# Relazione Prova finale di Reti Logiche

Francesco Caracciolo

Codice Persona: 10796701 Matricola: 980989

## Contents

# 1 Introduzione

## 1.1 Obiettivo del progetto

Il progetto consiste nello sviluppare un componente hardware nel linguaggio VHDL che ha il compito di scansionare una sequenza di misurazioni presenti in memoria ed associare ad ogni misurazione un relativo valore di confidenza.
Le misurazioni con valore 0 hanno il significato di ”valore non specificato”, fanno quindi calare la confidenza della misurazione.

## 1.2 Specifica del funzionamento

Ogni misurazione W può assumere un valore compreso tra 0 e 255 (quindi ha dimensione di 8 bit, ovvero 1 byte, in memoria).
Nell'indirizzo di memoria successivo a ogni misurazione è presente un byte che il componente deve completare con il valore di confidenza C.

- Ogni volta che si incontra una misurazione diversa da 0, la confidenza viene impostata a 31.
- Ogni volta che si incontra una misurazione uguale a 0, la confidenza, se maggiore di 0, viene decrementata.

Inoltre, il componente deve rimpiazzare le misurazioni non specificate con l'ultima misurazione valida. Se la sequenza inizia con valori non specificati (0), allora la confidenza parte da 0 e considera 0 tutti i valori.
In seguito un esempio:
Sequenza Iniziale:
[177, 0, 109, 0, 249, 0, 0, 0, 0, 0, 0, 102, 0, 0, 0]
Sequenza finale:
[177, 31, 109, 31, 249, 31, 249, 30, 249, 29, 249, 28, 102, 31, 102, 30]

## 1.3 Specifica hardware

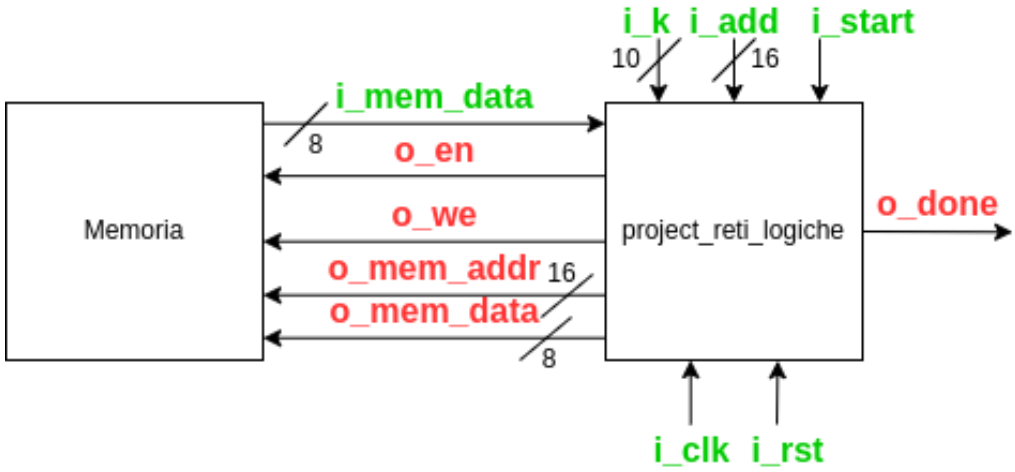


Figure 1: Rappresentazione schematica degli ingressi e delle uscite del componente.

Ingressi

- i_clk: segnale di clock;
- i_rst: segnale di reset;
- i_add: vettore da 16 bit, indirizzo di memoria a partire da cui inizia la sequenza;
- i_k: vettore da 10 bit, numero di misurazioni da elaborare nella sequenza;
- i_start: segnale di inizio della computazione;
- i_mem_data: vettore da 8 bit, contenuto della memoria all'indirizzo richiesto in precedenza;

Uscite

- o_done: alto quando l'elaborazione è stata completata e non è stato ancora dato il segnale di start;
- o_en: abilita la memoria;
- o_we: abilita la scrittura in memoria;
- o_mem_addr: vettore da 16 bit, indica l'indirizzo di memoria su cui scrivere/leggere;
- o_mem_data: vettore da 8 bit, indica il valore da scrivere in memoria se o_en è abilitato;

Al componente verrà sempre dato un segnale di reset prima della prima elaborazione, inoltre deve essere in grado di gestire più elaborazioni senza un segnale di reset intermedio.

# 2 Architettura

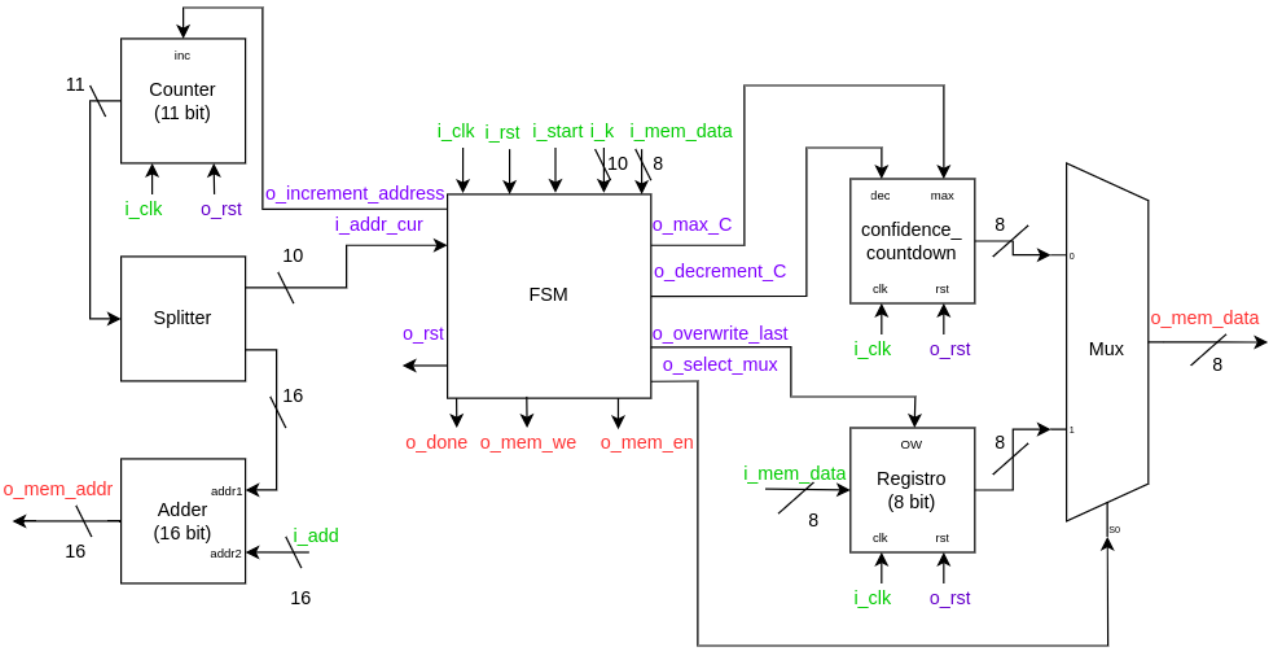


Figure 2: Struttura della rete
In rosso sono rappresentate le uscite principali del componente.
In verde sono rappresentate le entrate principali del componenete.
In viola sono rappresentati i segnali della macchina a stati.

Il sistema è composto da 7 componenti, in seguito analizzati approfonditamente:

- FSM: Macchina a stati finiti che si occupa di gestire i segnali degli altri componenti e della gestione dello stato;
- confidence_countdown: Un semplice decrementatore che serve per la gestione della confidenza. Ha un ingresso MAX che imposta il valore contenuto a 31, mentre DEC decrementa il valore fino allo 0;
- reg_8: un registro a 8 bit che serve a mantenere l'ultima misurazione valida;
- multiplexer: sceglie cosa scrivere in memoria, con sel a 0 scrive la confidenza, con sel a 1 scrive l'ultima misurazione utile;
- counter: un contatore a 11 bit che fa da cursore per l'indirizzo di memoria su cui si sta operando attualmente;
- splitter: questo componente divide il segnale risultante dal counter in due segnali:
  - un segnale a 16 bit che è semplicemente l'output del counter esteso;
  - un segnale a 10 bit che è utilizzato dalla macchina a stati e indica quante misurazioni sono state lette (cursore/2);
- adder: un semplice adder senza carry, ha il ruolo di calcolare l'indirizzo di memoria su cui si sta operando sommando al valore i_add, il valore del cursore;

## 2.1 Confidence Countdown

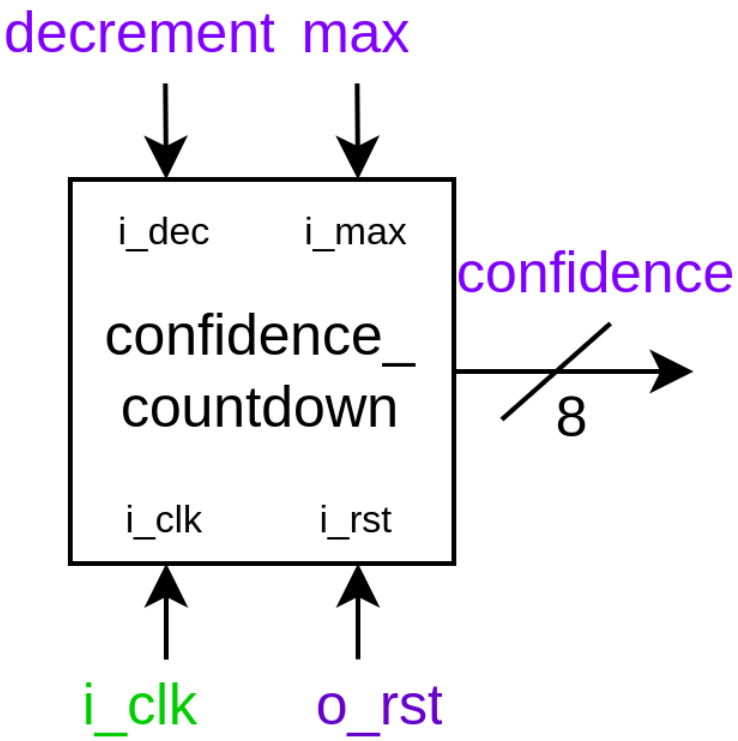


Figure 3: Rappresentazione schematica degli ingressi e delle uscite del componente.

Confidence Countdown è un componente sincrono che ha lo scopo di salvare il valore della confidenza durante l'esecuzione, portarlo al valore massimo (31) in un ciclo di clock quando necessario, e decrementarlo.

### 2.1.1 Interfaccia

- i_clk: segnale di clock;
- i_rst: segnale di reset, porta il valore contenuto nel componente a 0;
- i_dec: decrementa il valore contenuto nel componente;
- i_max: imposta il valore contenuto nel componente a 31;
- o_C: uscita del componente a 8 bit;

Il segnale di reset porta il contenuto 0 dato che all'inizio dell'esecuzione, se non viene letta alcuna misurazione valida, la confidenza è nulla.

### 2.1.2 Architettura

```
architecture confindence_countdown_arch of confidence_countdown is
signal stored_value : std_logic_vector(4 downto 0) := "00000";
begin
  o_C <= "000" & stored_value;
  process (i_clk, i_rst, i_max)
  begin
      if i_rst = '1' then
          stored_value <= (others => '0');
      elsif rising_edge(i_clk) then
        if i_max = '1' then
            stored_value <= (others => '1');
        elsif i_dec = '1' then
            if stored_value /= "00000" then
                stored_value <= stored_value - 1;
            end if;
        end if;
    end if;
  end process;
end confindence_countdown_arch;
```

L'architettura del componente è costituita da un processo in cui il segnale di reset è asincrono, mentre gli altri segnali sono sincroni con il clock.
Il segnale stored_value è a 5 bit dato che questo componente deve elaborare solo valori tra 0 e 31, viene poi esteso a 8 bit in modo da essere scritto in memoria (8 bit) più agevolmente.
Quando il segnale i_max è alto, il valore viene portato a 31, mentre quando i_max è basso e i_dec è alto, il valore viene decrementato di uno.
Il valore salvato viene sempre dato in output.

## 2.2 Registro a 8 bit

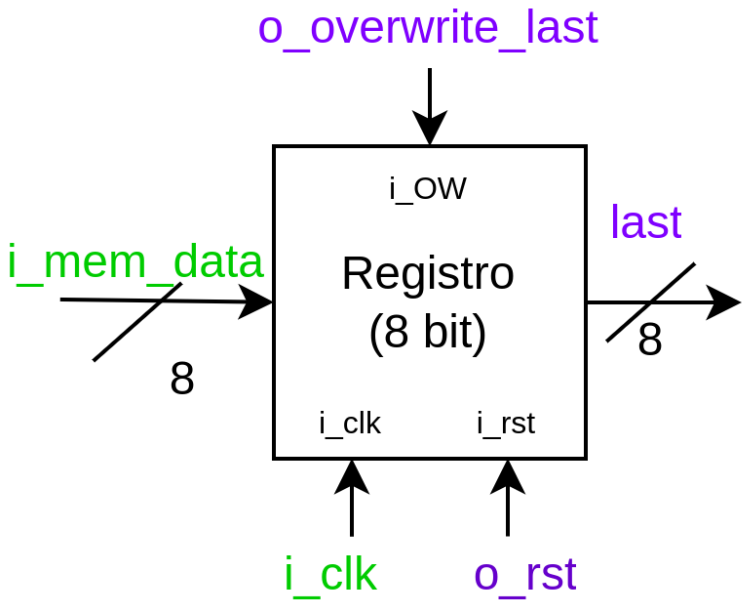


Figure 4: Rappresentazione schematica degli ingressi e delle uscite del componente.

Questo componente (reg_8) è un semplice registro sincrono a 8 bit che ha il ruolo di salvare l'ultima misurazione valida.

### 2.2.1 Interfaccia

- i_clk: segnale di clock;
- i_rst: segnale di reset, asincrono, porta il valore contenuto nel registro a 0;
- i_overwrite: se alto sovrascrive il valore contenuto nel registro con il valore di i_value;
- i_value: valore a 8 bit da inserire nel registro se i_overwrite è alto;
- output: uscita del registro a 8 bit con l'ultimo valore salvato;

Dato che in questo componente vengono salvati solo dati in memoria, i_value è direttamente collegato all'entrata principale i_mem_data. L'architettura di questo componente è omessa perché banale.

## 2.3 Multiplexer

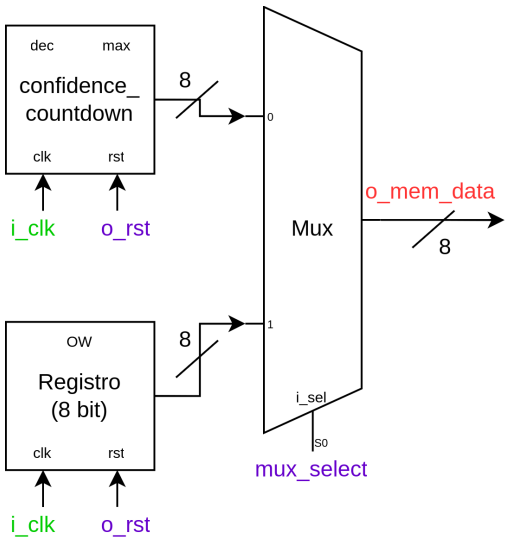


Figure 5: Rappresentazione schematica degli ingressi e delle uscite del componente nel contesto in cui si trova.

Questo componente è un semplice multiplexer da due ingressi a 8 bit. Il suo ruolo è quello di gestire cosa viene scritto in memoria. Se sel è 0 viene scritta la confidenza (da confidence_countdown), mentre se sel è 1 viene scritta l'ultima misurazione valida contenuta in reg_8.

## 2.4 Counter

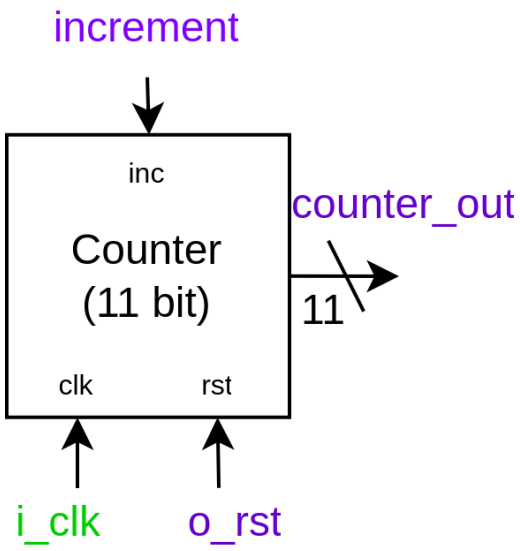


Figure 6: Rappresentazione schematica degli ingressi e delle uscite del componente.

Questo componente è un semplice contatore sincrono a 11 bit che ha il compito di conteggiare quanti indirizzi di memoria sono stati elaborati e di fungere da cursore per l'indirizzo di memoria

su cui si sta operando.
L'output del contatore viene sfruttato in due modi:

- Viene aggiunto all'ingresso i_add per stabilire su quale indirizzo si stia operando e costituire l'uscita o_mem_addr.
- I 10 bit più significativi (l'output numerico del contatore è quindi diviso per due) sono sfruttati dalla macchina a stati finiti per stabilire quante misurazioni sono state elaborate ed eventualmente terminare la computazione.

### 2.4.1 Interfaccia

- i_clk: segnale di clock;
- i_rst: segnale di reset, porta il valore del contatore a 0;
- i_increment: se alto incrementa di 1 il valore del contatore per ogni ciclo di clock;
- output: uscita del contatore;

### 2.4.2 Architettura

```
1  architecture counter_arch of counter is
2  signal stored_value: std_logic_vector(10 downto 0) :=
       "00000000000";
3  begin
4      output <= stored_value;
5      process (i_clk, i_rst)
6      begin
7      if i_rst = '1' then
8          stored_value <= (others => '0');
9      elsif rising_edge(i_clk) then
10         if increment = '1' then
11             stored_value <= stored_value + 1;
12         end if;
13     end if;
14     end process;
15 end counter_arch;
```

L'architettura del componente è costituita da un processo in cui il segnale di reset è asincrono, mentre gli altri segnali sono sincroni con il clock.
Il segnale stored_value è un vettore da 11 bit.
In uscita viene sempre dato il valore di stored_value.
Quando il reset è 1, indipendentemente dal clock, il contenuto di stored_value viene impostato a 0, dato che al reset del componente, non è stato ancora letto alcun indirizzo di memoria.
Sul fronte di salita del clock, se increment è alto, il valore contenuto all'interno di stored_value viene incrementato, altrimenti rimane invariato.

## 2.5 Splitter

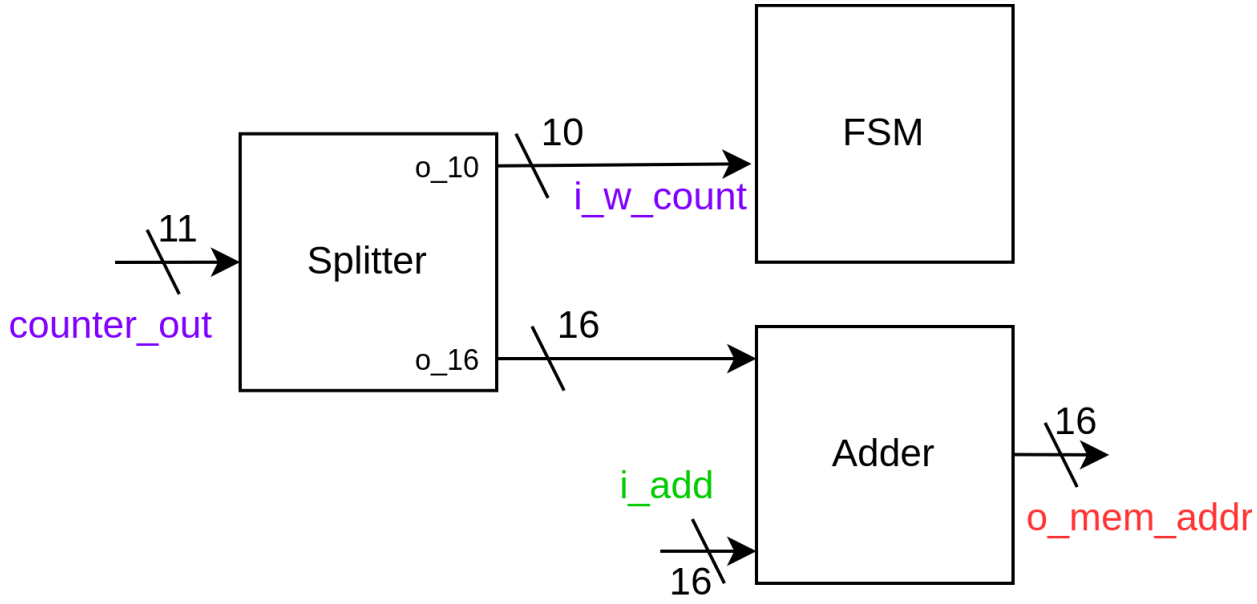


Figure 7: Rappresentazione schematica degli ingressi e delle uscite del componente nel contesto in cui si trova (FSM semplificata)

Lo splitter è un componente asincrono che da un segnale in input a 11 bit, lo divide in due segnali in output:

- Un segnale a 10 bit composto dai primi 10 bit del vettore in ingresso. Ha l'utilità pratica di dividere il valore in ingresso per due (arrotondato per difetto).
  All'interno della rete logica ha il compito di "informare" la macchina a stati finiti di quante misurazioni sono state lette in memoria.
- Un segnale a 16 bit che è l'estensione del segnale a 11 bit in ingresso, in modo da essere compatibile con l'adder.

Nota: questo componente non è strettamente necessario per la rete, un modo alternativo per ottenere lo stesso risultato è quello di collegare nel port mapping solo una parte del vettore di bit counter_out alla macchina a stati ed estendere il valore in entrata per l'adder.
Ho optato per questa soluzione poiché la ritenevo più chiara all'interno della rappresentazione e più modulare in caso di modifiche all'architettura.

### 2.5.1 Interfaccia

- input: segnale a 11 bit in input;
- o_10: segnale a 10 bit in output contenente i 10 bit più significativi del segnale d'ingresso;
- o_16: segnale a 16 bit in output che è un'estensione del valore d'ingresso a 16 bit;

### 2.5.2 Architettura

```
architecture splitter_arch of splitter is

begin
    o_10 <= input(10 downto 1);
    o_16 <= "00000" & input;
end splitter_arch;
```

L'uscita o_10 è data dai primi 10 bit dell'input.
L'uscita o_16 è ottenuta concatenando l'input ad un vettore di bit contenente quattro zeri.

## 2.6 Adder

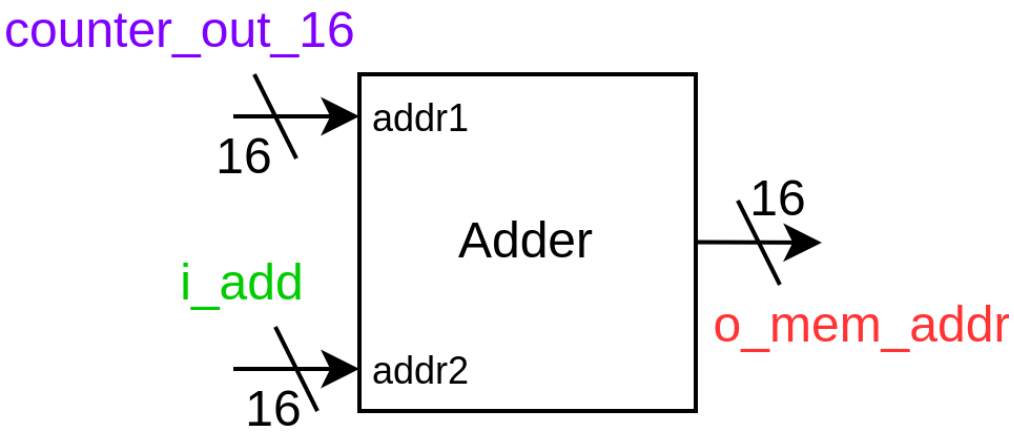


Figure 8: Rappresentazione schematica degli ingressi e delle uscite del componente.

Questo componente ha il compito di sommare due numeri a 16 bit senza segno e senza carry. Nella rete logica, l'adder ha il ruolo di sommare all'indirizzo di memoria da cui inizia la sequenza ( segnale i_add), il cursore di memoria, ovvero l'output esteso del counter, restituendo quindi l'indirizzo su cui operare (o_mem_addr).
Il carry non serve in questo caso dato che nessun indirizzo di memoria è espresso con più di 16 bit. Nel caso in cui venga data una combinazione di segnali tali che $i\_add + i\_k * 2$ sia maggiore del massimo indirizzo di memoria, si sta violando la specifica e il comportamento del componente project_reti_logiche non è specificato.

### 2.6.1 Interfaccia

- addr1: ingresso, vettore da 16 bit
- addr2: ingresso, vettore da 16 bit
- out: uscita, somma a 16 bit senza segno di addr1 e addr2 senza carry

## 2.7 Macchina a stati

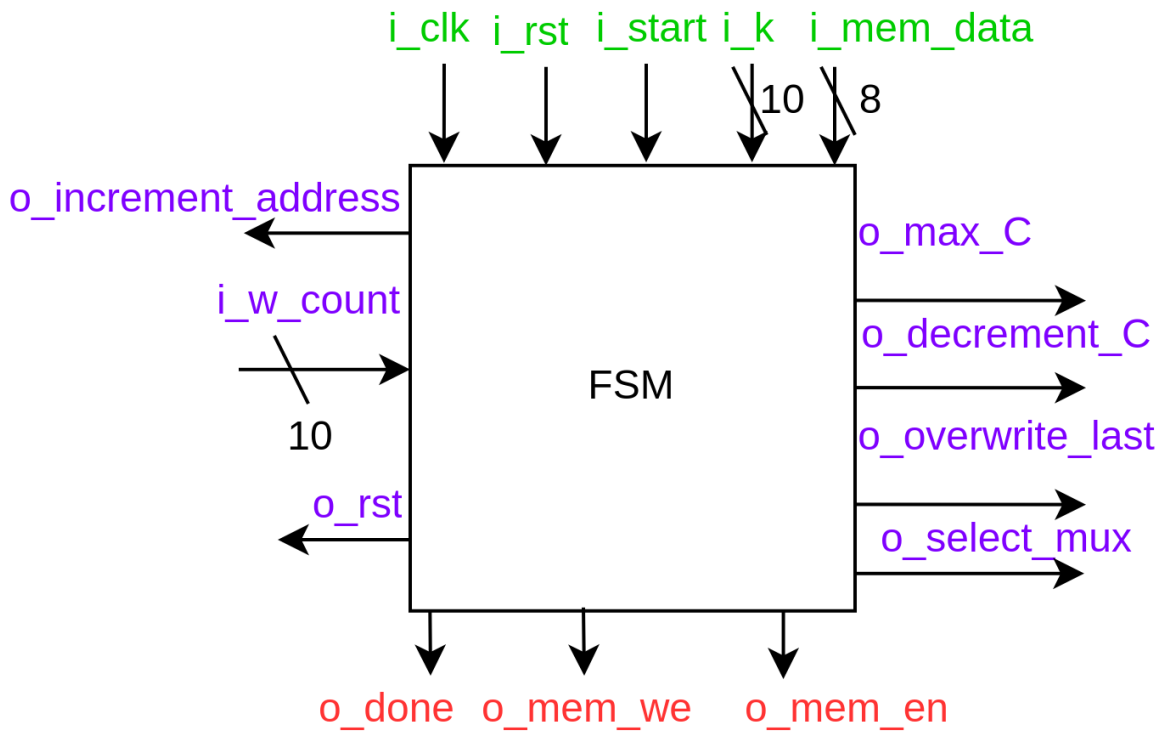


Figure 9: Rappresentazione schematica degli ingressi e delle uscite della macchina a stati.
In verde sono rappresentate le entrate principali
In rosso sono rappresentate le uscite principali
In viola sono rappresentati i segnali interni al componente

La macchina a stati ha il compito di gestire in comportamento sequenziale del componente e di tutti i moduli di cui è composto.
La rete è stata progettata in modo che l'FSM si occupi solo di alzare/abbassare segnali e di fare confronti.

### 2.7.1 Interfaccia

- i_clk, i_rst, i_start, i_k, i_mem_data sono gli ingressi principali del componente project_reti_logic
- o_done, o_mem_we, o_mem_en sono le uscite principali del compontente project_reti_logiche
- i_w_count è un vettore da 10 bit in ingresso fornito dal contatore che indica quante parole sono state elaborate fino a quel momento. Serve ad essere confrontato con i_k per stabilire quando l'esecuzione è terminata.
- o_rst è un segnale interno che viene dato ai componenti che necessitano di reset. La motivazione per cui i moduli non usano il segnale i_rst principale è perché a fine esecuzione è necessario mandare un segnale di reset ai vari componenti per permetterne il corretto funzionamento in caso di una seconda esecuzione senza un segnale di reset esterno.
- o_max_C è un segnale che viene dato al compontente confidence_countdown per impostare il massimo valore di confidenza (33)
- o_decrement_C è un segnale che viene dato al componente confidence_countdown per decrementare il valore di confidenza
- o_overwrite_last è un segnale che viene dato al registro a 8 bit per indicare di sovrascrivere l'ultima misurazione valida salvata nel registro
- o_select_mux sceglie cosa scrivere in memoria (quando o_mem_we e o_mem_en sono alti). Vale 0 per scrivere la confidenza, vale 1 per scrivere l'ultima misurazione valida contenuta nel registro.
- o_increment_address è un segnale dato al counter per incrementare il cursore dell'indirizzo di memoria

### 2.7.2 Architettura

La macchina a stati è composta da 9 stati e implementata tramite due processi:

- Un processo che gestisce le transizioni di stato
- Un processo che gestisce i segnali in funzione dello stato in cui si trova

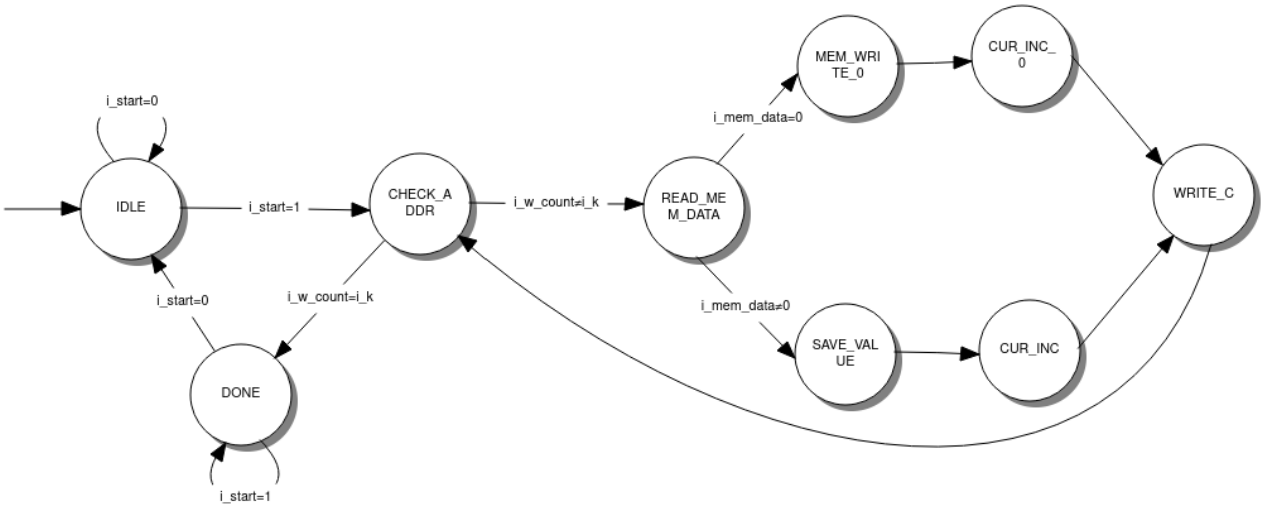


Figure 10: Diagramma delle transizioni della macchina a stati

Transizioni degli stati Le transizioni di stato sono sincrone con il clock. In seguito un estratto del codice dell'architettura:

```
architecture fsm_arch of fsm is
    type S is (IDLE, CHECK_ADDR, DONE,READ_MEM_DATA,
                    SAVE_VALUE, CUR_INC, MEM_WRITE,
                    MEM_WRITE_0, CUR_INC_0, WRITE_C);
    signal curr_state : S := IDLE;
begin
    -- State transitions
    process(i_clk, i_rst)
    begin
        if i_rst = '1' then
            curr_state <= IDLE;
        elsif rising_edge(i_clk) then
            case curr_state is
                when IDLE =>
                    if i_start = '1' then
                        curr_state <= CHECK_ADDR;
                    else
                        curr_state <= IDLE;
                    end if;
        [...]
```

Definisce un tipo enumerato S per rappresentare i possibili stati della macchina a stati e un segnale curr_state per salvare lo stato attualmente in esecuzione.
In modo asincrono, il reset imposta lo stato iniziale "IDLE".
Sul fronte di salita del ciclo di clock, invece, avvengono le transizioni di stato in base allo stato attuale e ad eventuali condizioni aggiuntive.
Il diagramma sovrastante indica tutte le altre transizioni di stato.

Gestione dei segnali I segnali sono gestiti da un altro processo che si occupa solo di alzare o abbassare segnali in funzione dello stato corrente.
Per evitare latch, tutti i segnali hanno come valore predefinito 0 all'inizio del processo.

```
process(curr_state)
begin
    o_mem_we <= '0';
    o_mem_en <= '0';
    o_done <= '0';
    o_max_C <= '0';
    o_decrement_C <= '0';
    o_select_mux <= '0';
    o_overwrite_last <= '0';
    o_increment_address <= '0';
    o_rst <= '0';
```

Nota: all'interno del codice riguardante i singoli stati, ho ripetuto l'istruzione che causa lo spegnimento di alcuni segnali per una maggiore leggibilità e sequenzialità di lettura, anche se a livello di codice non hanno alcun effetto.

Analizziamo cosa avviene nei singoli stati.

IDLE Idle ha il solo scopo di aspettare che il segnale i_start diventi 1 per poter proseguire con l'elaborazione. Mentre è in IDLE, la FSM dà il segnale di reset agli altri moduli in modo da permettere la corretta inizializzazione in caso di esecuzioni multiple senza che sia dato il segnale i_rst tra di esse.

```
    case curr_state is
        when IDLE =>
            o_rst <= '1';
```

CHECK_ADDR Questo stato ha lo scopo di controllare se la quantità di misurazioni elaborate (i_w_count) è uguale al valore i_k in modo da stabilire se la computazione deve terminare. Se i due valori sono uguali, l'esecuzione è terminata e passa allo stato DONE, altrimenti l'esecuzione avanza allo stato READ_MEM_DATA.
Viene raggiunto ogni volta che la computazione di una misurazione è terminata.

```
        when CHECK_ADDR =>
            -- Setup memory
            o_mem_we <= '0';
            o_mem_en <= '1';
            -- Fix increments
            o_max_C <= '0';
            -- Stop ad_cur
            o_increment_address <= '0';
```

Lo stato alza il segnale o_mem_en in modo da richiedere la lettura del valore in memoria nello stato successivo READ_MEM_DATA. I segnali abbassati servono ad indicare che le azioni di stati precedenti sono interrotte.

READ_MEM_DATA Questo stato ha il solo scopo di leggere la misurazione in memoria e stabilire se è maggiore di 0. Nel caso in cui sia 0 passa allo stato MEM_WRITE_0, altrimenti passa allo stato SAVE_VALUE.

SAVE_VALUE Questo stato viene raggiunto nel caso in cui si stia analizzando una misurazione maggiore di 0, quindi valida.

```
1            when SAVE_VALUE =>
2                -- Maximize confidence
3                o_max_C <= '1';
4                -- Save W in reg_8
5                o_overwrite_last <= '1';
6                o_mem_en <= '1';
```

In questo stato vengono effettutate due azioni:

1. Viene massimizzata la confidenza con il segnale o_max_C
2. Viene salvata la misurazione nel registro in modo da poterla utilizzare come ultima misurazione valida nel caso in cui venga incontrato uno 0. Per salvare la misurazione abilito il segnale di overwrite e la memoria per permettere la lettura del valore.

CUR_INC Questo stato ha il ruolo di incrementare la posizione del cursore in memoria di uno per raggiungere l'indirizzo su cui successivamente andrà scritto il valore di confidenza.

```
1            when CUR_INC =>
2                -- Stop confidence maximizing
3                o_max_C <= '0';
4                -- Stop W saving
5                o_overwrite_last <= '0';
6                -- Increase memory cursor
7                o_increment_address <= '1';
```

o_increment_address è il segnale che incrementa il counter che si occupa di fare da cursore per la memoria.
I segnali di max e overwrite vengono abbassati dato che non è più necessario massimizzare la confidenza o sovrascrivere il valore nel registro con il valore in memoria.

MEM_WRITE_0 Questo è il primo stato dell'elaborazione nel caso in cui i_mem_data = 0.
Lo scopo di questo stato è quello di scrivere l'ultima misurazione valida in memoria e decrementare la confidenza.

```
1            when MEM_WRITE_0 =>
2                -- Write value of register (last W) in memory
3                o_select_mux <= '1';
4                o_mem_we <= '1';
5                o_mem_en <= '1';
6                -- Decrease Confidence
7                o_decrement_C <= '1';
```

o_select_mux a 1 seleziona il valore del registro come valore da scrivere in memoria. I segnali o_mem_we e o_mem_en sono necessari per permette che il valore venga scritto in memoria. Il segnale o_decrement_C alto decrementa il valore di confidenza contenuto nel modulo confidence_countdown di 1.

CUR_INC_0 Questo è il secondo stato dell'elaborazione nel caso in cui i_mem_data = 0. Lo scopo di questo stato è quello di incrementare il cursore dell'indirizzo di memoria in modo da scrivere nello stato successivo la confidenza in memoria.

```
1                when CUR_INC_0 =>
2                    -- Stop writing
3                    o_mem_we <= '0';
4                    o_mem_en <= '0';
5                    -- Increase address cursor
6                    o_increment_address <= '1';
7                    -- Stop confidence decrement
8                    o_decrement_C <= '0';
```

I segnali o_mem_we e o_mem_en vengono abbassati dato che non ci sono operazioni da fare in memoria.
Il segnale o_decrement_C viene abbassato dato che non è più necessario decrementare la confidenza.
o_increment_address viene alzato per aumentare il cursore di memoria e scrivere la confidenza nello stato successivo.

WRITE_C Questo è il terzo e ultimo stato dell'elaborazione della singola misurazione. Lo scopo di questo stato è quello di scrivere il valore della confidenza in memoria ed incrementare il cursore di memoria.

```
1                when WRITE_C =>
2                    -- Write confidence
3                    o_select_mux <= '0';
4                    o_mem_we <= '1';
5                    o_mem_en <= '1';
6                    -- Increase address
7                    o_increment_address <= '1';
```

Il segnale o_select_mux è 0 per indicare che in memoria va scritta la confidenza. I segnali o_mem_we e o_mem_en sono necessari per permettere che il valore venga scritto in memoria. Il segnale o_decrement_C viene abbassato dato che non è più necessario decrementare la confidenza.
o_increment_address viene mantenuto alto per fare in modo che nello stato CHECK_ADDR venga letta dalla memoria la misurazione successiva.

DONE Questo è l'ultimo stato raggiunto prima che la macchina a stati torni in IDLE per attendere una nuova computazione.
Il compito dello stato DONE è quello di impostare a 1 il segnale o_done finchè il segnale i_start non viene abbassato. Successivamente, la macchina torna in IDLE in attesa di una nuova esecuzione.

```
                when DONE =>
                    o_done <= '1';
                    o_mem_en <= '0';
                    o_mem_we <= '0';
```

I segnali o_mem_we e o_mem_en vengono abbassati dato che non ci sono operazioni da fare in memoria.
Il segnale o_done è alzato in attesa di un segnale di start basso.

# 3 Risultati sperimentali

## 3.1 Sintesi

La sintesi del componente è effettuata sulla board Atrix-7 FPGA xc7a200tfbg484-1 e viene completata senza warning.
Utilizzando i tool forniti dalla Tcl console di Vivado, il comando usage_utilization segnala l'utilizzo di 41 Look Up Tables, 33 flip flop e di 0 latch.
Utilizzando il comando report_timing i vincoli sul tempo sono rispettati:

```
Slack (MET) : 16.498ns (required time - arrival time)
```

## 3.2 Simulazioni

Il componente è in grado di eseguire correttamente tutti i test proposti in seguito sia eseguendo la simulazione behavoiral, sia quella post sintesi funzionale.

### 3.2.1 Testbench fornito dai docenti

Il testbench fornito dai docenti controlla che una generica sequenza di X valori sia elaborata correttamente e che il componente risponda bene ai vari segnali dati.

### 3.2.2 Sequenza che inizia con una serie di zeri

Quando una sequenza di misurazioni inizia con uno 0, ovvero una misurazione non valida, la confidenza deve rimanere nulla e come misurazione bisogna riportare lo 0 fino alla prima misurazione valida.
Questo test ha il compito di verficare il corretto funzionamento specifica.
Il caso specifico di test utilizzato è il seguente:
[0, 0, 0, 0, 239, 0, 0, 0, 0, 0, 0]
Risultato:
[0, 0, 0, 0, 239, 31, 239, 30, 239, 29, 239, 28]

### 3.2.3 Caso k=0

Nel caso in cui venga richiesta una computazione in un'area di memoria con k=0, bisogna calcolare la confidenza di 0 misurazioni. La memoria quindi deve rimanere invariata.

### 3.2.4 Caso k = 1023

Dato che k è un valore a 10 bit, il massimo valore che può assumere in decimale è 1023, al componente quindi può essere richiesto un massimo di 1023 misurazioni.
La sequenza soggetta a caso di test è stata generata autmaticamente, con una probabilità del 60% di generare misurazioni nulle, in modo da mettere maggiormente alla prova il funzionamento del componente.
Questo test è utile per verficare che il componente sia in grado di gestire un'elaborazione con una quantità di valori al limite del dominio.

### 3.2.5 Test con 33 0 consecutivi

La confidenza massima è 31, e va decrementata ogni volta che si incontra una misurazione non specificata.
Il test consiste in una generica sequenza, a cui succedono 33 misurazioni a zero consecutive.
Questo test è utile per provare che la confidenza rimane nulla anche in seguito ad una sequenza di zeri.

### 3.2.6 Due elaborazioni consecutive senza reset

Il componente deve essere in grado di eseguire correttamente due elaborazioni anche nel caso in cui non venga dato il segnale di reset tra queste.
Il test consiste nella seguente sequenza:
[29, 0, 255, 0, 0, 0 | 0, 0, 0, 0, 29, 0, 0, 0]
Nella prima elaborazione viene dato k=3, nella seconda k = 4 ma con l'indirizzo i_add a partire dalla quarta misurazione (in corrispondenza del divisore).
La sequenza risultante deve essere
[19, 31, 255, 31, 255, 30 | 0, 0, 0, 0, 29, 31, 29, 30]
Nota: nella specifica non viene esplicitamente indicato se la confidenza e l'ultima misurazione valida vadano impostate a 0 tra un'elaborazione e l'altra senza reset. Dato che più elaborazioni possono essere lanciate su zone di memoria totalmente diverse, ho assunto che questi valori vadano reimpostati. Se si volesse creare un componente simile ma che non reimposta i valori senza il segnale di reset, basterebbe collegare il reset dei componenti reg_8 e confidence_countdown ad i_rst invece di quello della FSM.

### 3.2.7 Reset durante l'elaborazione

Il test consiste nel dare il segnale di reset e impostare start a 0 quando l'elaborazione non è ancora terminata. Il componente deve reimpostare tutti i moduli ed essere in grado di riprendere l'esecuzione su un'altra sequenza correttamente.
Per questo test ho utilizzato la stessa sequenza del test "Due elaborazioni consecutive senza reset". Viene data una prima elaborazione con k = 7 (quindi su tutti i valori), per poi interromperla con il segnale di reset dopo 16 cicli di clock (Sufficiente per completare l'elaborazione di 3 misurazioni), per poi rilanciarlo dall'indirizzo corrisponente alla quarta misurazione con k = 4. Il risultato è identico a quelli riportati con le due elaborazioni consecutive senza reset.

# 4 Conclusioni

Dai test condotti, si è giunti alla conclusione che il componente rispetta tutte le richieste della specifica, sia tramite simulazione behavoiral, sia tramite simulazione post-sintesi funzionale. Inoltre, la sintesi non genera alcun latch e i requisiti temporali sono rispettati.
Dal punto di vista architetturale, si è scelto l'utilizzo di una FSM e di sei componenti esterni in modo da permettere maggiore modularità e semplicità.