.

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 esperienza, 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.

<sup>1</sup> Modified National Institute of Standards and Technology

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:

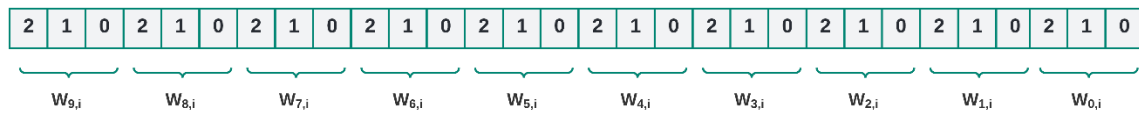
$$O_j = \sum_{i=0}^{N-1} I_i \cdot W_{j,i}$$

1.2:  $O_j$  è l'output j-esimo,  $I_i$  è l'input i-esimo e  $W_{j,i}$  è il peso della sinapsi

## 2 Specifiche di progetto

### 2.1 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 ( $\overline{WR}$ ). 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.

## 2.2 Parallelismi

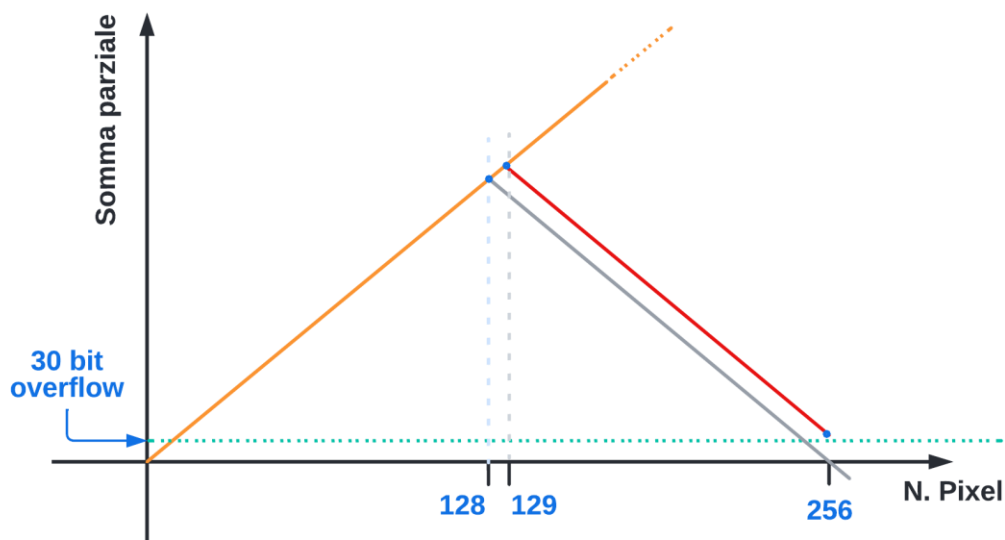
La procedura standard per la definizione del parallelismo in una serie di operazioni logiche consiste nell'individuare il massimo numero ottenibile e calcolarne il logaritmo in base due per capire quanti bit siano necessari per la sua rappresentazione binaria. In questo sistema, il worst case scenario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

$$\log_2(256 * 3 * (2^{30} - 1)) = 39.58 \approx 40 \text{ bit}$$

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:

$$\log_2(129 * 3 * (2^{30} - 1)) = 38.596 \approx 39 \text{ bit}$$



**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

<sup>2</sup> Vale anche nel caso in cui i pesi fossero tutti a -3

### 3 Implementazione del sistema

In questa sezione viene presentato e descritta l'implementazione della rete neurale descritta in precedenza. I file VHDL che descrivono 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"

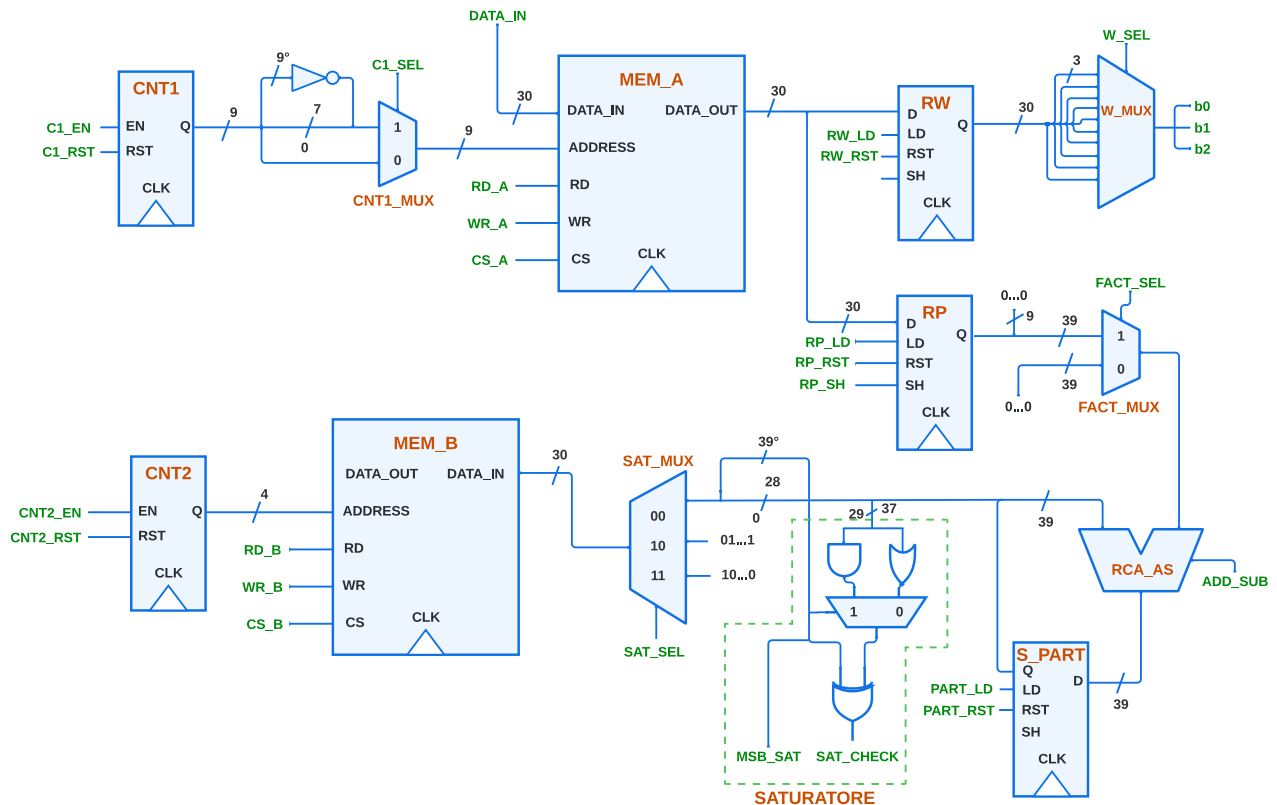
#### 3.1 Pseudocodice

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
    }
}
```

## 3.2 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

### a) 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.

### b) 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.

### c) **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.

### d) **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 necessita 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
SAT_CHECK:	1	1	0

### e) **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.

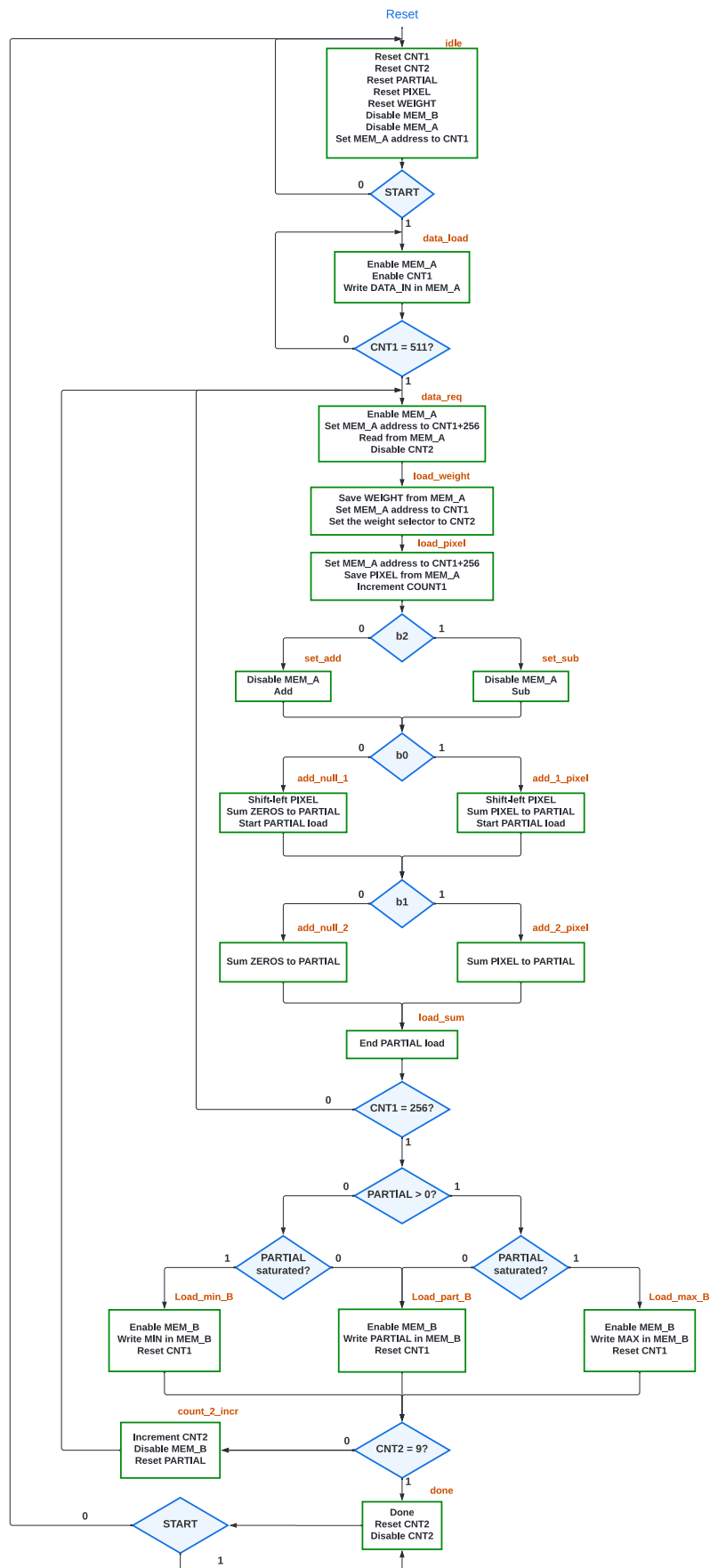
### f) **Adder/Subtracter:**

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<sup>4</sup>, per permettere all'RCA di svolgere le operazioni in un singolo colpo di clock.

<sup>3</sup> Il simbolo "-" significa "Don't Care", in quanto ai fini della CU il valore del bit in tale configurazione è irrilevante

<sup>4</sup> Questo è quanto risulta dalla simulazione dell'RCA su ModelSim

### 3.3 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:  $x_i = p_i \cdot w_i$
- Sommare tali prodotti per ottenere il valore del neurone d'uscita corrispondente:  

$$o_j = \sum_i x_i$$

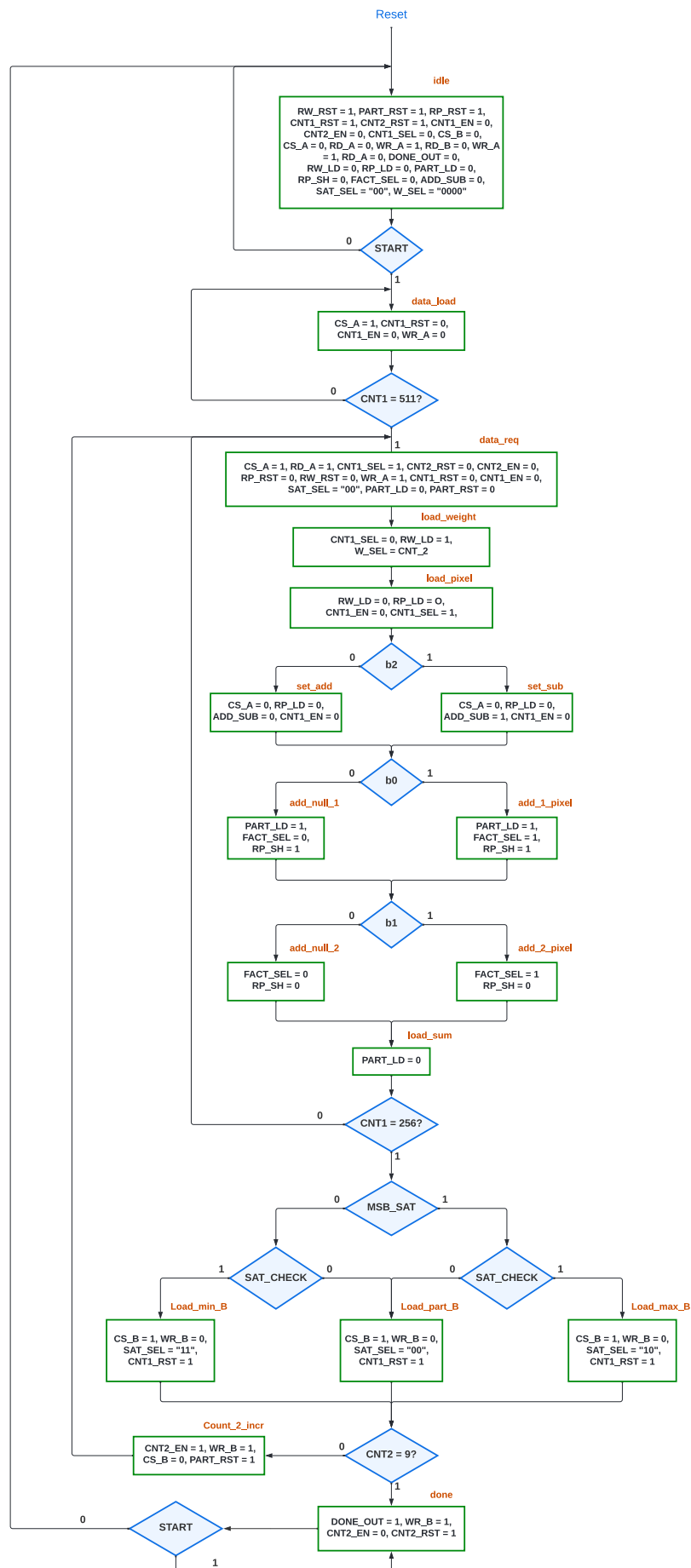
Entrambe le operazioni sono realizzate da un unico adder grazie ad una temporizzazione accurata dei registri di pixel, pesi e somme parziali. La procedura è la seguente:

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.



### 3.4 ASM chart della control unit



## 4 *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.

### 4.1 *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<sup>5</sup> 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".

---

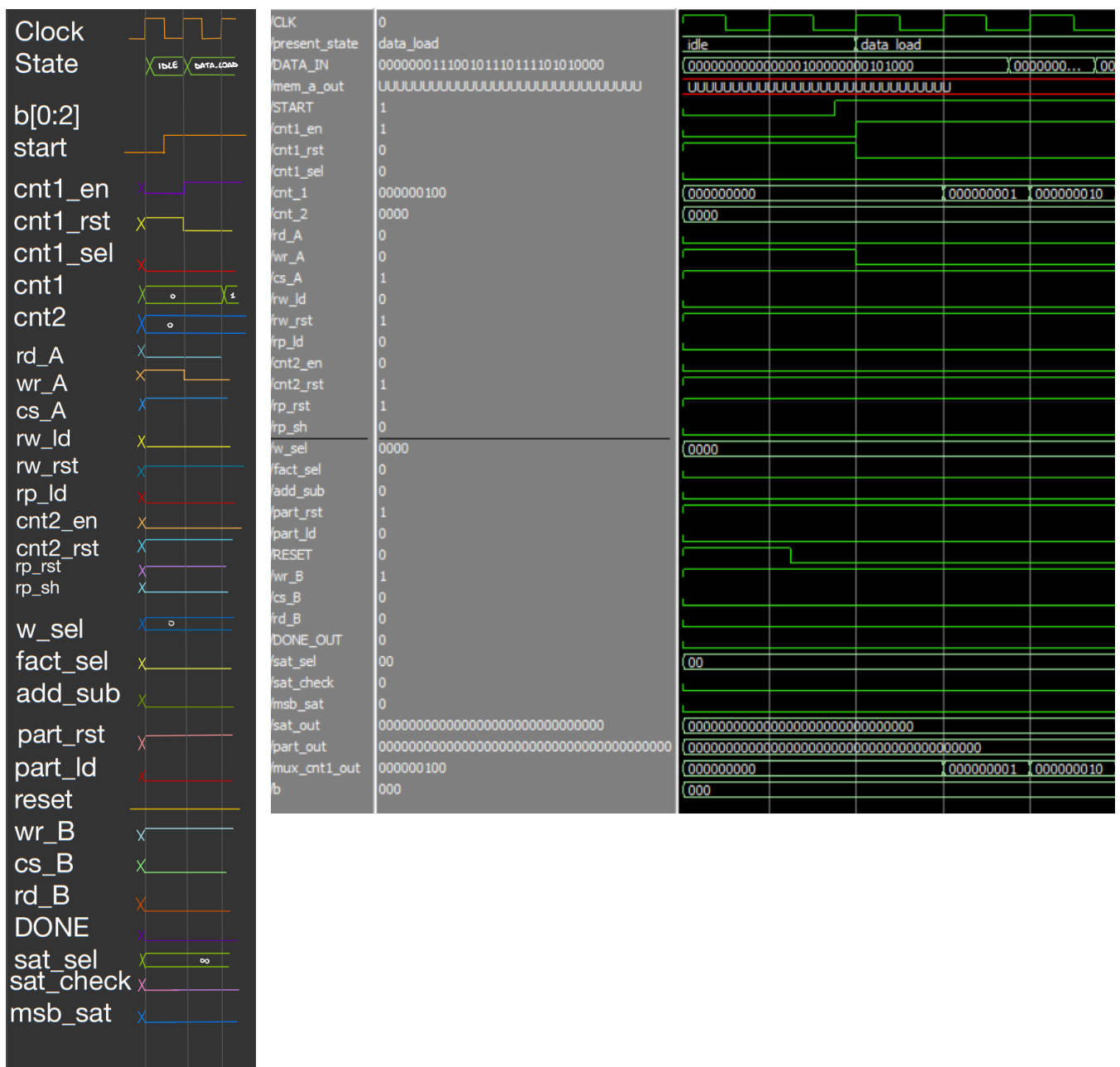
<sup>5</sup> 512 word da 30 bit

## 4.2 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.

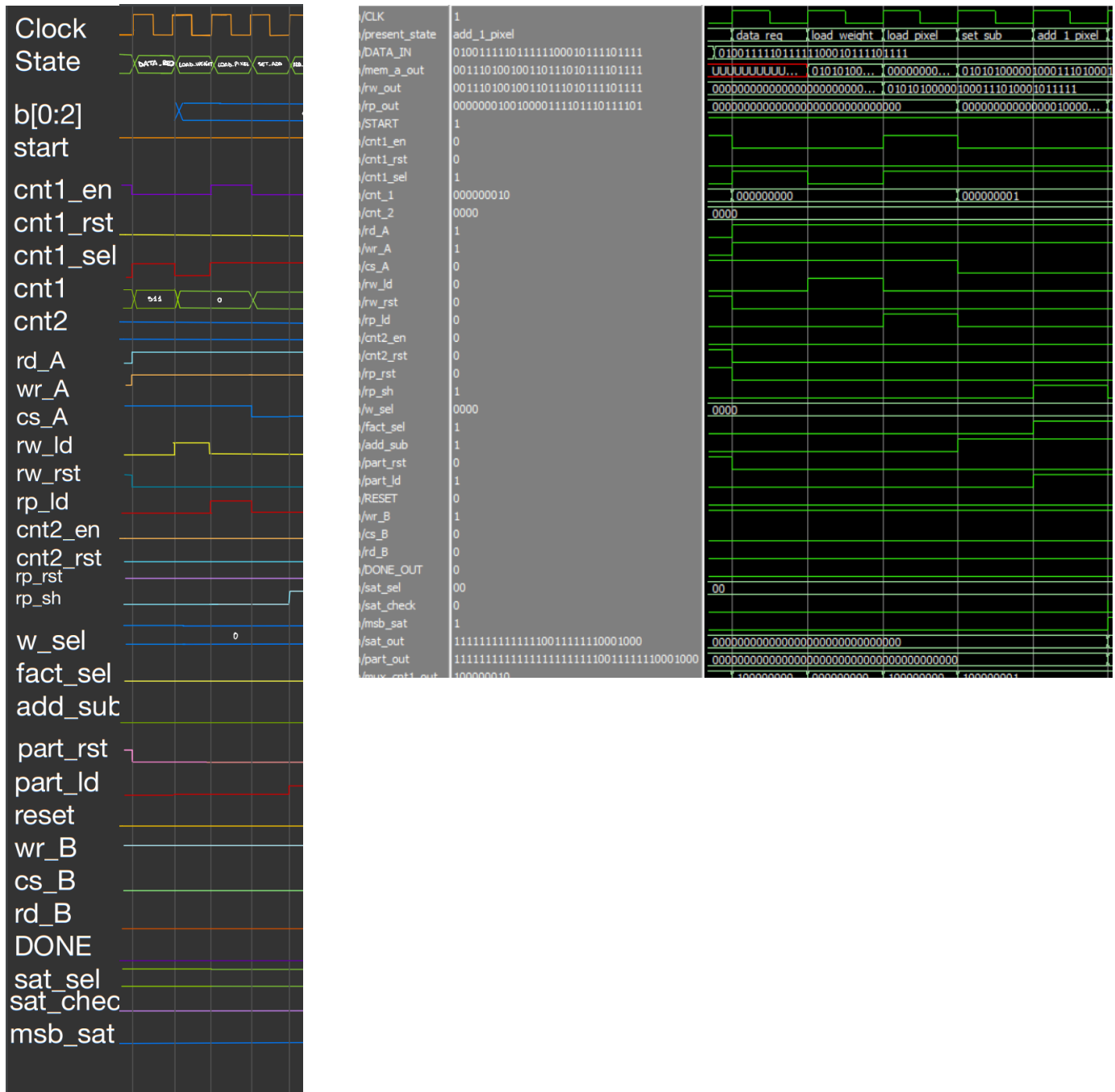
- **Fase di avviamento del sistema**

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":



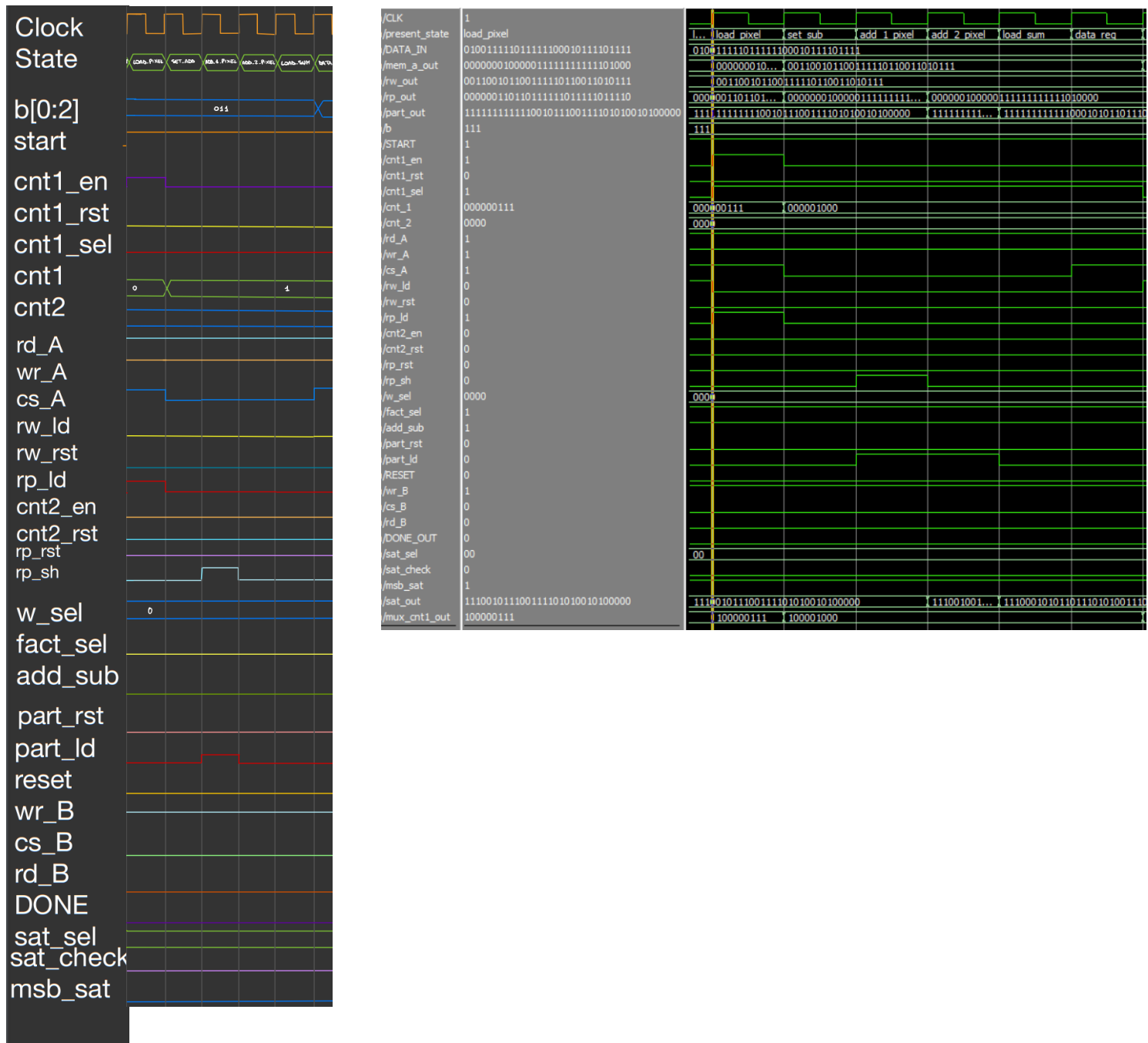
- **Caricamento pixel e pesi**

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 dello stesso 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.



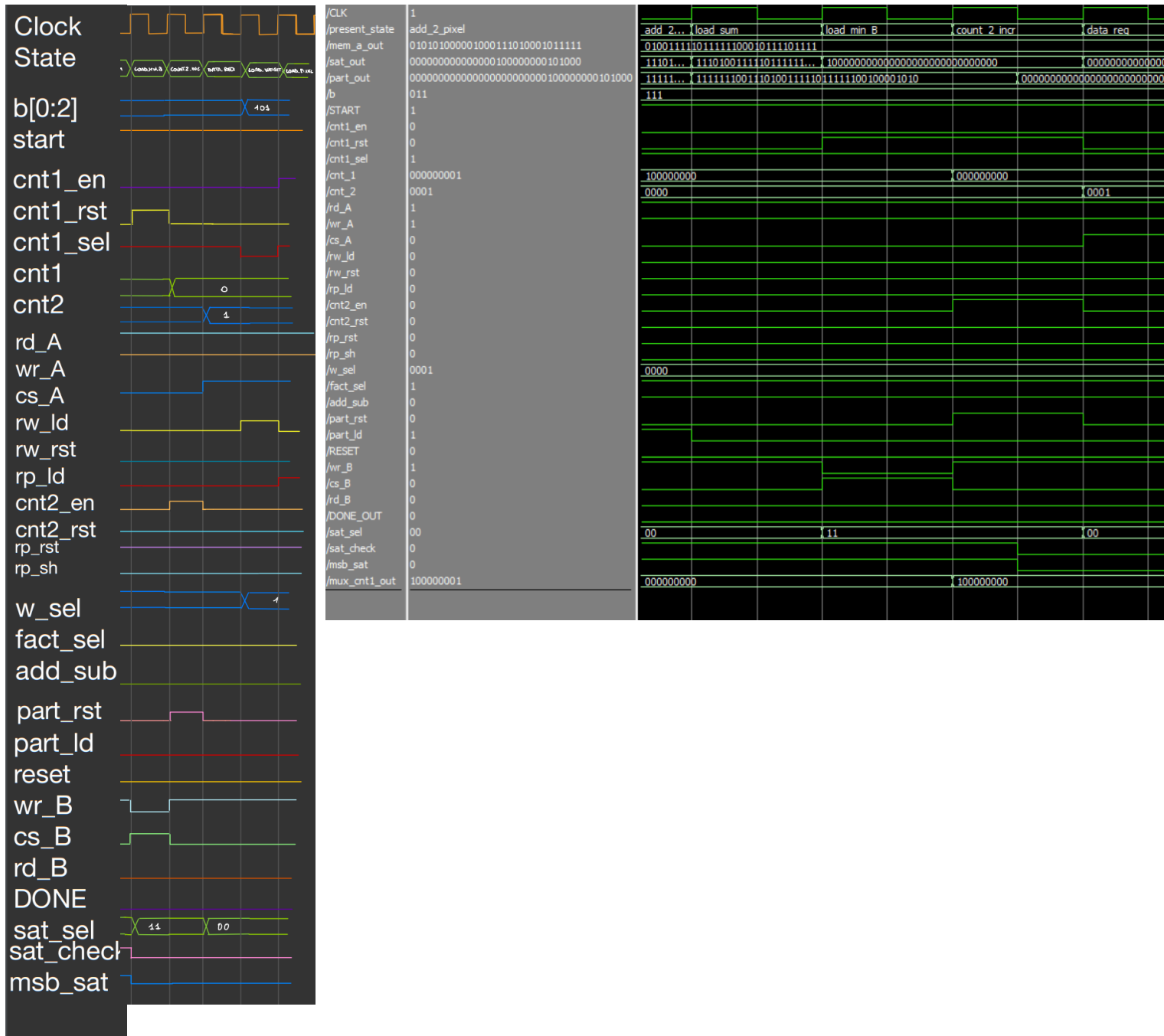
- **Calcolo del prodotto fra pesi \* pixel e somma parziale**

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.



- **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. 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":

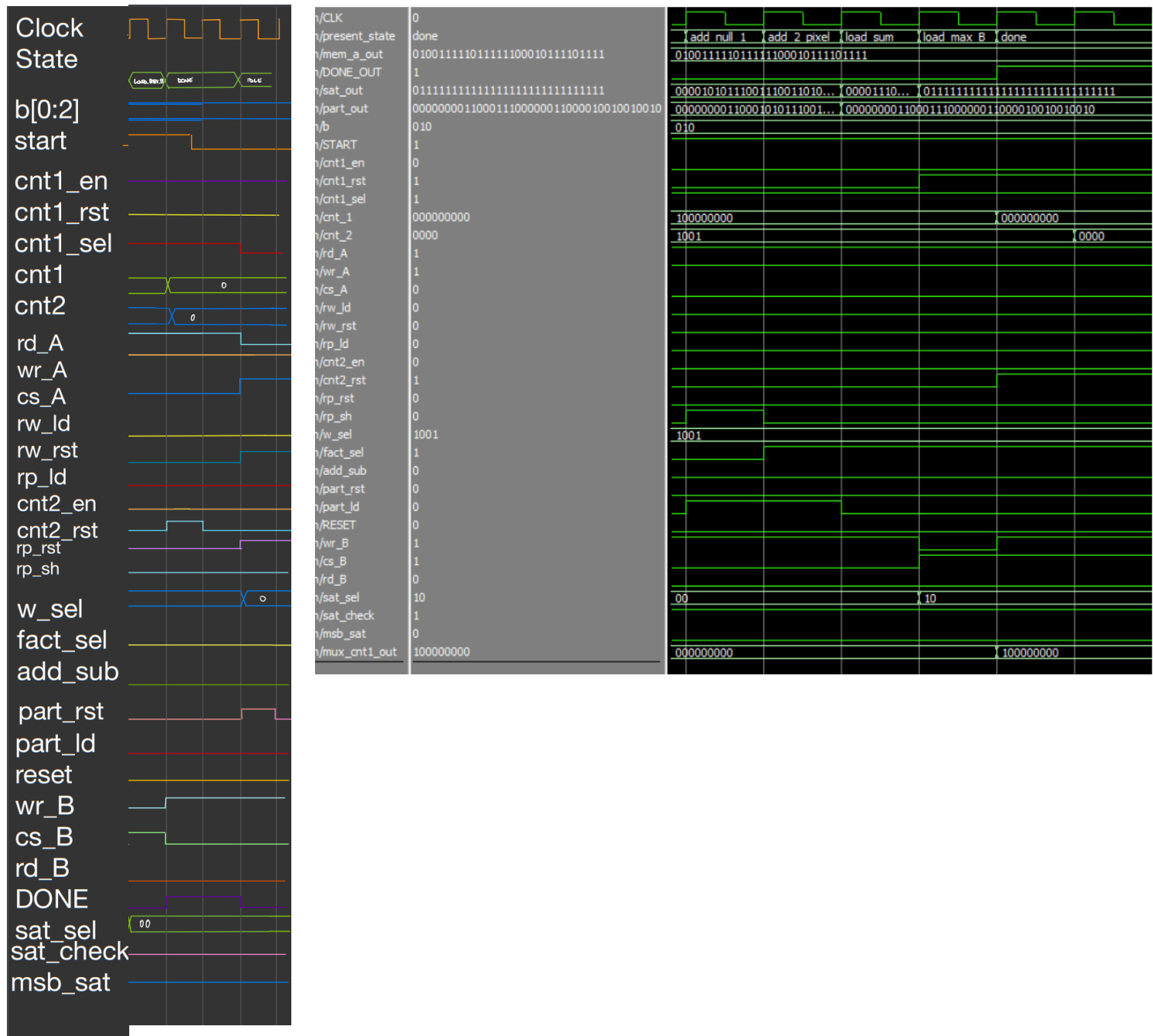
CLK	set_sub	load ...	load pixel	set sub	add 1 pixel	add null 2	load sum	load part B	count 2 incr	data req	load_weight
/present_state	001110100100110110101110101111	0100...	00000001...	010011111011111100010111101111							01010100...
/mem_a_out	00000000000000001000000000101000	0011011111100110101000100001010					0011011000011110111010101101011		00000000000000000000000000000000		0000000000000000
/sat_out	00000000000000000000000000101000	000000000001101111100110101000100001010					00000000000011011000111101110101101011	011	00000000000000000000000000000000		0000000000000000
/b	111	111	101								111
/START	1										
/cnt1_en	0										
/cnt1_rst	0										
/cnt1_sel	1										
/cnt_1	000000010	011111111		100000000				000000000			
/cnt_2	0010	0001							0010		
/rd_A	1										
/wr_A	1										
/cs_A	0										
/rw_ld	0										
/rw_rst	0										
/rp_ld	0										
/cnt2_en	0										
/cnt2_rst	0										
/rp_rst	0										
/rp_sh	0										
/w_sel	0010	0001									0010
/fact_sel	0										
/add_sub	1										
/part_rst	0										
/part_ld	0										
/RESET	0										
/wr_B	1										
/cs_B	0										
/rd_B	0										
/DONE_OUT	0										
/sat_sel	00	00									
/sat_check	0										
/msb_sat	0										
/mux_cnt1_out	100000010	0111... 111111111		000000000					100000000		000000000

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

CLK	1								
/present_state	load_weight		add 1 pixel	add 2 pixel	load sum		load part B		count 2 incr
/mem_a_out	010101000001000111010001011111	0100111110111111000	10111101111						
/sat_out	00000000000000000000000000000000	1111000000001100...	11110001101010010100...	111101001100100000010010111101					00000000000000
/part_out	00000000000000000000000000000000	11111111111100...	1111111111110001101...	11111111111110100110010000010010111101					00000000000000
/b	111	011							
/START	1								
/cnt1_en	0								
/cnt1_rst	0								
/cnt1_sel	0								
/cnt_1	000000000	100000000							000000000
/cnt_2	0111	0110							
/rd_A	1								
/wr_A	1								
/cs_A	1								
/rw_ld	1								
/rw_rst	0								
/rp_ld	0								
/cnt2_en	0								
/cnt2_rst	0								
/rp_rst	0								
/rp_sh	0								
/w_sel	0111	0110							
/fact_sel	1								
/add_sub	0								
/part_rst	0								
/part_ld	0								
/RESET	0								
/wr_B	1								
/cs_B	0								
/rd_B	0								
/DONE_OUT	0								
/sat_sel	00	00							
/sat_check	0								
/msb_sat	0								
/mux_cnt1_out	000000000	000000000							100000000

- **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*





# Allegato A – Codice C per generazione Test bench

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define UPPLIM RAND_MAX
#define LOWLIM RAND_MAX/2

void dectobin(unsigned int n, short **arr)
{
    // INPUT : NUMERO INTERO DECIMALE ; OUTPUT : NUMERO N ESPRESSO IN BINARIO UNSIGNED SU 30
    BIT
    unsigned int d = n;
    int i = 0;
    int res = 0;
    for(i=0;n>0;i++)
    {
        arr[i]=n%2;
        res = arr[i]=n%2;
        n /= 2;
    }
    while(i < 30)
    {
        arr[i] = 0;
        i++;
    }
}

void pr_arr_file(short **arr, FILE *fp)
{

```

```

//SCRITTURA SU FILE DEL NUMERO BINARIO
for(int i = 29;i >= 0;i--)
{
    fprintf(fp,"%d", arr[i]);
}
}

void printn_bin_file(unsigned int n, short **arr,FILE *fp)
{
    //SCRITTURA SU FILE DEL NUMERO DECIMALE n IN FORMATO BINARIO
    dectobin(n, arr);
    pr_arr_file(arr, fp);
}

int dec_weight(const int *vec)
{
    // CONVERSIONE DELL'ARRAY DI PESI IN NUMERO INTERO DECIMALE
    int res = 0;
    for(int i = 0;i < 2;i++)
    {
        if(vec[i] == 1)
        {
            res += pow(2,i);
        }
    }
    if(vec[2] == 1)
    {
        res = - res;
    }
    return res;
}

void dec_to_2s_39bit(long long dec, unsigned int **arr)
{
    // INPUT : INTERO IN DECIMALE ; OUTPUT : NUMERO BINARIO IN COMPLEMENTO A 2 SU 39 BIT
    int flag = 0;
    int i = 0;

```

```

int neg = 0;
if(dec < 0)
{
    dec = - dec;
    neg = 1;
}
//conversione in binario
for(i=0;dec>0;i++)
{
    arr[i]=dec%2;
    dec /= 2;
}
while(i < 39)
{
    arr[i] = 0;
    i++;
}
if(neg)
{
    //complemento ad 1 di conv
    for(i = 0;i < 39;i++)
    {
        if(arr[i] == 1)
        {
            arr[i] = 0;
        }
        else arr[i] = 1;
    }
    i = 0;
    //complemento a 2
    while(!flag)
    {
        if(arr[i] == 1)
        {

```

```

        arr[i] = 0;
    }
    else
    {
        arr[i] = 1;
        flag = 1;
    }
}
}

void sat_30bit(unsigned int *arr_39, unsigned int **arr_30)
{
    //INPUT : NUMERO BINARIO IN COMPLEMENTO A 2 SU 39 BIT ; OUTPUT : NUMERO BINARIO IN
    //COMPLEMENTO A 2 SU 30 BIT
    //CON EVENTUALE SATURAZIONE
    char saturo = 0;
    if(arr_39[38] == 1)
    {
        //controllo saturo negativamente
        for(int i = 29; i < 38 && !saturo; i++)
        {
            if(arr_39[i] == 0)
            {
                //allora è saturo
                saturo = 1;
            }
        }
        if(saturo)
        {
            arr_30[29] = 1;
            for(int i = 0; i < 29; i++)
            {
                arr_30[i] = 0;
            }
        }
    }
}

```

```

else
{
    arr_30[29] = arr_39[38];
    for(int i = 0; i < 29; i++)
    {
        arr_30[i] = arr_39[i];
    }
}
else
{
    // controllo saturazione positiva
    for(int i = 29; i < 38 && !saturo; i++)
    {
        if(arr_39[i] == 1)
        {
            //allora è saturo
            saturo = 1;
        }
    }
    if(saturo)
    {
        arr_30[29] = 0;
        for(int i = 0; i < 29; i++)
        {
            arr_30[i] = 1;
        }
    }
    else
    {
        arr_30[29] = arr_39[38];
        for(int i = 0; i < 29; i++)
        {
            arr_30[i] = arr_39[i];

```

```

    }
}
}
}

int main()
{
    FILE *fp;

    unsigned int inputs[256] = {0};    //vettore contenente i 256 inputs
    long long out_dec[10] = {0};    //vettore contenente gli output in formato DECIMALE NON SATURO
    int w[256][30] = {0};    //matrice contenente i 256 pesi in binario
    int out_bin[10][39] = {0};    //vettore contenente gli output in formato BINARIO NON SATURO
    int out_bin_sat[10][30] = {0};    //vettore contenente gli output in formato BINARIO SATURO
    int buff[30] = {0};    //buffer per array di bit
    int buff_39bits[39] = {0};

    unsigned int inc_data = rand()%(UPPLIM - LOWLIM + 1) + LOWLIM;
    inputs[0] = inc_data;

    fp = fopen("rand_data.txt", "w");
    //stampa del primo pixel
    fprintf(fp, "rst <= '1';\nns <= '0';\nninc_data <= \"");
    printn_bin_file(inc_data, buff, fp);

    fprintf(fp, "\";    -- PIXEL 0 : %d\nnclock <= '1';\nwait for 10 ns;\nclock <= '0';\nwait for 10 ns;\nclock <=
'1';\nwait for 5 ns;\nrst <= '0';\nwait for 5 ns;\nclock <= '0';\nwait for 5 ns;\ns <= '1';\nwait for 5
ns;\nclock<= '1';\nwait for 10 ns;\nclock <= '0';\nwait for 10 ns;\nclock<= '1';\nwait for 10 ns;\n", inc_data);
    fprintf(fp, "\n\n-- INIZIO PROCESSO CARICAMENTO DATI\n");
    //CREAZIONE ALTRI 255 INPUT
    for(int n = 1; n < 256; n++)
    {
        fprintf(fp, "clock <= '0';\n");
        fprintf(fp, "WAIT FOR 5 ns;\n");
        fprintf(fp, "inc_data <= \"");

        inc_data = rand() << 8 | rand()%(UPPLIM - LOWLIM + 1) + LOWLIM;
        inputs[n] = inc_data;
    }
}

```

```

    printn_bin_file(inc_data,buff,fp);

    fprintf(fp,"\\";  -- PIXEL %d : %d\\nWAIT FOR 5 ns;\\n", n, inc_data);
    fprintf(fp,"clock <= '1';\\nWAIT FOR 10 ns;\\n");
}
//CREAZIONE 256 PESI

fprintf(fp,"\\n\\n\\n\\n-- CARICAMENTO DEI PESI");
fprintf(fp, "\\n\\n\\n\\n");

for(int n = 0; n < 256;n++)
{
    fprintf(fp, "clock <= '0';\\n");
    fprintf(fp, "WAIT FOR 5 ns;\\n");
    fprintf(fp, "inc_data <= \\");

    inc_data = (rand() << 14) | rand();
    dectobin(inc_data, buff);
    //PESI RELATIVI AL PRIMO OUTPUT = -3 PER ASSICURARE SATURAZIONE
    for(int i = 0;i < 3;i++)
    {
        buff[i] = 1;
    }
    //caricamento dell'input in matrice di pesi, BINARIO
    for(int i = 0;i < 30;i++)
    {
        w[n][i] = buff[i];
    }
    pr_arr_file(buff,fp);

    fprintf(fp,"\\";  -- PESO %d\\nWAIT FOR 5 ns;\\n", n);
    fprintf(fp,"clock <= '1';\\nWAIT FOR 10 ns;\\n");
}

```

```

//calcolo dell'OUTPUT DECIMALE
for(int i = 0;i < 10;i++)
{
    long long buff_out = 0;
    int weight[3] = {0};          //sottovettore del peso
    for(int j = 0;j < 256;j++)
    {
        weight[0] = w[j][3*i+0];
        weight[1] = w[j][3*i+1];
        weight[2] = w[j][3*i+2];
        int p = dec_weight(weight);
        unsigned int n = inputs[j];
        buff_out += (long long) n * p;
    }
    out_dec[i] = buff_out;
}

fprintf(fp, "\n\n-- RISULTATI ASPETTATI\n");
for(int i = 0;i < 10;i++)
{
    fprintf(fp, "-- OUTPUT [%d] DECIMALE: %lld ; BINARIO : ", i, out_dec[i]);
    dec_to_2s_39bit(out_dec[i], buff_39bits);
    //stampa dell'output in binario COMPLEMENTO A 2
    for(int j = 38;j >= 0;j--)
    {
        fprintf(fp, "%d", buff_39bits[j]);
    }
    fprintf(fp, " ; BINARIO SAT : ");
    sat_30bit(buff_39bits, buff);
    for(int j = 29;j >= 0;j--)
    {
        fprintf(fp, "%d", buff[j]);
    }
    fprintf(fp, "\n");
}

```



```

}
printf("FILE : \"rand_data.txt\" AGGIORNATO CORRETTAMENTE");
fprintf(fp, "\n\nfor i in 0 to 160000 loop\n'clock <= '0';\nwait for 10 ns;\n'clock <= '1';\nwait for 10
ns;\nend loop;\n");
fprintf(fp, "end process;\n");
fprintf(fp, "datapath : neural_network port map(CLK => clock, DATA_IN => inc_data, DATA_OUT =>
d_out, START => s, DONE_OUT => stop, RESET => rst);\nend behavior;\n");
fclose(fp);
return 0;
}

```

# Allegato B – Files VHDL

## 1. *neural\_network.vhd*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity neural_network is
port(CLK : in std_logic;
      DATA_IN : in unsigned(29 downto 0); --dato in ingresso di tipo unsigned
      DATA_OUT : out signed(29 downto 0); --dati in uscita di tipo signed
      START : in std_logic; --segnale che da inizio al caricamento dei dati di ingresso
      nella memoria
      DONE_OUT : out std_logic; --segnale in uscita che segna il completamento del
      caricamento dei dati nella memoria di uscita
      RESET : in std_logic); --segnale di reset
end neural_network;

architecture behavior of neural_network is

--dichiarazione componenti

component add_sub_39 is
port (a,b: in std_logic_vector(38 downto 0);
      sel : in std_logic;
      s : out std_logic_vector(38 downto 0));
end component;

component mux_3b_10to1 is
port( in_0, in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, in_9: in std_logic_vector(2 downto 0);
      sel: in std_logic_vector(3 downto 0);
      out_mux: out std_logic_vector( 2 downto 0));
end component;

component mux_9b_2to1 is
port( in_0, in_1: in std_logic_vector (8 downto 0);
      sel: in std_logic;
      out_mux: out std_logic_vector( 8 downto 0));
end component;

component mux_39b_2to1 is
port( in_0: in std_logic_vector (38 downto 0); -- Input che viene da mem_a, a cui si devono
concaternare 9 '0'
      in_1: in std_logic_vector (38 downto 0); -- 39 '0'
      sel: in std_logic;
```

```

        out_mux: out std_logic_vector( 38 downto 0)
    );
end component;

```

```

component mux_30b_4to1 is
    port( in_0, in_1, in_2, in_3: in std_logic_vector (29 downto 0);
          sel: in std_logic_vector(1 downto 0);
          out_mux: out std_logic_vector(29 downto 0)
    );
end component;

```

```

component n_counter is
    generic (N : integer := 4);
    port    (clock, enable, reset : in std_logic;
             n_out : buffer std_logic_vector(N-1 downto 0));
end component;

```

```

component RAM10x30 is
    port(      Address: in std_logic_vector(3 downto 0);
             D_in: in std_logic_vector(29 downto 0);
             D_out : out signed(29 downto 0);
             WE,RE,CS,Clk: in std_logic);
end component;

```

```

component RAM512x30 is
    port(      Address: in std_logic_vector(8 downto 0);
             D_in: in unsigned(29 downto 0);
             D_out : out std_logic_vector(29 downto 0);
             WE,RE,CS,Clk: in std_logic);
end component;

```

```

component mux2to1 is
    port ( w0, w1, s : in std_logic;
          f : out std_logic);
end component;

```

```

component shiftn is
    generic (n : integer := 30);
    port( r: in std_logic_vector( n-1 downto 0);
          clk, ld_en, sh_en, reset: in std_logic;
          q: buffer std_logic_vector( n-1 downto 0)
    );
end component;

```

--dichiarazione stati dell' ASM

```

type state is (idle, data_load, data_req, load_weight, load_pixel, set_add, set_sub, add_null_1,
add_1_pixel, add_null_2, add_2_pixel, load_sum, load_min_B, load_max_B, load_part_B, count_2_incr,
done);

```

```

signal present_state : state;
signal next_state : state;

--segnali interni
signal mem_a_out , sat_out, sat_in: std_logic_vector(29 downto 0);
signal part_out, part_in, fact_in, rp_out_39bit: std_logic_vector(38 downto 0);
signal cnt_1, mux_cnt1_out : std_logic_vector(8 downto 0);
signal b : std_logic_vector(2 downto 0 );
signal sat_sel : std_logic_vector (1 downto 0);
signal cnt_2, w_sel : std_logic_vector ( 3 downto 0);
signal cnt1_en, cnt1_rst, cnt1_sel, cnt2_en, cnt2_rst, rd_A, wr_A, cs_A, rd_B, wr_B, cs_B, rw_ld, rp_ld,
rw_rst, rp_rst, rp_sh, fact_sel, add_sub, part_rst, part_ld, msb_sat, sat_check, sat_check_in,
mux_sat_check_sel : std_logic;
signal cnt_1_neg : std_logic_vector(8 downto 0);
signal rw_out, rp_out : std_logic_vector(29 downto 0);
signal sat_check_cntrl_1, sat_check_cntrl_2, cnt1_511_check, s: std_logic;

begin

s <= START;

state_registers: process (CLK, RESET)                                -- update degli stati tramite clock
begin
if RESET = '1' then
present_state <= idle;
elsif (CLK' event and CLK = '1')then
present_state <= next_state;
end if;
end process;

state_transitions : process(s, b, cnt_1, cnt_2, msb_sat, sat_check, present_state)    --
transizione da uno stato all'altro
begin
case present_state is
when idle => case s is
when ('1') => next_state <= data_load;
when others => next_state <= idle;
end case;

when data_load => case cnt_1 is
when ("11111111") => next_state <= data_req ;
when others => next_state <= data_load;
end case;

when data_req => next_state <= load_weight;
when load_weight => next_state <= load_pixel;
when load_pixel => if b(2) = '0' then next_state <= set_add ; else next_state <= set_sub; end
if;

when set_add => if b(0) = '0' then next_state <= add_null_1 ; else next_state <=
add_1_pixel; end if;

```

```

        when set_sub => if b(0) = '0' then next_state <= add_null_1 ; else next_state <=
add_1_pixel; end if;
        when add_null_1 => if b(1) = '0' then next_state <= add_null_2 ; else next_state <=
add_2_pixel; end if;
        when add_1_pixel => if b(1) = '0' then next_state <= add_null_2 ; else next_state <=
add_2_pixel; end if;
        when add_null_2 => next_state <= load_sum;
        when add_2_pixel => next_state <= load_sum;
        when load_sum => if cnt_1 = "100000000" then
                                if msb_sat = '1' then
                                    if sat_check = '1' then
                                        next_state <=
load_min_B;
                                else next_state <=
load_part_B;
                                end if;
                                else
                                    if sat_check = '1' then
                                        next_state <=
load_max_B;
                                else next_state <=
load_part_B;
                                end if;
                                end if;
                                else next_state <= data_req; end if;
        when load_min_B => if cnt_2 = "1001" then next_state <= done; else next_state <=
count_2_incr; end if;
        when load_max_B => if cnt_2 = "1001" then next_state <= done; else next_state <=
count_2_incr; end if;
        when load_part_B => if cnt_2 = "1001" then next_state <= done; else next_state <=
count_2_incr; end if;
        when count_2_incr => next_state <= data_req;
        when done => if s = '1' then next_state <= done; else next_state <= idle; end if;
    end case;
end process;

```

state\_outputs: process(present\_state) --comandi di controllo sui segnali eseguiti nei vari stati  
begin

```

    case present_state is
        when idle =>
            rw_rst <= '1';
            part_rst <= '1';
            rp_rst <= '1';
            cnt1_rst <= '1';
            cnt2_rst <= '1';
            cs_A <= '1';
            cs_B <= '0';

```

```

rd_A <= '0';
cnt1_en <= '0';
DONE_OUT <= '0';
cnt1_sel <= '0';
cnt2_en <= '0';
wr_A <= '1';
rd_B <= '0';
wr_B <= '1';
rw_ld <= '0';
rp_ld <= '0';
rp_sh <= '0';
fact_sel <= '0';
add_sub <= '0';
part_ld <= '0';
sat_sel <= "00";
w_sel <= "0000";
when data_load =>
    cnt1_en <= '1';
    cnt1_sel <= '0';
    cs_A <= '1';
    wr_A <= '0';
    cnt1_rst <= '0';
when data_req =>
    cs_A <= '1';
    rd_A <= '1';
    cnt1_sel <= '1';
    cnt2_rst <= '0';
    cnt2_en <= '0';
    rp_rst <= '0';
    rw_rst <= '0';
    wr_A <= '1';
    cnt1_rst <= '0';
    cnt1_en <= '0';
    sat_sel <= "00";
    part_ld <= '0';
    part_rst <= '0';
when load_weight =>
    part_rst <= '0';
    part_ld <= '0';
    rp_ld <= '0';
    cnt1_sel <= '0';
    rw_ld <= '1';
    w_sel <= cnt_2;
when load_pixel =>
    rw_ld <= '0';
    rp_sh <= '0';
    cnt1_sel <= '1';
    rp_ld <= '1';

```

```

        cnt1_en <= '1';
when set_add =>
    cs_A <= '0';
    add_sub <= '0';
    rp_ld <= '0';
    cnt1_en <= '0';
when set_sub =>
    cs_A <= '0';
    add_sub <= '1';
    cnt1_en <= '0';
    rp_ld <= '0';
when add_null_1 =>
    part_ld <= '1';
    fact_sel <= '0';
    rp_sh <= '1';
when add_null_2 =>
    fact_sel <= '0';
    rp_sh <= '0';
when add_1_pixel =>
    part_ld <= '1';
    fact_sel <= '1';
    rp_sh <= '1';
when add_2_pixel=>
    fact_sel <= '1';
    rp_sh <= '0';
when load_sum =>
    part_ld <= '0';
when load_min_B =>
    cs_B <= '1';
    wr_B <= '0';
    cnt1_rst <= '1';
    cnt2_rst <= '0';
    sat_sel <= "11";
when load_max_B =>
    cs_B <= '1';
    wr_B <= '0';
    cnt1_rst <= '1';
    cnt2_rst <= '0';
    sat_sel <= "10";
when load_part_B =>
    cs_B <= '1';
    wr_B <= '0';
    cnt1_rst <= '1';
    cnt2_rst <= '0';
    sat_sel <= "00";
when done =>
    DONE_OUT <= '1';
    wr_B <= '1';

```

```

        cnt2_en <= '0';
        cnt2_rst <= '1';
    when count_2_incr =>
        cnt2_en <= '1';
        wr_B <= '1';
        cs_B <= '0';
        part_rst <= '1';

    end case;
end process;

--costruzione datapath

--counter
counter_1 : n_counter generic map(N => 9) port map(clock => CLK, enable => cnt1_en, reset => cnt1_rst,
n_out => cnt_1);
counter_2 : n_counter generic map(N => 4) port map(clock => CLK, enable => cnt2_en, reset => cnt2_rst,
n_out => cnt_2);

--check counter
cnt1_511_check <= cnt_1(0) and cnt_1(1) and cnt_1(2) and cnt_1(3) and cnt_1(4) and cnt_1(5) and cnt_1(6)
and cnt_1(7) and cnt_1(8);

--negazione del nono bit
cnt_1_neg(7 downto 0) <= cnt_1(7 downto 0);
cnt_1_neg(8) <= not cnt_1(8);
mux_cnt1: mux_9b_2to1 port map (in_0 => cnt_1, in_1 => cnt_1_neg, sel => cnt1_sel, out_mux =>
mux_cnt1_out);

--memorie
mem_a : RAM512x30 port map(Address => mux_cnt1_out, D_in => DATA_IN , D_out => mem_a_out, WE =>
wr_A, RE => rd_A, CS => cs_A, Clk => CLK);
mem_b : RAM10x30 port map(Address => cnt_2, D_in => sat_out, D_out => DATA_OUT, WE => wr_B, RE =>
rd_B, CS => cs_B, Clk => CLK);

--shift register
rw: shiftn port map(r => mem_a_out, clk => CLK, ld_en => rw_ld, sh_en => '0', reset => rw_rst, q => rw_out);
rp: shiftn port map(r => mem_a_out, clk => CLK, ld_en => rp_ld, sh_en => rp_sh, reset => rp_rst, q
=>rp_out);
sum_part : shiftn generic map( N => 39) port map(r => part_in, clk => CLK, ld_en => part_ld, sh_en => '0',
reset => part_rst, q => part_out);

--saturatore

sat_in <= part_out(38) & part_out(28 downto 0); --numero di 30 bit in uscita dal saturatore se non c'è
saturazione

```



```

mux_sat : mux_30b_4to1 port map(in_0 => sat_in, in_1 => "0000000000000000000000000000", in_2 =>
"01111111111111111111111111111111", in_3 => "1000000000000000000000000000", sel => sat_sel,
out_mux => sat_out);

msb_sat <= part_out(38);
mux_sat_check_sel <= part_out(38);

sat_check_cntrl_1 <= ( part_out(37) or part_out(36) or part_out(35) or part_out(34) or part_out(33) or
part_out(32) or part_out(31) or part_out(30) or part_out(29)); --saturazione positivi
sat_check_cntrl_2 <= ( part_out(37) and part_out(36) and part_out(35) and part_out(34) and part_out(33)
and part_out(32) and part_out(31) and part_out(30) and part_out(29));      --saturazione negativi

mux_sat_check : mux2to1 port map (sat_check_cntrl_1, sat_check_cntrl_2, mux_sat_check_sel,
sat_check_in);
sat_check <= msb_sat xor sat_check_in;

-- weight mux

mux_weight : mux_3b_10to1 port map(in_0 => rw_out(2 downto 0), in_1 => rw_out(5 downto 3), in_2 =>
rw_out(8 downto 6), in_3 => rw_out(11 downto 9), in_4 => rw_out(14 downto 12), in_5 => rw_out(17
downto 15), in_6 => rw_out(20 downto 18), in_7 => rw_out(23 downto 21), in_8 => rw_out(26 downto 24),
in_9 => rw_out(29 downto 27), sel => w_sel, out_mux => b);

--add/sub
rp_out_39bit <= "0000000000" & rp_out;
mux_add_sub : mux_39b_2to1 port map(in_0 => "000000000000000000000000000000000000", in_1
=>rp_out_39bit, sel => fact_sel, out_mux => fact_in);
adder_subtractor : add_sub_39 port map(a => part_out, b => fact_in, sel => add_sub, s => part_in);

end behavior;

```

## 2. *add\_sub\_39.vhd*

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Add_sub senza overflow bit

entity add_sub_39 is
port (a,b: in std_logic_vector(38 downto 0);
      sel : in std_logic;
      s : out std_logic_vector(38 downto 0));
end add_sub_39;

architecture behavior of add_sub_39 is

```

```

component full_adder is
port (n,m,ci : in std_logic;
      s, co : out std_logic);
end component;
signal cout : std_logic;
signal cin : std_logic_vector(38 downto 1);
signal j: std_logic_vector(38 downto 0);
begin

j <= b xor (b'range => sel);

F0: full_adder port map(n=>a(0),m=>j(0),ci=>sel,s=>s(0),co=>cin(1));

gen: for i in 1 to 37 generate
F: full_adder port map(n=>a(i),m=>j(i),ci=>cin(i),s=>s(i),co=>cin(i+1));
end generate gen;

F38: full_adder port map(n=>a(38),m=>j(38),ci=>cin(38),s=>s(38),co=>cout);

end behavior;

```

### **3. *full\_adder.vhd***

```

library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
port (n,m,ci : in std_logic;
      s, co : out std_logic);
end full_adder;

architecture behavior of full_adder is

component mux2to1 is
port ( w0, w1, s : in std_logic;
      f : out std_logic);
end component;
signal p : std_logic;

begin

p <= n xor m;
L1: mux2to1 port map(w0=>m,w1=>ci,s=>p,f=>co);
s <= ci xor p;

end behavior;

```

#### **4. mux\_3bit\_10to1.vhd**

```
library ieee;
use ieee.std_logic_1164.all;

-- mux di 3 bit 10to1

entity mux_3b_10to1 is
    port( in_0, in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, in_9: in std_logic_vector (2 downto 0);
          sel: in std_logic_vector(3 downto 0);
          out_mux: out std_logic_vector( 2 downto 0)
        );
end mux_3b_10to1;

architecture behaviour of mux_3b_10to1 is

begin
    --logica del multiplexer
    out_mux <= in_0 when (sel = "0000")
               else in_1 when (sel = "0001")
               else in_2 when (sel = "0010")
               else in_3 when (sel = "0011")
               else in_4 when (sel = "0100")
               else in_5 when (sel = "0101")
               else in_6 when (sel = "0110")
               else in_7 when (sel = "0111")
               else in_8 when (sel = "1000")
               else in_9;

end behaviour;
```

#### **5. mux\_9bit\_2to1.vhd**

```
library ieee;
use ieee.std_logic_1164.all;

-- mux di 9 bit 2to1

entity mux_9b_2to1 is
    port( in_0, in_1: in std_logic_vector (8 downto 0);
          sel: in std_logic;
          out_mux: out std_logic_vector( 8 downto 0)
        );
end mux_9b_2to1;

architecture behaviour of mux_9b_2to1 is

begin

    out_mux <= in_0 when (sel = '0') else in_1;
```

end behaviour;

## **6. mux\_30bit\_4to1.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

-- mux di 30 bit 4to1 che effettua le operazioni di saturazione

entity mux\_30b\_4to1 is

port( in\_0, in\_1, in\_2, in\_3: in std\_logic\_vector (29 downto 0);  
sel: in std\_logic\_vector(1 downto 0);  
out\_mux: out std\_logic\_vector( 29 downto 0)  
);

end mux\_30b\_4to1;

architecture behaviour of mux\_30b\_4to1 is

begin

out\_mux <= in\_0 when (sel = "00")  
else in\_1 when (sel = "01")  
else in\_2 when (sel = "10")  
else in\_3;

end behaviour;

## **7. mux\_39bit\_2to1.vhd**

library ieee;

use ieee.std\_logic\_1164.all;

-- mux di 39 bit 2to1 che porta l'input di mem\_A da 30 a 39 bit per evitare overflow nelle operazioni successive

entity mux\_39b\_2to1 is

port( in\_0: in std\_logic\_vector (38 downto 0); -- Input che viene da mem\_a  
in\_1: in std\_logic\_vector (38 downto 0); -- 39 '0'  
sel: in std\_logic;  
out\_mux: out std\_logic\_vector( 38 downto 0)  
);

end mux\_39b\_2to1;

architecture behaviour of mux\_39b\_2to1 is

begin

out\_mux <= in\_0 when (sel = '0') else in\_1;

end behaviour;

## **8. mux2to1.vhd**

```

library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
    port ( w0, w1, s : in std_logic;
          f : out std_logic);
end mux2to1;

architecture behavior of mux2to1 is
begin
    f <= (not (s) and w0) or (s and w1);
end behavior;

```

## 9. *n\_counter.vhd*

```

library ieee;
use ieee.std_logic_1164.all;

-- CONTATORE GENERICO DI n bit (con valore di default = 4), che ritorna un vettore rappresentante
-- un numero binario di modulo 2^n (std_logic_vector)
-- L'implementazione consiste nell'uso di n flip flop di tipo T posti in cascata
-- in modo da creare un contatore SINCRONO
entity n_counter is
generic (N : integer := 4);
port    (clock, enable, reset : in std_logic;
          n_out : buffer std_logic_vector(N-1 downto 0));    -- vettore con l'MSB al posto piu a
sinistra
end n_counter;

architecture behavior of n_counter is

-- DICHIARAZIONE T FLIP-FLOP, il componente base da utilizzare
component t_ff is
port (T, clk, clr: in std_logic;
      Q: buffer std_logic
      );
end component;

signal Ti : std_logic_vector(0 to N-1);

begin
-- assegnazione dei primi ingressi al primo flip flop T
T0 : t_ff port map(T => enable,clk => clock,clr => reset,Q => n_out(0));
Ti(0) <= enable;

-- generazione del contatore con n ff
gen : for i in 1 to N-1 generate
Ti(i) <= Ti(i-1) and n_out(i-1);-- assegnazione del toggle di ingresso all' i-esimo ff
T : t_ff port map(T => Ti(i),clk => clock,clr => reset,Q => n_out(i));
end generate;

```

```
end behavior;
```

## **10. *ram10x30.vhd***

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
use IEEE.numeric_std.all;
```

```
-- MEMORIA SRAM 10x30 CON : WE attivo basso, RE attivo alto
```

```
entity RAM10x30 is
```

```
port(
    Address: in std_logic_vector(3 downto 0);
    D_in: in std_logic_vector(29 downto 0);
    D_out : out signed(29 downto 0);
    WE,RE,CS,Clk: in std_logic);
```

```
end RAM10x30;
```

```
architecture behavior of RAM10x30 is
```

```
type ram_array is array (0 to 9) of std_logic_vector(29 downto 0);
```

```
signal mem: ram_array;
```

```
--
```

```
--begin
```

```
--process (clk)
```

```
--    begin
```

```
--    if (clk'event and clk = '1') then
```

```
--        if(WE = '0') then
```

```
--            mem(to_integer(unsigned(Address))) <= D_in;
```

```
--        elsif(RE = '1')then
```

```
--            D_out <= signed(mem(to_integer(unsigned(Address))));
```

```
--        end if;
```

```
--    end if;
```

```
--end process;
```

```
begin
```

```
process (clk, CS)
```

```
    begin
```

```
    if(CS='1') then
```

```
        if (clk'event and clk = '1') then
```

```
            if(WE = '0') then
```

```
                mem(to_integer(unsigned(Address))) <= D_in;
```

```
            elsif(RE = '1') then
```

```
                D_out <= signed(mem(to_integer(unsigned(Address))));
```

```
            end if;
```

```
        end if;
```

```
    end if;
```

```
end process;
```

```
end behavior;
```

## **11. *ram512x30.vhd***

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
use IEEE.numeric_std.all;
```

```
-- MEMORIA SRAM 512x30 CON : WE attivo basso, RE attivo alto
```

```
entity RAM512x30 is
```

```
port(      Address: in std_logic_vector(8 downto 0);
```

```
          D_in: in unsigned(29 downto 0);
```

```
          D_out : out std_logic_vector(29 downto 0);
```

```
          WE,RE,CS,Clk: in std_logic);
```

```
end RAM512x30;
```

```
architecture behavior of RAM512x30 is
```

```
type ram_array is array (0 to 511) of std_logic_vector(29 downto 0);
```

```
signal mem: ram_array;
```

```
begin
```

```
process (clk, CS)
```

```
begin
```

```
  if(CS ='1') then
```

```
    if (clk'event and clk = '1') then
```

```
      if(WE = '0') then
```

```
        mem(to_integer(unsigned(Address))) <= std_logic_vector(D_in);
```

```
      elsif(RE = '1') then
```

```
        D_out <= mem(to_integer(unsigned(Address)));
```

```
      end if;
```

```
    end if;
```

```
  end if;
```

```
end process;
```

```
end behavior;
```

## **12. *shiftn.vhd***

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
-- Shift register a n bit con parallel load e shift enable
```

```
entity shiftn is
```



```
  generic (n : integer := 30);
```

```
  port( r: in std_logic_vector( n-1 downto 0);
```

```

        clk, ld_en, sh_en, reset: in std_logic;
        q: buffer std_logic_vector( n-1 downto 0)
    );

end shiftn;

architecture behaviour of shiftn is
begin
    process (clk) is
    begin
        if(reset = '0') then
            if (clk'event and clk = '1') then
                if ld_en = '1' then      --quando il load  attivo, carica i bit in ingresso al registro
                    q <= r;
                end if;
                if sh_en = '1' then      --quando lo shift  attivo, shifta i bit di una posizione verso
sinistra
                    Genbits: for i in n-2 downto 0 loop
                        q(i+1) <= q(i);
                    end loop;
                    q(0) <= '0';
                end if;
            end if;
        else
            gen_zeros : for i in n-1 downto 0 loop
                q(i) <= '0';
            end loop;
        end if;
    end process;
end behaviour;

```

### 13. *t\_ff.vhd*

```

library ieee;
use ieee.std_logic_1164.all;

-- Toggle FF con clear attivo alto sincrono

```

```

entity t_ff is
port (T, clk, clr: in std_logic;
      Q: buffer std_logic
    );
end t_ff;

```

architecture behaviour of t\_ff is

```

signal Qa : STD_LOGIC ; -- Definisco i segnali interni

```

```

begin

```


```

Q <= Qa;

```



```

process (clk)
begin
    if (clk'event and clk = '1') then
        if (clr = '1') then
            Qa <= '0'; --uscita a 0 se il clear  attivo
        elsif (T = '1') then
            Qa <= not(Qa);
        elsif (T = '0') then
            Qa <= Qa;
        end if;
    end if;
end process;

```

end behaviour;

#### **14. *tb\_neural\_network\_complete.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

```

```

entity tb_neural_network_complete is
end tb_neural_network_complete;

```

architecture behavior of tb\_neural\_network\_complete is

```

component neural_network
port(CLK : in std_logic;
      DATA_IN : in unsigned(29 downto 0);
      DATA_OUT : out signed(29 downto 0);
      START : in std_logic;
      DONE_OUT : out std_logic;
      RESET : in std_logic);
end component;

```

```

signal s,rst, clock, stop : std_logic;
signal inc_data : unsigned(29 downto 0);
signal d_out : signed(29 downto 0);

```

```

begin
incoming_data : process
begin

```

```

rst <= '1';
s <= '0';
inc_data <= "0000000000000000100000000101000"; -- PIXEL 0 : 16424
clock <= '1';
wait for 10 ns;
clock <= '0';

```

```
wait for 10 ns;
clock <= '1';
wait for 5 ns;
rst <= '0';
wait for 5 ns;
clock <= '0';
wait for 5 ns;
s <= '1';
wait for 5 ns;
clock <= '1';
wait for 10 ns;
clock <= '0';
wait for 10 ns;
clock <= '1';
wait for 10 ns;
```

```
-- INIZIO PROCESSO CARICAMENTO DATI
```

```
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010000111101110111101"; -- PIXEL 1 : 4750269
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001111100111011011111"; -- PIXEL 2 : 6803167
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001111010110110011010101"; -- PIXEL 3 : 4025557
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100101110111101010000"; -- PIXEL 4 : 7532368
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111111101011001001000"; -- PIXEL 5 : 6280776
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001101101111101111101111"; -- PIXEL 6 : 7207919
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001000001111111111101000"; -- PIXEL 7 : 4325352
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000000011110101110110010"; -- PIXEL 8 : 125874
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011101111011011011010"; -- PIXEL 9 : 3077850
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000101010111111010000101"; -- PIXEL 10 : 1408645
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110010100111100111101"; -- PIXEL 11 : 3755837
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000000001101100100100011"; -- PIXEL 12 : 55587
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100000101111000001011"; -- PIXEL 13 : 3169803
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010010101110100000100"; -- PIXEL 14 : 4807940
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011011111011101000110"; -- PIXEL 15 : 5109574
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101001111111110110010"; -- PIXEL 16 : 5570482
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011010101011101001100"; -- PIXEL 17 : 2971468
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010011011110110001000001"; -- PIXEL 18 : 5106753
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000011001101111101110001001"; -- PIXEL 19 : 6749065
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001101111011000011101"; -- PIXEL 20 : 2553373
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010111010111101101011000"; -- PIXEL 21 : 6126424
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101100111110100000111"; -- PIXEL 22 : 7765255
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100100111101100100100"; -- PIXEL 23 : 1211172
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111100111111101011011"; -- PIXEL 24 : 1998683
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110101111011111001001"; -- PIXEL 25 : 1767369
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001101011111111110010100"; -- PIXEL 26 : 7077780
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111111111111101000011"; -- PIXEL 27 : 8388419
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100100111101100010010"; -- PIXEL 28 : 3308306
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001100110111110000111"; -- PIXEL 29 : 2518919
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000000110111101000011011"; -- PIXEL 30 : 227867
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010111101111110101100"; -- PIXEL 31 : 778156
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001110111001100011111"; -- PIXEL 32 : 488223
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101011111101101001111"; -- PIXEL 33 : 7732047
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010001011101111100111110"; -- PIXEL 34 : 2289470
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000101110000111101100110100"; -- PIXEL 35 : 5798708
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111001111111100010001"; -- PIXEL 36 : 6094609
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110100101111100110000"; -- PIXEL 37 : 1728304
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000111011111111010011101"; -- PIXEL 38 : 3931805
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001111001011111101000111"; -- PIXEL 39 : 7962439
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000011011101110010101011"; -- PIXEL 40 : 908459
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100010101111100010010"; -- PIXEL 41 : 3235602
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011011111101101000010"; -- PIXEL 42 : 5110594
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011100101001101100101"; -- PIXEL 43 : 3036005
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111001111011001101010"; -- PIXEL 44 : 1898090
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001101100011000101110"; -- PIXEL 45 : 6735406
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001111101111011100110000"; -- PIXEL 46 : 8320816
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000101100011111110100000"; -- PIXEL 47 : 2916256
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001010100011111011110101"; -- PIXEL 48 : 5537525
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010000111101110001111"; -- PIXEL 49 : 555919
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001000000110111111100000"; -- PIXEL 50 : 4251616
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110011101101100011110"; -- PIXEL 51 : 7985950
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100111101101010101110"; -- PIXEL 52 : 7592622
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001101111111010011000"; -- PIXEL 53 : 2555544
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```



```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010010111101110110111"; -- PIXEL 54 : 621495
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101110111001110011100"; -- PIXEL 55 : 5731228
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100000110100100101010"; -- PIXEL 56 : 7366954
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010101101100001111101"; -- PIXEL 57 : 4905085
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000101101110110110010111"; -- PIXEL 58 : 1502615
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001111001101011111101000"; -- PIXEL 59 : 3987432
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010000001101110110110000"; -- PIXEL 60 : 4251056
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100111110101111001000"; -- PIXEL 61 : 3402696
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010001101111101010001"; -- PIXEL 62 : 4775761
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011000001111111101100101"; -- PIXEL 63 : 6356837
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001111001101111110111110"; -- PIXEL 64 : 3989438
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011110111111011010100"; -- PIXEL 65 : 3112660
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001000111111000101011"; -- PIXEL 66 : 294443
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101001111111010001101"; -- PIXEL 67 : 5570189
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000001101011111110000001"; -- PIXEL 68 : 884609
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101011110111101010101"; -- PIXEL 69 : 7728981
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011000111110101100000"; -- PIXEL 70 : 2915680
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000101111111111101100111"; -- PIXEL 71 : 3145575
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000101000111111101100000"; -- PIXEL 72 : 2686816
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001000101111110111001111"; -- PIXEL 73 : 2293199
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001100101111010011011"; -- PIXEL 74 : 2514587
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010010001101110100010101"; -- PIXEL 75 : 4775189
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000011000010111101101110000"; -- PIXEL 76 : 6388592
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100101110011000011011"; -- PIXEL 77 : 3335707
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100011111001110000011"; -- PIXEL 78 : 7467907
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111110100111101001000"; -- PIXEL 79 : 8343368
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000001100111011100000000"; -- PIXEL 80 : 423680
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110001111111111100110"; -- PIXEL 81 : 1638374
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100000111110100101010"; -- PIXEL 82 : 5274922
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110010111101111001001"; -- PIXEL 83 : 1670089
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000011111100111100010001"; -- PIXEL 84 : 1036049
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111110111111000110010"; -- PIXEL 85 : 6258226
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110000111010010011101"; -- PIXEL 86 : 7894173
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010110101110111110011"; -- PIXEL 87 : 2842099
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111011111111111010010"; -- PIXEL 88 : 6160338
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110101101111111010010"; -- PIXEL 89 : 5955538
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111111110010001011000"; -- PIXEL 90 : 6284376
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100100111111000110100"; -- PIXEL 91 : 1211956
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000001111110111100110000"; -- PIXEL 92 : 519984
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011010111101011110011"; -- PIXEL 93 : 7174899
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000011101100111011001101"; -- PIXEL 94 : 970445
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000000111011111110001111"; -- PIXEL 95 : 122767
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000001110100110110101000110"; -- PIXEL 96 : 3829062
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000101011111010111110001101"; -- PIXEL 97 : 5756813
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000100101110111111100101"; -- PIXEL 98 : 620517
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110011101100100011011"; -- PIXEL 99 : 1693979
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010010101111111111100100"; -- PIXEL 100 : 2457572
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111011100100111110101"; -- PIXEL 101 : 1952245
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010001000111101101110110"; -- PIXEL 102 : 4488054
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000101001100000110000101"; -- PIXEL 103 : 1360261
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110110101110100001100"; -- PIXEL 104 : 8084748
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101100101111101001111"; -- PIXEL 105 : 7757647
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010110101011011010011"; -- PIXEL 106 : 2840275
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111110111101110001100"; -- PIXEL 107 : 8354700
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111111111111001111010"; -- PIXEL 108 : 8388218
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100000100110100010100"; -- PIXEL 109 : 5262612
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110000111011100111001"; -- PIXEL 110 : 3700537
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001100111001110000000"; -- PIXEL 111 : 422784
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001010101111100010111"; -- PIXEL 112 : 2449175
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011000100111010010010001"; -- PIXEL 113 : 6452369
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110011101101001100010"; -- PIXEL 114 : 1694306
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011010111110111001101"; -- PIXEL 115 : 5078477
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110111111110110000011"; -- PIXEL 116 : 3931523
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100010111111101100111"; -- PIXEL 117 : 5341031
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```



```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001101010010111111110110"; -- PIXEL 118 : 6971382
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111000100111001101000"; -- PIXEL 119 : 6049384
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100000101011110010101"; -- PIXEL 120 : 3168149
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111100111011100001100"; -- PIXEL 121 : 6190860
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100111101111100010101"; -- PIXEL 122 : 7593749
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000110000110111101100101"; -- PIXEL 123 : 1601381
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110010110101011010010"; -- PIXEL 124 : 5860050
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011001111111101001001"; -- PIXEL 125 : 2948937
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010100101111011001110"; -- PIXEL 126 : 679630
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011100101111011001110"; -- PIXEL 127 : 5136230
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110000111011011111000"; -- PIXEL 128 : 5797624
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100110101111110111001"; -- PIXEL 129 : 1269689
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011110111010110101011"; -- PIXEL 130 : 7304619
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011101111111100100010"; -- PIXEL 131 : 5177122
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001011101110101110111"; -- PIXEL 132 : 384375
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111001111111001010001"; -- PIXEL 133 : 8191569
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010001111111100100100"; -- PIXEL 134 : 4783908
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000101100100111111100100"; -- PIXEL 135 : 1462244
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000011011110111110011110010"; -- PIXEL 136 : 7306482
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111110101011111010010"; -- PIXEL 137 : 6248402
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010011101101000100111"; -- PIXEL 138 : 2742823
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010011101111000001001"; -- PIXEL 139 : 646665
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011010001111010111000011"; -- PIXEL 140 : 6878659
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110010111010001011000"; -- PIXEL 141 : 3765336
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000100110011111110010110"; -- PIXEL 142 : 2523030
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010000000111011110001001"; -- PIXEL 143 : 4224905
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010010101111101110100"; -- PIXEL 144 : 2711412
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010000110110111001110"; -- PIXEL 145 : 2649550
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110101110111111000111"; -- PIXEL 146 : 8056775
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111001110111101100111"; -- PIXEL 147 : 6090599
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110001101010001001000"; -- PIXEL 148 : 7918664
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010000110111001110111"; -- PIXEL 149 : 552567
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010000110110111100000110"; -- PIXEL 150 : 4419334
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110100110100100111111"; -- PIXEL 151 : 8022335
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100000101011110101111"; -- PIXEL 152 : 7362479
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010010111111101101001"; -- PIXEL 153 : 2719593
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010000001111110100010000"; -- PIXEL 154 : 4259088
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001010111110110111111"; -- PIXEL 155 : 2457023
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010001101101110011010100"; -- PIXEL 156 : 4644052
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000011101111111100001010"; -- PIXEL 157 : 982794
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100001111111110010110"; -- PIXEL 158 : 3211158
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001011100110100101101"; -- PIXEL 159 : 6671661
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100001111100100010011"; -- PIXEL 160 : 3209491
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010001000101110110011000"; -- PIXEL 161 : 4480408
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100101101010111000000"; -- PIXEL 162 : 3331520
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110101101111111100000"; -- PIXEL 163 : 5955552
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011111110111001111100"; -- PIXEL 164 : 5238396
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001111101111111110100110"; -- PIXEL 165 : 4128678
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000111111111101010100110"; -- PIXEL 166 : 4192934
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001001100011011000000"; -- PIXEL 167 : 6604480
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011011111111101010010"; -- PIXEL 168 : 3014482
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100001111111110011001"; -- PIXEL 169 : 5308313
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001011111111011111011000"; -- PIXEL 170 : 3143640
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010111110111101111001111"; -- PIXEL 171 : 6257615
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011100101011110001111"; -- PIXEL 172 : 5134223
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010100111111100100111"; -- PIXEL 173 : 2785063
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100011101000111011000"; -- PIXEL 174 : 5362136
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011000110111010011111"; -- PIXEL 175 : 7106207
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011000111111001011100"; -- PIXEL 176 : 5013084
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011010100111111100000"; -- PIXEL 177 : 7163872
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100000111101010011010"; -- PIXEL 178 : 1079962
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011101111111111011000"; -- PIXEL 179 : 5177304
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010000111010110011110"; -- PIXEL 180 : 554398
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011111110101110100100"; -- PIXEL 181 : 5237668
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```



```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001010001110111100001011"; -- PIXEL 182 : 2682635
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001010100111111110110010"; -- PIXEL 183 : 5570482
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001110100011101100011"; -- PIXEL 184 : 6768483
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011100111111000101000"; -- PIXEL 185 : 5144104
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100110101011010001000"; -- PIXEL 186 : 7558792
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010100001111101110000001"; -- PIXEL 187 : 5307265
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001000000111100101111001"; -- PIXEL 188 : 2128249
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011010111111001110111"; -- PIXEL 189 : 7175799
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001101101100101101100001"; -- PIXEL 190 : 3591009
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001001101111011100100100"; -- PIXEL 191 : 2553636
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111001111111111011001"; -- PIXEL 192 : 1900505
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000011100110111101101011"; -- PIXEL 193 : 946027
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100010111111001100001"; -- PIXEL 194 : 1146465
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001001111001010010100"; -- PIXEL 195 : 6615700
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001010101111100110010"; -- PIXEL 196 : 6643506
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011101111111100111000"; -- PIXEL 197 : 5177144
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011011111111100111110"; -- PIXEL 198 : 7208766
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011101101110101011101"; -- PIXEL 199 : 7265629
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000011111111010101101011"; -- PIXEL 200 : 2094443
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111100100111011100010"; -- PIXEL 201 : 8277730
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010100110111100100110"; -- PIXEL 202 : 683814
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001101101110110100011101"; -- PIXEL 203 : 3599645
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001000010111001000011011"; -- PIXEL 204 : 2191899
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111011101101111000001"; -- PIXEL 205 : 1956801
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100000111011100111101"; -- PIXEL 206 : 1079101
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010100100011100000110"; -- PIXEL 207 : 673542
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001111110111100001110"; -- PIXEL 208 : 519950
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101010111011100000100"; -- PIXEL 209 : 7698180
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001101110111110100011001"; -- PIXEL 210 : 3636505
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011010110110110001011111"; -- PIXEL 211 : 7040095
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011010111101100111000011"; -- PIXEL 212 : 7068099
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100101110011100101011"; -- PIXEL 213 : 3335979
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000000000001100111101000100"; -- PIXEL 214 : 53060
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110100111011100000111"; -- PIXEL 215 : 8025863
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000111101101111011110001"; -- PIXEL 216 : 2023153
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010000011111111110001011"; -- PIXEL 217 : 2162571
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110001111111110111101"; -- PIXEL 218 : 7929789
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011100011111001011101010"; -- PIXEL 219 : 7467754
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110000111110111111101"; -- PIXEL 220 : 7896573
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001001110100111001110"; -- PIXEL 221 : 6613454
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000010010011111101110100"; -- PIXEL 222 : 1212276
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100010100011010001011"; -- PIXEL 223 : 3229323
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001101010111111010000110"; -- PIXEL 224 : 3505798
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101000111110101110001"; -- PIXEL 225 : 5537137
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000010000110111111001110"; -- PIXEL 226 : 552910
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101001111111010000000"; -- PIXEL 227 : 5570176
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001100110111111111011001"; -- PIXEL 228 : 6750169
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010101111101001001000101"; -- PIXEL 229 : 5755461
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110000101110100101001"; -- PIXEL 230 : 5791017
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011000111100111111111001"; -- PIXEL 231 : 6541305
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011110111001001111101"; -- PIXEL 232 : 7303805
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010110100111101011101111"; -- PIXEL 233 : 5929711
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000100011011111111101010010"; -- PIXEL 234 : 4652882
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000001111110011100110001"; -- PIXEL 235 : 517937
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100101111110110101000"; -- PIXEL 236 : 1244584
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011110111111101100111"; -- PIXEL 237 : 5209959
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011101100111111100001010"; -- PIXEL 238 : 7765770
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000001011000111111111111011"; -- PIXEL 239 : 5832699
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000100111111111111001001"; -- PIXEL 240 : 1310665
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010000101111111101111011"; -- PIXEL 241 : 4390779
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000000011011111111110000111"; -- PIXEL 242 : 917383
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110001100101001000000"; -- PIXEL 243 : 3721792
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011001000101111111111100"; -- PIXEL 244 : 6578172
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111001111101101001101"; -- PIXEL 245 : 8190797
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```



```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011011110111101011110101"; -- PIXEL 246 : 7305973
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001110100111001001111010"; -- PIXEL 247 : 3830394
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011000000101111010011000"; -- PIXEL 248 : 6315672
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000001100111110111111010010"; -- PIXEL 249 : 3403730
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111110100110111101111"; -- PIXEL 250 : 8343023
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000000010000001011110000111001"; -- PIXEL 251 : 2120761
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00000000011111111011110100101"; -- PIXEL 252 : 2095013
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010011110111011101010001"; -- PIXEL 253 : 5207889
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011110000111101100001010"; -- PIXEL 254 : 7895818
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011000111101101110011111"; -- PIXEL 255 : 6544287
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

```

-- CARICAMENTO DEI PESI

```

clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101000001000111010001011111"; -- PESO 0
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110100100110111010111101111"; -- PESO 1
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101010000001101001101001111"; -- PESO 2
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100010111001100100011100111"; -- PESO 3
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101100000010111100010110111"; -- PESO 4
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001010011000100101000001111"; -- PESO 5
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001001010101010000010101111"; -- PESO 6
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100101100111110110011010111"; -- PESO 7
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001001011000110010111001111"; -- PESO 8
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110010010011100100011111111"; -- PESO 9
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100011101100100101110011111"; -- PESO 10
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010010001010001100101001111"; -- PESO 11
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100111111010101001001111111"; -- PESO 12
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110100010011100110101011111"; -- PESO 13
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110010100001011100001111111"; -- PESO 14
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101111001110011001011000111"; -- PESO 15
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011010101111100100101111001111"; -- PESO 16
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110011000110111100110000111"; -- PESO 17
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100000010000011001000100111"; -- PESO 18
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011111100110010001011110111111"; -- PESO 19
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100101010011100100110000111"; -- PESO 20
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110001001010101000001001111"; -- PESO 21
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110101011000000011001011111"; -- PESO 22
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001001000110110000101001111"; -- PESO 23
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101000010110111001010011111"; -- PESO 24
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110101110011101110100111111"; -- PESO 25
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100101011000111001001100111"; -- PESO 26
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100101101111111001110110111"; -- PESO 27
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001111000000011000110101111"; -- PESO 28
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010010000100010110101000111"; -- PESO 29
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111011110100101111101100111"; -- PESO 30
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111100100000101111010100111"; -- PESO 31
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100011101010101100110011111"; -- PESO 32
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001000101110010101011100011111"; -- PESO 33
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101001001111110111100010111"; -- PESO 34
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000100100001000111000000111"; -- PESO 35
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101000010011001100111111111"; -- PESO 36
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000100100110101001111010111"; -- PESO 37
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0011110000010000101100000000111"; -- PESO 38
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010111001001100101000101111"; -- PESO 39
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0000100110110011000011110001111"; -- PESO 40
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011010111001110011101010111"; -- PESO 41
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110010100000001001001000111"; -- PESO 42
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100101000101000100010101111"; -- PESO 43
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101100111111101110011001111"; -- PESO 44
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110010110111101100011010111"; -- PESO 45
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010011101100111101001111010111"; -- PESO 46
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001011010000100110010010111"; -- PESO 47
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001011100111000011001011111"; -- PESO 48
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011111001011111011101101100111"; -- PESO 49
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0111110000100111011110100101111"; -- PESO 50
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0100001111110111101011100000111"; -- PESO 51
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110001010110110111110100111"; -- PESO 52
WAIT FOR 5 ns;

```



```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100011011111000011001100111"; -- PESO 53
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001111010011001010110110111"; -- PESO 54
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001111100100110011111000111"; -- PESO 55
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001000110100101101100110111"; -- PESO 56
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010010011010011111000001111"; -- PESO 57
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000010010110011100001001111"; -- PESO 58
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000010000000100110110001111"; -- PESO 59
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000010001011111001101010111"; -- PESO 60
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011001100011110101111011111"; -- PESO 61
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110001001110100111111001111"; -- PESO 62
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011111111010011000011011011111"; -- PESO 63
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010000000101110111010001111"; -- PESO 64
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100011000000110011100111111"; -- PESO 65
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001000110000100001101101111"; -- PESO 66
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010011100110100011001000010111"; -- PESO 67
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000100110001000001111111111"; -- PESO 68
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110100101010110110111010111"; -- PESO 69
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101111110011111001101110111"; -- PESO 70
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111010001011100111001001111"; -- PESO 71
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001110111100111000000001111"; -- PESO 72
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010101001001100110000101111"; -- PESO 73
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010100001110110010110000111"; -- PESO 74
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110110010100100101110011111"; -- PESO 75
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010110111001100100011011111"; -- PESO 76
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011010000111010101011000011111"; -- PESO 77
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010111111101000000101001111"; -- PESO 78
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001000100011011110101110111111"; -- PESO 79
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000111011011010110001010110111"; -- PESO 80
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011111001010110111101010111"; -- PESO 81
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011101111010101110011011111"; -- PESO 82
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100001000000111001001101111"; -- PESO 83
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101110110110100010010001111"; -- PESO 84
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100110111010110001111011111"; -- PESO 85
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110111000101101100101000111"; -- PESO 86
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100110110010110011000010111"; -- PESO 87
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110100000100110110010100111"; -- PESO 88
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100011111011101000001111111"; -- PESO 89
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101111011110100111011110111"; -- PESO 90
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001000011110101110000000010111"; -- PESO 91
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111011011100110010011100111"; -- PESO 92
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101011110000101101101100111"; -- PESO 93
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111011000111100101001101111"; -- PESO 94
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001110011110110110101111111"; -- PESO 95
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011111111010010011101011111"; -- PESO 96
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010010110001011110010001111"; -- PESO 97
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000011000101001100010001111"; -- PESO 98
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111010010011111111101011111"; -- PESO 99
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000011111011101001011100111"; -- PESO 100
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110001000011101011001001111"; -- PESO 101
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010010011101110001100110111"; -- PESO 102
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101101001101110101101100111"; -- PESO 103
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001110100000010111110000111"; -- PESO 104
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000111100011001110111000111"; -- PESO 105
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100011011000101110110100111"; -- PESO 106
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110100011000111110001001111"; -- PESO 107
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001000001000000001011101111"; -- PESO 108
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000100111010101001101101111"; -- PESO 109
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001100000001100000100101111"; -- PESO 110
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011001001111101001001101111"; -- PESO 111
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010111110111111010100010111"; -- PESO 112
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101010101000000101110011111"; -- PESO 113
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001011100100101100110000111"; -- PESO 114
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010110100001101010000101111"; -- PESO 115
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100000010100100010010001111"; -- PESO 116
WAIT FOR 5 ns;

```



```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010001111011000011011101111"; -- PESO 117
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110111001111010000101000111"; -- PESO 118
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101101010000110111010000111"; -- PESO 119
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111100100100000101100110111"; -- PESO 120
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010000110110101101011001111"; -- PESO 121
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000111010010110111010011111"; -- PESO 122
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0110110011011111010010110000111"; -- PESO 123
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011011101010001101001010001111"; -- PESO 124
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111111001100110101000010111"; -- PESO 125
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000011111111110111110011111"; -- PESO 126
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101000100001111010011001111"; -- PESO 127
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000101100111111100001010111"; -- PESO 128
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011011100011110110101010111"; -- PESO 129
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011010010101100111011010111111"; -- PESO 130
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101011011110100000000011111"; -- PESO 131
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101000010011110001100000111"; -- PESO 132
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010000010100000101101111111"; -- PESO 133
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100010101110101011110001111"; -- PESO 134
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001110101111101001110111111"; -- PESO 135
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100100100100100011000110111"; -- PESO 136
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011000100111000001001100111"; -- PESO 137
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011111010011110101001011100111"; -- PESO 138
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001101101111111010100000111"; -- PESO 139
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100011001100011011101000111"; -- PESO 140
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000101011001001111100001111"; -- PESO 141
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100100010101101001110110111"; -- PESO 142
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001000001010000001011101111"; -- PESO 143
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011010011011101101000110111111"; -- PESO 144
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100010010111100011001110111"; -- PESO 145
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011111011110001011101101010111"; -- PESO 146
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00011011011111110100110011111111"; -- PESO 147
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001000001110110101111011111"; -- PESO 148
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111011000010100110111010111"; -- PESO 149
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011010001111111111000000111"; -- PESO 150
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000001011111001001101010111"; -- PESO 151
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110100110101101101100010111"; -- PESO 152
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "01111111101011110001101000111"; -- PESO 153
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100100010000100111110000111"; -- PESO 154
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110010001100101100000010111"; -- PESO 155
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001010001100100100010011111"; -- PESO 156
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001010010101100011000101111"; -- PESO 157
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110110010111001111110001111"; -- PESO 158
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "01000101111110011111100001111"; -- PESO 159
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110011101110101010110111111"; -- PESO 160
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000011111000100111111001111"; -- PESO 161
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110010000111110011000111111"; -- PESO 162
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111110110110111111010010111"; -- PESO 163
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110100011000101000010010111"; -- PESO 164
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101000111010100101110101111"; -- PESO 165
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101011000110100000010110111"; -- PESO 166
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111101010010011100110010111"; -- PESO 167
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100011101010111100001011111"; -- PESO 168
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101110110101010000001000111"; -- PESO 169
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110011111110101110110000111"; -- PESO 170
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011110000111011101000000000111"; -- PESO 171
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001010100101101001000010111"; -- PESO 172
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001000110101100011100110111"; -- PESO 173
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001011011000111100100100111"; -- PESO 174
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001111101101111111101111111"; -- PESO 175
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100000101101100110011101111"; -- PESO 176
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001100111001001011000011111"; -- PESO 177
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010101001000001010101010001111"; -- PESO 178
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010000111000101001110101111"; -- PESO 179
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101001000110011011010111111"; -- PESO 180
WAIT FOR 5 ns;

```



```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000111000101111101011011011111"; -- PESO 181
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101010100111011111110101111"; -- PESO 182
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000001111101110011010111111"; -- PESO 183
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101111011000000000010010111"; -- PESO 184
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001010110111010011000000011111"; -- PESO 185
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100000111010110100111100111"; -- PESO 186
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111011101110111001000010111"; -- PESO 187
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100100101100101001101000111"; -- PESO 188
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010000011101101101110111110111"; -- PESO 189
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010111100101101111101010000111"; -- PESO 190
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110111010000010111001111111"; -- PESO 191
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011100101110111001111001111"; -- PESO 192
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011010010001110111101000110111"; -- PESO 193
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000011011110100110111101001111"; -- PESO 194
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000010111010000010100000111"; -- PESO 195
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001101010101010111011111010111"; -- PESO 196
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010001010110001010100011111"; -- PESO 197
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110110000101100111011110111"; -- PESO 198
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010010111010100100111100100111"; -- PESO 199
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010011110010001100101001000111"; -- PESO 200
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110000101101101110110100111"; -- PESO 201
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101010011001101010110111111"; -- PESO 202
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001100011010110011110001111"; -- PESO 203
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001100000110100111111110101111"; -- PESO 204
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000101110000101101000101111"; -- PESO 205
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101011101010111000110110111"; -- PESO 206
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101110011001100101001011111"; -- PESO 207
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110010000101111101011100111"; -- PESO 208
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110000101101011000011011111"; -- PESO 209
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001101110100100111101110111"; -- PESO 210
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110010101110100010111101111"; -- PESO 211
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000101011010110000101101111"; -- PESO 212
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100001100010100000000111111"; -- PESO 213
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001110010110110011011010010111"; -- PESO 214
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001101001100011101100001111"; -- PESO 215
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "0011111101010111011111111101111"; -- PESO 216
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000111101000011100011110011111"; -- PESO 217
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010111101111111011000000111"; -- PESO 218
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100100110100110110110110111"; -- PESO 219
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001111010001011111011010111"; -- PESO 220
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100001110011101111110100111"; -- PESO 221
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000001100011010000010111111"; -- PESO 222
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101000111011110010101011111"; -- PESO 223
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110011000110100011011000111"; -- PESO 224
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001101000111100001011010111"; -- PESO 225
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011001010010101101101000100111"; -- PESO 226
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000000110001010101110110101111"; -- PESO 227
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010001101101010111010100000111"; -- PESO 228
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100000001000110011000000111"; -- PESO 229
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001001110001100011001010001111"; -- PESO 230
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111011010111000011010111111"; -- PESO 231
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100110111001100000000011111"; -- PESO 232
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011000100110000110000011001111"; -- PESO 233
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000111000010000001001111001111"; -- PESO 234
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001110101110101010111010111"; -- PESO 235
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001000010111110110001000111"; -- PESO 236
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010110010101001110011000101111"; -- PESO 237
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "00110111110101111110110101111"; -- PESO 238
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010100010101100110010111001111"; -- PESO 239
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001111010000110110111111101111"; -- PESO 240
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000010100010011000110011101111"; -- PESO 241
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000100100101010101001001001111"; -- PESO 242
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011101011000000010011110000111"; -- PESO 243
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011111101100110100000101111"; -- PESO 244
WAIT FOR 5 ns;

```



```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000001011111100110100000110111"; -- PESO 245
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000111101011100111001011101111"; -- PESO 246
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001000001101010110001001111111"; -- PESO 247
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000110101010000001110101011111"; -- PESO 248
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011011000100111110111111111111"; -- PESO 249
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "011100011001111110110000100111"; -- PESO 250
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010011011111110000010100111111"; -- PESO 251
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101100011011001011110000111"; -- PESO 252
WAIT FOR 5 ns;

```

```

clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "001011011000111011100110000111"; -- PESO 253
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "000101111011000111001001111111"; -- PESO 254
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;
clock <= '0';
WAIT FOR 5 ns;
inc_data <= "010011111011111100010111101111"; -- PESO 255
WAIT FOR 5 ns;
clock <= '1';
WAIT FOR 10 ns;

-- RISULTATI ASPETTATI
-- OUTPUT [0] DECIMALE: -3313633014 ; BINARIO : 111111100111010011111011111100100001001 ;
BINARIO SAT : 10000000000000000000000000000000
-- OUTPUT [1] DECIMALE: 227505515 ; BINARIO : 000000000001101100011110111010101101011 ;
BINARIO SAT : 001101100011110111010101101011
-- OUTPUT [2] DECIMALE: -90485492 ; BINARIO : 111111111111010100110110100110100001011 ;
BINARIO SAT : 111010100110110100110100001011
-- OUTPUT [3] DECIMALE: 17391591 ; BINARIO : 000000000000000100001001010111111100111 ; BINARIO
SAT : 000001000010010101111111100111
-- OUTPUT [4] DECIMALE: -655909989 ; BINARIO : 11111111011000111001111001101110011011 ;
BINARIO SAT : 10000000000000000000000000000000
-- OUTPUT [5] DECIMALE: 10315675 ; BINARIO : 0000000000000000100111010110011110011011 ; BINARIO
SAT : 000000100111010110011110011011
-- OUTPUT [6] DECIMALE: -47052419 ; BINARIO : 1111111111111010011001000001001011111101 ;
BINARIO SAT : 111101001100100000100101111101
-- OUTPUT [7] DECIMALE: 21111397 ; BINARIO : 0000000000000001010000100010001001100101 ; BINARIO
SAT : 000001010000100010001001100101
-- OUTPUT [8] DECIMALE: -130387868 ; BINARIO : 111111111111000001110100111000001100011 ;
BINARIO SAT : 111000001110100111000001100011
-- OUTPUT [9] DECIMALE: 1669432466 ; BINARIO : 000000001100011100000011000010010010010 ;
BINARIO SAT : 01111111111111111111111111111111

for i in 0 to 160000 loop
clock <= '0';
wait for 10 ns;

```

```
clock <= '1';  
wait for 10 ns;  
end loop;  
end process;  
datapath : neural_network port map(CLK => clock, DATA_IN => inc_data, DATA_OUT => d_out, START => s,  
DONE_OUT => stop, RESET => rst);  
end behavior;
```