

SCUOLA DI INGEGNERIA



Electronics Systems Project

Professor
L. Fanucci

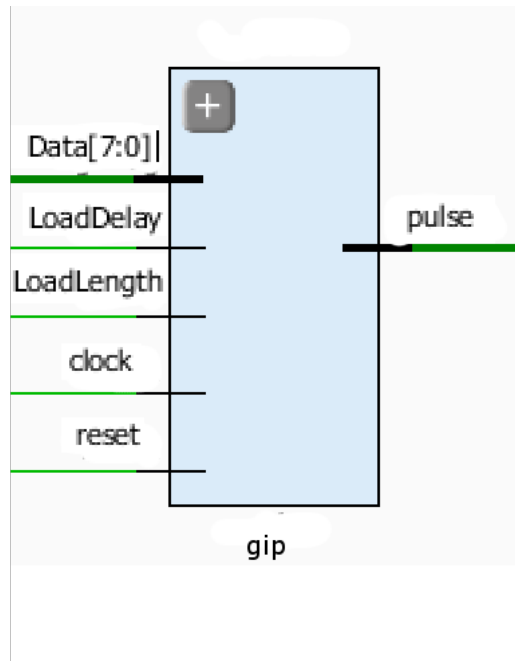
Assistant Professor
L. Pilato

Student
V. Tanferna

A.A. 2016/2017

Generatore di impulsi programmabile

Il requisito di questo progetto è la creazione di un generatore di impulsi programmabile. Il circuito deve essere programmabile sia nella durata dell'impulso (length), che nel ritardo (delay), è composto da un bus dati in ingresso di 8 bit, 4 ingressi: clock, reset e i due segnali di acquisizione dei dati ed un'uscita, l'impulso.

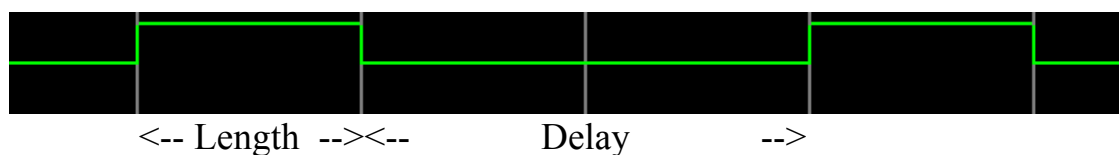


Il segnale di reset è attivo basso per un ciclo di clock, tutti gli elementi con memoria vengono inizializzati a 0.

I segnali per l'acquisizione dei dati sono LoadLength e LoadDelay, entrambi attivi bassi per un ciclo di clock.

Quando uno di essi è attivo basso il valore sul bus dati in ingresso Data, verrà memorizzato nel rispettivo registro interno.

Il bus di uscita avrà la seguente forma:



Descrizione dell'algoritmo

All'avvio, il circuito non produce nessun impulso finché non viene resettato.

Al reset il circuito genera un impulso di durata pari ad un clock con un ritardo di un clock, quindi l'impulso sarà un'onda quadra.

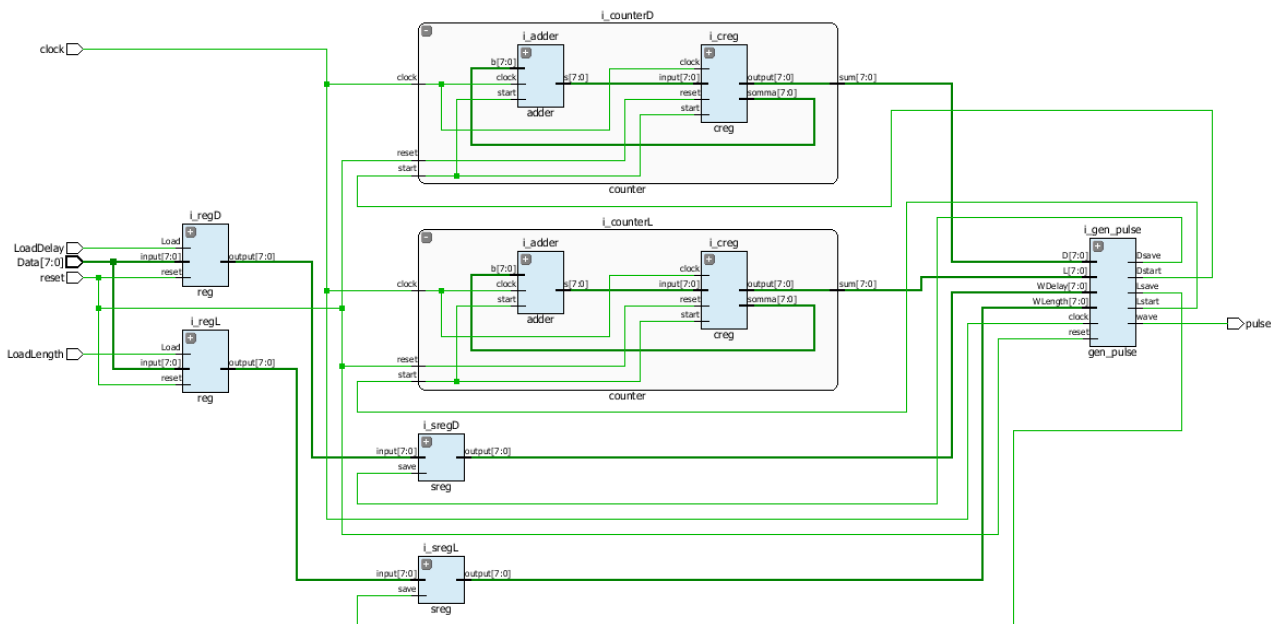
Ogni qual volta un segnale tra LoadDelay e LoadLength diviene attivo basso, il comportamento in durata dell'impulso in uscita cambierà, ma non in maniera istantanea, la modifica del ciclo avverrà alla successiva iterazione.

Questo comportamento si è reso necessario perché nel caso avviene una modifica di uno dei due valori di Length o Delay mentre essi vengono generati, l'impulso potrebbe venire istantaneamente troncato, oppure aumentato saltando un numero di iterazioni.

I valori nuovi vengono memorizzati, in un registro temporaneo interno quando uno dei due impulsi di Load sono attivi bassi ma, essi vengono effettivamente memorizzati ed usati per la generazione dell'impulso solo quando avviene il cambio di stato dell'impulso da basso ad alto/alto a basso.

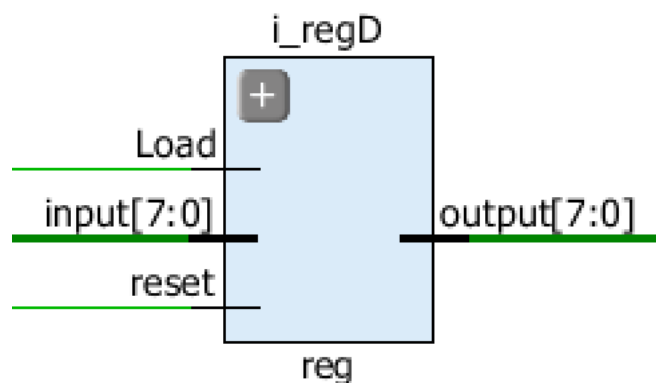
Descrizione dell'architettura

Generatore di impulsi programmabile



Il circuito è composto da due registri di tipo reg, due registri di tipo sreg, un gen_pulse e due counter, il quale possiede all'interno un adder ed un registro di tipo creg. Sono stati adottati registri di diversa tipologia in quanto si necessitava di connessioni e comportamenti differenti.

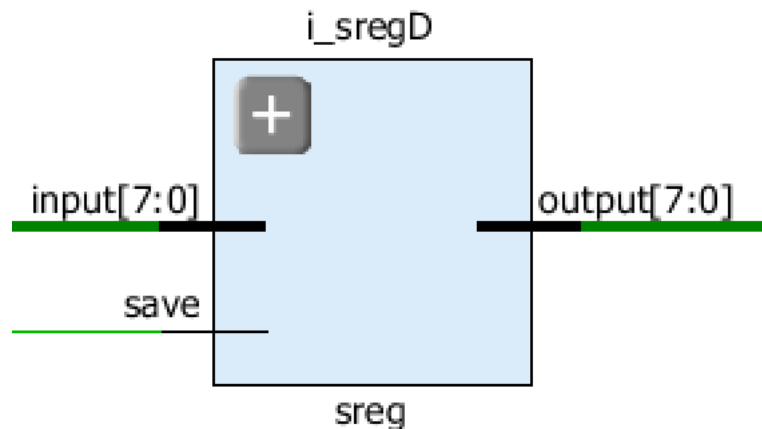
Registro reg



Il registro reg è composto da un segnale di ingresso ad 8 bit dal quale provengono i dati in ingresso, un segnale attivo basso Load, un reset ed un segnale di output su 8 bit. Quando il segnale Load è basso il registro memorizza il valore in ingresso e lo manda in uscita. Al reset memorizza zero. Il registro è necessario affinché il dato in ingresso venga memorizzato solo quando i segnali di Load diventano attivi bassi. L'input è connesso al bus in ingresso, il segnale Load è connesso al segnale LoadLength e LoadDelay rispettivamente per il registro per Length e Delay.

L'output è connesso ad un sreg.

Registro sreg



Il registro sreg è composto da un segnale in ingresso ad 8 bit, il segnale save è connesso al gen_pulse, come anche l'output.

Quando il segnale save diviene alto il registro memorizza il segnale input e lo manda in uscita. Se il segnale in ingresso è uguale a 0 memorizza 1, affinché il circuito GIP in caso di reset abbia in uscita un impulso con Length 1 e Delay 1.

Il registro è necessario affinché il dato in ingresso venga memorizzato solo quando save è attivo alto e cioè, solo quando il segnale deve eseguire lo switch da alto a basso e/o viceversa perché ha raggiunto il numero di clock programmato.

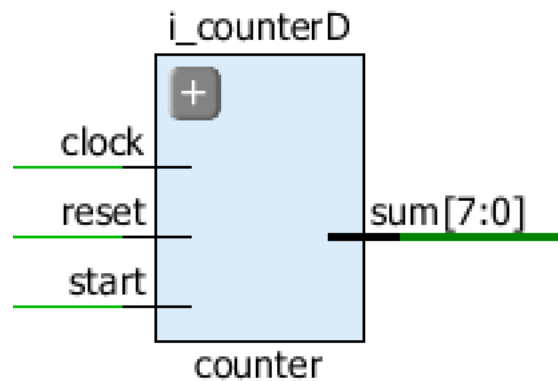
Se il registro non è presente, la lunghezza del segnale verrebbe modificata mentre il segnale è generato, il che fa generare 2 problemi:

- 1) Una lunghezza programmata viene modificata e non viene eseguita.
- 2) In caso il nuovo valore risulti minore rispetto al numero attuale di cicli del segnale, il contatore verrebbe incrementato all'infinito e il circuito non evolverebbe. Di seguito un esempio esplicativo:

Valore attuale memorizzato: 5 5 5 5 5 2 2 2 2 2

Valore contatore: 0 1 2 3 4 5 6 7 8 9...

Counter



Il counter è un componente che, quando il segnale start è attivo alto produce in uscita un valore che, partendo da 0 viene incrementato ogni ciclo di clock.

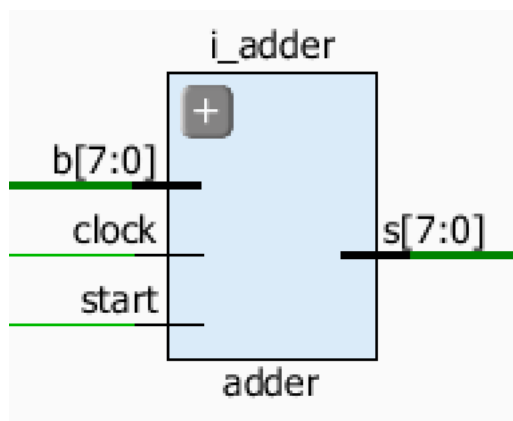
Il componente si rende necessario per contare il numero di cicli in cui, il segnale pulse del circuito GIP deve essere attivo alto o attivo basso.

Il reset è un segnale basso che resetta il registro a 0.

Il segnale start proviene dal componente gen_pulse, sum è connesso al gen_pulse.

Il componente è composto da altri due elementi circuitali: un incrementatore, denominato adder ed un registro creg.

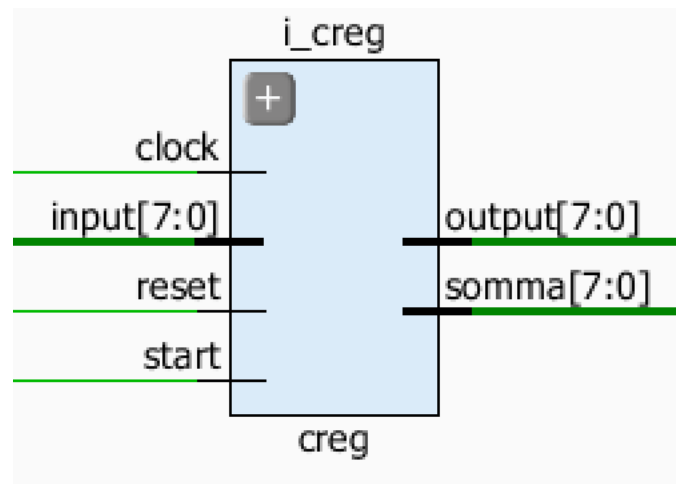
Adder



Questo componente si occupa di incrementare, ogni ciclo di clock, il valore di ingresso b su 8 bit e mandarlo in uscita (s su 8 bit) finché il valore start è alto. s e b sono collegati al creg.

È stato denominato adder, in quanto quel che fa è sommare 1 al valore in ingresso.

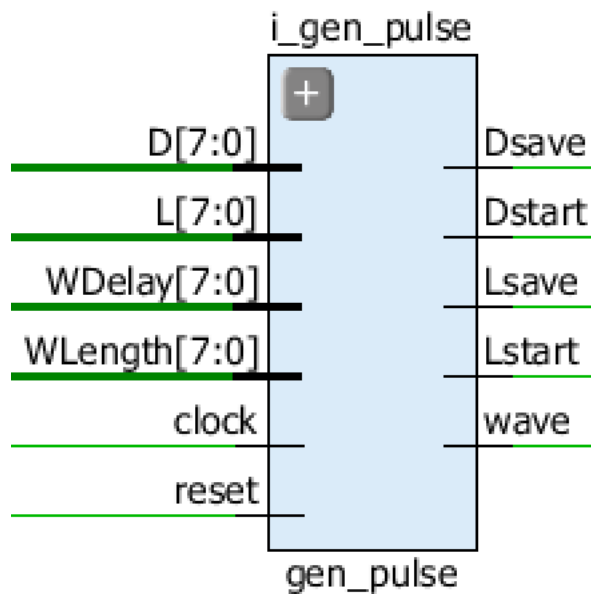
Creg



Questo componente ha in ingresso un valore di input proveniente dal componente adder. L'output è collegato con l'uscita del counter, somma con l'adder. In caso di reset (attivo basso) il valore di uscita (output e somma) assumono valore 0.

Il componente è necessario per memorizzare il valore che, ogni ciclo di clock viene incrementato e mandato al gen_pulse.

Gen_pulse



Questo componente si occupa di generare l'impulso che viene mandato in uscita. In ingresso, oltre ai segnali di clock e reset (sempre attivo basso) ha una serie di segnali su 8 bit, che rappresentano:

WDelay il numero di clock in cui il segnale di uscita deve essere attivo basso, questa porta è connessa al componente sreg per il Delay;

WLength il numero di clock in cui il segnale di uscita deve essere attivo alto, questa porta è connessa al componente sreg per il Length;

D il valore attuale del contatore per il Delay;

L il valore attuale del contatore per il Length.

All'interno sono presenti due porte xor con input ad 8 bit, una confronta WDelay con D e l'altra WLength con L. Questo controllo viene eseguito ogni ciclo di clock per controllare se il valore del counter è uguale al valore di numeri di clock massimo per ogni stato.

In uscita ha un segnale wave, l'impulso da generare, una serie di segnali di tipo attivo alto. Dsave e Lsave sono segnali che vengono inviati, per un ciclo di clock, ai componenti sreg ogni qual volta si rende necessario uno switch del segnale da alto a basso e/o viceversa. Questo indica ai registri di memorizzare il nuovo valore di ingresso. I segnali Dstart e Lstart, anch'essi di tipo attivo alto, vengono posti ad 1 quando il segnale ha eseguito lo switch e permane in tale stato finché il counter non raggiunge il valore limite. In quel caso, i segnali vengono posti a 0.

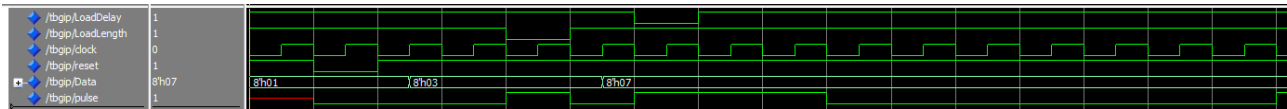
In caso di reset tale circuito cambia lo stato del segnale generato in uscita: se l'uscita attuale è alta e il circuito riceve un reset, l'uscita passa allo stato basso; viceversa se l'uscita attuale è bassa e riceve il reset, l'uscita passa allo stato alto.

Questo comportamento si è reso necessario per risolvere un possibile problema: inizialmente il circuito era stato pensato per generare sempre un'uscita bassa in caso di reset, ma nel caso l'uscita attuale era già bassa e si riceveva il reset, il circuito diveniva insensibile al reset e non inviava i segnali per l'interruzione dei contatori.

Test plan

Sono stati eseguiti vari test per controllare il corretto funzionamento del circuito e possibili bug. Di seguito vedremo i risultati grafici delle simulazioni eseguite con il programma ModelSim.

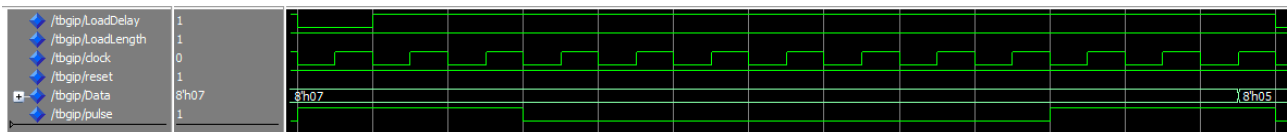
Circuito all'avvio



Al reset i registri sreg avranno memorizzato 1, infatti l'impulso inizia con un andamento oscillatorio. Successivamente LoadLength diviene basso per un ciclo di clock, questo fa sì che il registro sreg per il LoadLength memorizza il dato in ingresso, cioè 3 ed infatti l'onda permane nello stato alto per 3 cicli di clock.

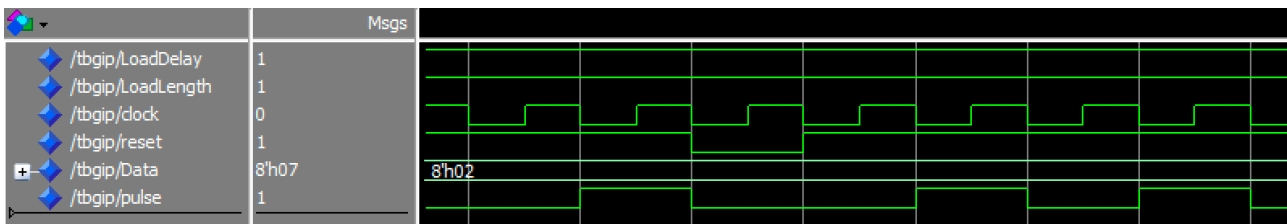
LoadDelay diviene basso per un ciclo di clock, questo fa sì che il registro sreg per il Delay memorizza il dato in ingresso, pari a 7 e ora l'andà avrà una durata attiva alta per 3 cicli di clock e attiva bassa per 7 cicli di clock.

Circuito a regime



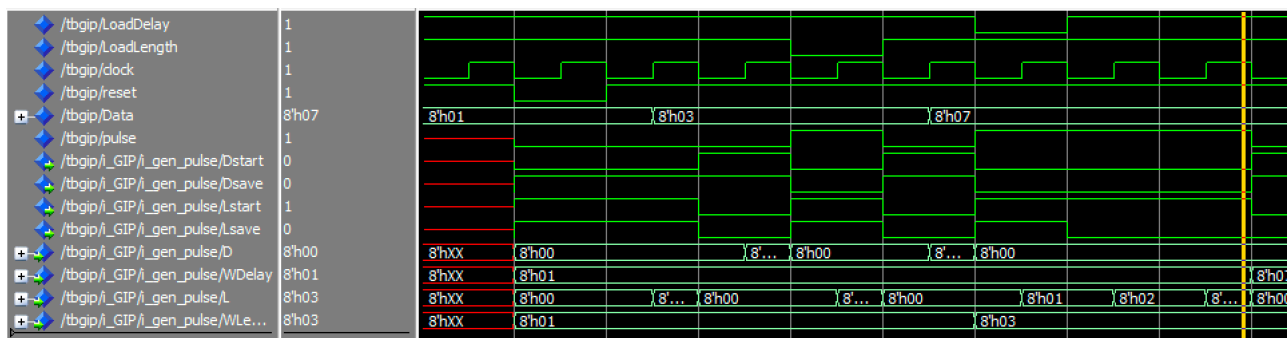
Possiamo vedere il circuito a regime e la memorizzazione di un nuovo dato. Se il salvataggio di un valore relativo ad uno stato (esempio Delay) avviene mentre l'uscita è attualmente nell'altro stato (esempio Length) alla successiva iterazione, essa assumerà questa lunghezza. Diversamente se tale valore (Delay) viene memorizzato durante la fase di Delay, la lunghezza cambierà all'iterazione successiva senza modificare la durata di quella attuale. Questo è stato ottenuto aggiungendo un'altra componente al circuito (sreg).

Reset a regime



A regime quando avviene il reset è possibile osservare che l'impulso di uscita assume valori di Length e Delay pari ad 1.

Avvio: dettagli dei segnali



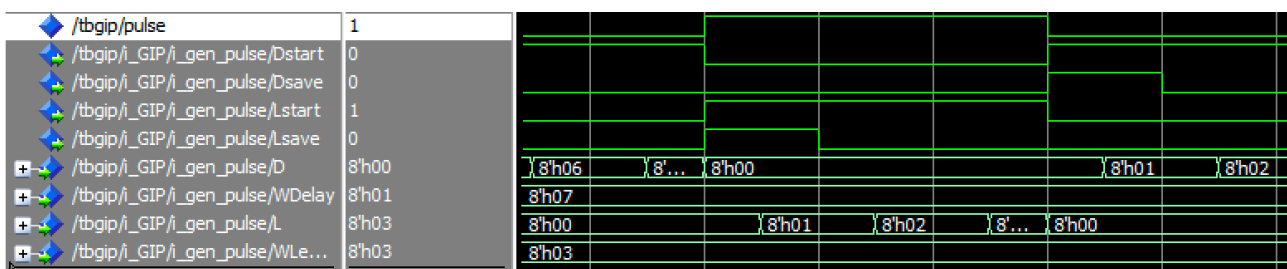
Al reset i segnali di save diventano attivi alti per due cicli di clock e si avvia il contatore per il Delay (Dstart ad 1).

Dopo un solo ciclo di clock viene raggiunto il numero massimo di clock per quello stato (in quanto al reset assume valore 1 sia per Delay che Length) e avviene lo switch dei segnali.

Si memorizza il nuovo valore per entrambi i registri (Lsave e Dsave attivi alti per un ciclo di clock) e si avvia il counter per l'altro registro (in questo caso Length). Di nuovo il numero massimo di clock per quello stato viene raggiunto in un ciclo di clock e così via.

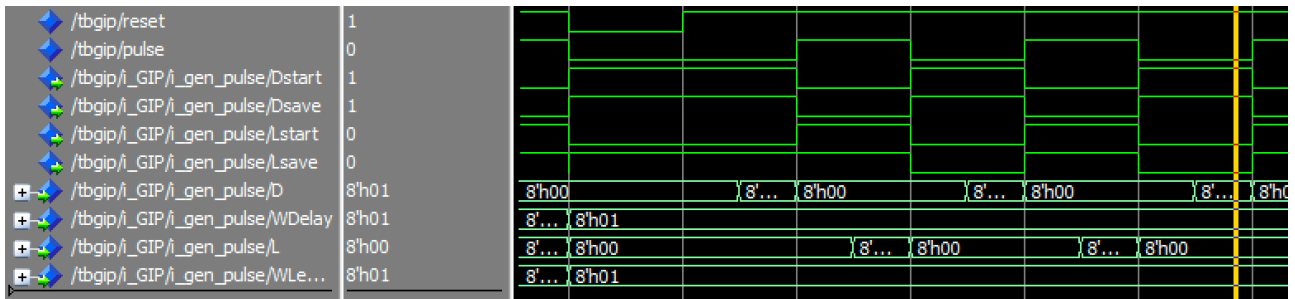
Quando i segnali LoadDelay e LoadLength divengono attivi bassi per un ciclo di clock, avviene la memorizzazione all'interno del registro reg (corrispondere al segnale Load attivo) e quando avviene lo switch del segnale di uscita avviene la memorizzazione nel registro sreg (corrispondere al nuovo stato del segnale di uscita).

È possibile osservare come il valore dei contatori adesso non si ferma solo ad 1, ma da 0 va a 3 per il Length.



Nella figura sopra è possibile vedere il dettaglio con i valori del contatore.

Reset a regime: dettagli dei segnali

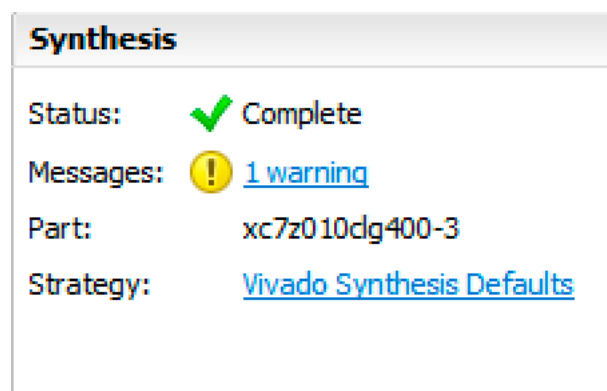


Vediamo un dettaglio dei segnali al reset a regime. L'impulso assume valore basso, i segnali di memorizzazione sono alti (Dsave ed Lsave), viene avviato il counter per il delay (Dstart ad 1 e Lstart a 0).

Da notare il comportamento di Lsave. L'impulso ha terminato lo stato alto e quindi si apprestava ad eseguire lo switch (Dsave a 1 e Lsave a 0) ma, con il segnale di reset Lsave viene riportato ad 1, questo avviene perché al reset entrambi i registri vengono forzati a memorizzare il nuovo valore, cioè 1. Il save permane per 2 cicli di clock ed un terzo perché adesso il segnale di uscita avrà valore 1.

Sintesi

La sintesi è stata eseguita con il tool Vivado 2016.3.



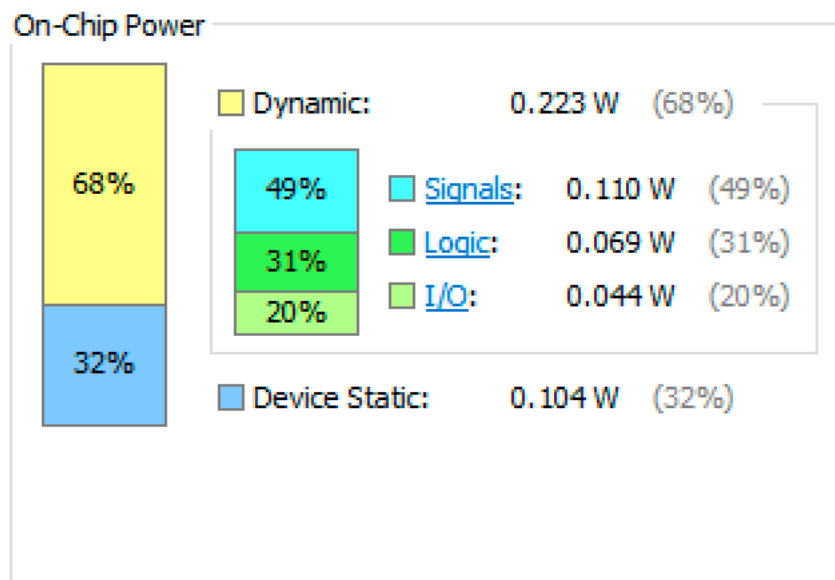
I warning fanno riferimento ad una porta non collegata nel file adder.vhd, in particolare la seguente riga di codice:

```
s<="00000000";
```

Effettivamente la porta risulta collegata non ad un segnale ma ad un valore pari a 0 (cioè tutti i segnali dei singoli bit a valore basso).

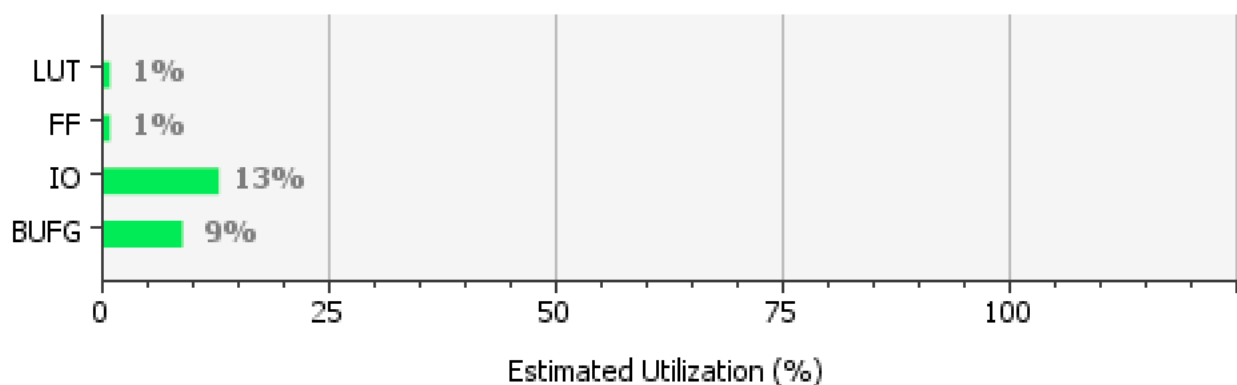
Per quanto riguarda il consumo di potenza si osservano i seguenti valori:

Power	
Total On-Chip Power:	0.327 W
Junction Temperature:	28.8 °C
Thermal Margin:	71.2 °C (6.0 W)
Effective θ_{JA} :	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low
Implemented Power Report	

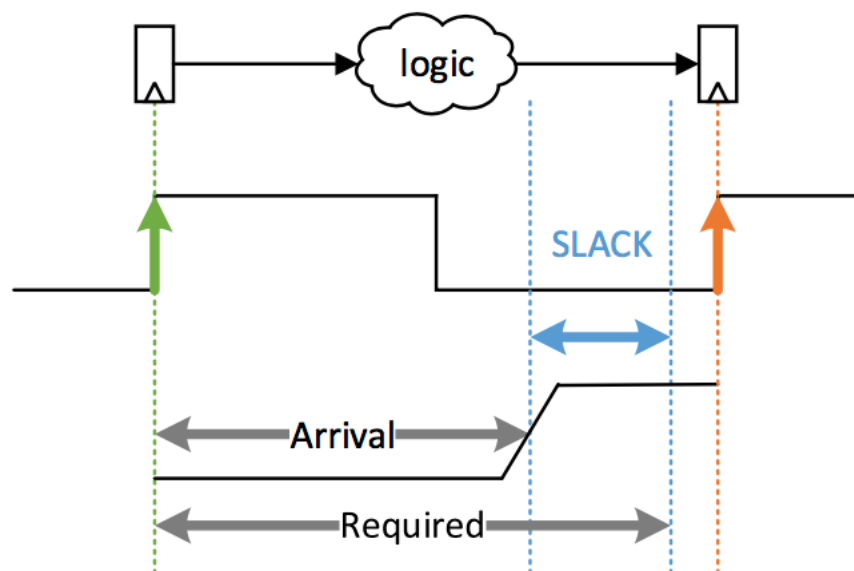


L'utilizzazione del circuito è la seguente:

Resource	Estimation	Available	Utilization %
LUT	36	17600	0.20
FF	56	35200	0.16
IO	13	100	13.00
BUFG	3	32	9.38



Calcolo relativo alla massima frequenza operativa, affinché lo slack sia positivo.



La forma d'onda è settata con periodo di 9 ns, salita a 0 ns e caduta a 4.5 ns.
La frequenza massima è 111 MHz.

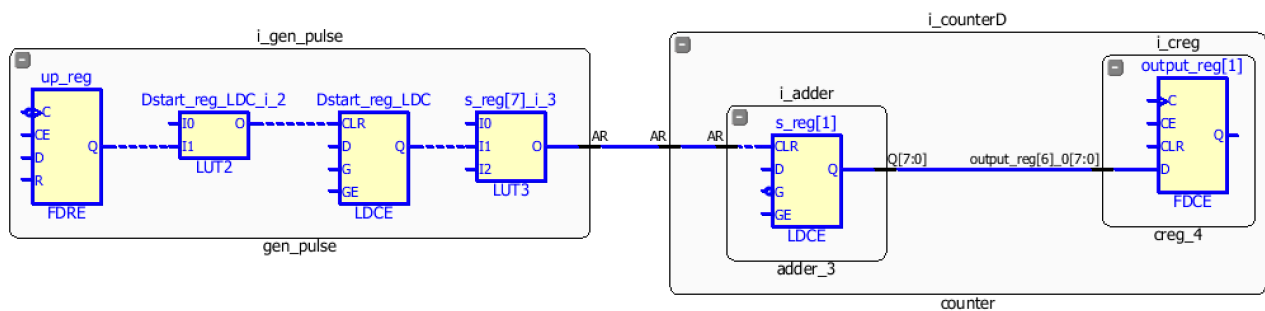
Il risultato è il seguente:

Design Timing Summary			
Setup		Hold	Pulse Width
Worst Negative Slack (WNS): 0.192 ns		Worst Hold Slack (WHS): 0.182 ns	Worst Pulse Width Slack (WPWS): 4.000 ns
Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0		Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 77		Total Number of Endpoints: 77	Total Number of Endpoints: 41
All user specified timing constraints are met.			

I ritardi dei percorsi sono i seguenti:

Intra-Clock Paths - clk - Setup											
Name	...	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	0.192	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[1]/D	4.197	2.013	2.184	4.500	clk	clk
Path 2	0.220	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[2]/D	4.184	2.013	2.171	4.500	clk	clk
Path 3	0.321	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[3]/D	4.079	2.013	2.066	4.500	clk	clk
Path 4	0.387	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[7]/D	4.021	2.013	2.008	4.500	clk	clk
Path 5	0.397	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[6]/D	4.012	2.013	1.999	4.500	clk	clk
Path 6	0.398	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[0]/D	4.001	2.013	1.988	4.500	clk	clk
Path 7	0.476	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[4]/D	3.913	2.013	1.900	4.500	clk	clk
Path 8	0.481	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[3]/D	3.947	2.009	1.938	4.500	clk	clk
Path 9	0.486	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[6]/D	3.945	2.009	1.936	4.500	clk	clk
Path 10	0.491	4	8	i_gen_pulse/up_reg/C	i_counterD/i_creg/output_reg[5]/D	3.913	2.013	1.900	4.500	clk	clk

Il percorso critico è il seguente:


















Summary

Name	Path 1
Slack	0.192ns
Source	i_gen_pulse/up_reg/C (falling edge-triggered cell FDRE clocked by clk111 {rise@0.000ns fall@4.500ns period=9.000ns})
Destination	i_counterD/i_creg/output_reg[1]/D (rising edge-triggered cell FDCE clocked by clk111 {rise@0.000ns fall@4.500ns period=9.000ns})
Path Group	clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	4.500ns (clk111 rise@9.000ns - clk111 fall@4.500ns)
Data Path Delay	4.197ns (logic 2.013ns (47.958%) route 2.184ns (52.042%))
Logic Levels	4 (LDCE=2 LUT2=1 LUT3=1)
Clock Path Skew	-0.026ns
Clock Uncertainty	0.035ns

Percorso critico nel dettaglio:

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDRE (Prop_fdre_C_Q)	(f) 0.347	8.655	Site: SLICE_X41Y5	 i_gen_pulse/up_reg/Q
net (fo=3, routed)	0.418	9.073		 i_gen_pulse/up
			Site: SLICE_X41Y5	 i_gen_pulse/Dstart_reg_LDC_i_2/I1
LUT2 (Prop_lut2_I1_O)	(r) 0.097	9.170	Site: SLICE_X41Y5	 i_gen_pulse/Dstart_reg_LDC_i_2/O
net (fo=4, routed)	0.396	9.566		 i_gen_pulse/Dstart_reg_LDC_i_2_n_0
			Site: SLICE_X41Y4	 i_gen_pulse/Dstart_reg_LDC/CLR
LDCE (SetClr_ldce_CLR_Q)	(f) 0.656	10.222	Site: SLICE_X41Y4	 i_gen_pulse/Dstart_reg_LDC/Q
net (fo=2, routed)	0.471	10.693		 i_gen_pulse/Dstart_reg_LDC_n_0
			Site: SLICE_X39Y4	 i_gen_pulse/s_reg[7]_i_3/I1
LUT3 (Prop_lut3_I1_O)	(r) 0.114	10.807	Site: SLICE_X39Y4	 i_gen_pulse/s_reg[7]_i_3/O
net (fo=8, routed)	0.458	11.265		 i_counterD/i_adder/AR[0]
			Site: SLICE_X41Y2	 i_counterD/i_adder/s_reg[1]/CLR
LDCE (SetClr_ldce_CLR_Q)	(f) 0.799	12.064	Site: SLICE_X41Y2	 i_counterD/i_adder/s_reg[1]/Q
net (fo=1, routed)	0.442	12.506		 i_counterD/i_creg/output_reg[6]_0[1]
FDCE			Site: SLICE_X40Y2	 i_counterD/i_creg/output_reg[1]/D
Arrival Time		12.506		

Conclusioni

Il circuito progettato è in grado di generare un impulso programmabile, rispetta tutti i requisiti progettuali richiesti. È possibile rendere più semplice il circuito, il circuito counter fa da contenitore all'adder ed al creg, questo perché è stato pensato in ottica di scalabilità in caso di modifiche future e per lasciare più pulita l'implementazione del circuito principale.

Codice vhd1

```
-----
-- (Structural)
--
-- File name : GIP.vhd
-- Purpose   : Generatore di impulsi programmabile
--           :
-- Library    : IEEE
-- Author(s)  : Valerio Tanferna
-----
--
-- Il circuito principale, memorizza un nuovo dato su due registri differenti ogni
-- qual volta e' attivo basso uno dei due segnali di Load
-- Genera in uscita un segnale con duty cycle proporzionale ai valori memorizzati nei registri
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity GIP is
    generic (N: integer:=8);
    port(
        Data : in std_logic_vector(N-1 downto 0); -- Dato in ingresso
        reset : in std_logic;                      -- Segnale di reset
        clock : in std_logic;
        LoadLength : in std_logic; -- Segnale per memorizzare il Length del segnale, attivo basso
        LoadDelay : in std_logic;  -- Segnale per memorizzare il Delay del segnale, attivo basso
        pulse : out std_logic);      -- Segnale di uscita
end GIP;

architecture structural of GIP is

    component reg is
        generic (N : integer:=8);
        port(
            input : in std_logic_vector(N-1 downto 0);
            Load : in std_logic;
            reset : in std_logic;
            output : out std_logic_vector(N-1 downto 0));
    end component;

    component counter is
        generic (N : integer:=8);
        port( start          : in std_logic;
              clock          : in std_logic;                      -- incremento sempre di uno
              reset          : in std_logic;
              sum            : out std_logic_vector (N-1 downto 0)); -- inizializzo a zero
    end component;

    component gen_pulse is
        generic (N : integer:=8);
        port( L              : in std_logic_vector(N-1 downto 0); -- from counter D
              D              : in std_logic_vector(N-1 downto 0); -- from counter L
              WDelay         : in std_logic_vector(N-1 downto 0); -- from regD
              WLength        : in std_logic_vector(N-1 downto 0); -- from regL
              reset          : in std_logic;
              clock          : in std_logic;
              Dstart         : out std_logic; -- for start counter D
              Lstart         : out std_logic; -- for start counter L
              Dsave          : out std_logic;
```



```

        Lsave          : out std_logic;
    wave          : out std_logic);
end component;

```

```

component sreg
generic (N : integer:=8);
port(
    input : in std_logic_vector(N-1 downto 0);
    save : in std_logic;
    output : out std_logic_vector(N-1 downto 0));

end component;

```

```

signal Dstart:          std_logic; -- equalL avvia D counter AVVIO IL CIRCUITO ATTIVO BASSO
signal Lstart:          std_logic; -- equalD avvia L counter

```

```

signal Dsave:          std_logic;
signal Lsave:          std_logic;

```

```

signal regL2gen: std_logic_vector(N-1 downto 0);
signal regD2gen: std_logic_vector(N-1 downto 0);

```

```

signal sumD:          std_logic_vector(N-1 downto 0);
signal sumL:          std_logic_vector(N-1 downto 0);

```

```

begin
i_counterD: counter
port map(start => Dstart, clock => clock, reset => reset, sum => sumD);

```

```

i_counterL: counter
port map(start => Lstart, clock => clock, reset => reset, sum => sumL);

```

```

i_regL: reg
port map(input => Data, Load => LoadLength, reset => reset, output => regL2reg);

```

```

i_regD: reg
port map(input => Data, Load => LoadDelay, reset => reset, output => regD2reg);

```

```

i_sregL: sreg
port map(input => regL2reg, save => Lsave, output => regL2gen);

```

```

i_sregD: sreg
port map(input => regD2reg, save => Dsave, output => regD2gen);

```

```

i_gen_pulse: gen_pulse
port map (L => sumL, D => sumD, WDelay => regD2gen, WLength => regL2gen, reset => reset, clock =>
clock, Dstart => Dstart, Lstart => Lstart, Dsave => Dsave, Lsave => Lsave, wave => pulse);

```

```

end structural;

```

```

-----
-- TestBench (GIP)
--
-- File name : tbGIP.vhd
-- Purpose   : Test bench per il GIP
--           :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Il circuito principale, memorizza un nuovo dato su due registri differenti ogni
-- qual volta e' attivo basso uno dei due segnali di Load
-- Genera in uscita un segnale con duty cycle proporzionale ai valori memorizzati nei registri
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbGIP is
end tbGIP;

architecture GIP_test of tbGIP is

    component GIP

        generic (N : integer:=8);
        port(
            LoadDelay : in std_logic; -- Segnale attivo basso, memorizza Delay dell'impulso
            LoadLength : in std_logic; -- Segnale attivo basso, memorizza Length dell'impulso
            clock : in std_logic;      -- Clock
            reset : in std_logic;      -- Segnale attivo basso, resetta il circuito
            Data : in std_logic_vector(N-1 downto 0); -- Segnale per i dati

            pulse : out std_logic);    -- Segnale di uscita, l'impulso
        end component;

    -----

    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

    -----initial value-----
-- INPUT SIGNALS

    signal LoadDelay : std_logic := '1';    -- Inizializzo ad 1, non memorizza il dato
    signal LoadLength : std_logic := '1';    -- Inizializzo ad 1, non memorizza il dato
    signal clock : std_logic := '0';        -- clock iniziale a 1
    signal reset : std_logic := '1';        -- reset ad 1
    signal Data : std_logic_vector(N-1 downto 0) := "00000001"; -- Bus dati ad 1

-- OUTPUT SIGNALS

    signal pulse : std_logic;

    signal clk_cycle : integer;    -- Numero di periodi di clock
    signal Testing : boolean := True;

begin
    i_GIP: GIP generic map(N=>8)
        port map(LoadDelay, LoadLength, clock, reset, Data, pulse);

```

-- Generates clk

clock <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clock)

variable count: integer:= 0;

begin

clk_cycle <= (count+1)/2;

case count is

when 2 => reset<='0'; -- resetto il circuito

when 4 => reset <= '1';

when 5 => Data <="00000011"; -- nuovo dato

when 8 => LoadLength <='0'; -- lo memorizzo nel registro per la lunghezza

when 10 => LoadLength <='1';

when 11 => Data <="00000111"; -- nuovo dato

when 12 => LoadDelay <='0'; -- lo memorizzo nel registro per il ritardo

when 14 => LoadDelay <='1';

when 37 => Data <="00000101"; -- nuovo dato

when 38 => LoadDelay <='0'; -- lo memorizzo nel registro per il ritardo

when 40 => LoadDelay <='1';

when 51 => Data <="00000010"; -- nuovo dato

when 52 => LoadLength <='0'; -- lo memorizzo nel registro per la lunghezza

when 54 => LoadLength <='1';

when 70 => reset <= '0'; -- resetto il circuito

when 72 => reset <= '1';

when 81 => Data <="00000010"; -- nuovo dato

when 82 => LoadLength <='0'; -- lo memorizzo nel registro per la lunghezza

when 84 => LoadLength <='1';

when 99 => Data <="00000001"; -- nuovo dato

when 100 => LoadDelay <='0'; -- lo memorizzo nel registro per il ritardo

when 102 => LoadDelay <='1';

when (TestLen - 1) => Testing <= False;

when others => null;

end case;

count:= count + 1;

end process Test_Proc;

end GIP_test;

-- (Structural)

--
-- File name : gen_pulse.vhd
-- Purpose : Generatore di impulsi casuali
-- :
-- Library : IEEE
-- Author(s) : Valerio Tanferna

-- XOR XNOR
-- 00 0 1
-- 01 1 0
-- 10 1 0
-- 11 0 1
--

--
-- Genera l'impulso che sara' in uscita dal circuito
-- Al reset avvia il counter per il Delay e manda in uscita un segnale basso
-- Per avviare un contatore manda attivo alto il segnale save per salvare
-- Il nuovo dato ingresso al registro, sara' il valore di permanenza del segnale
-- In quello stato (alto o basso) e terra' attivo alto il segnale start
-- Affinche' il counter non generera' un valore uguale a quello ricevuto dal registro
-- Quando tale valore sara' uguale (confronto con una porta XOR), bisognera'
-- Eseguire lo switch del segnale, da attivo alto a basso (o viceversa)
-- Mettera' lo start relativo al contatore che attualmente sta eseguendo a 0
-- Ed alto lo start per l'altro contatore. Save sara' alto per un ciclo di
-- Clock affinche' l'altro registro possa salvare il nuovo dato.
-- Ripetutamente viene eseguito questo ciclo.
--
-- Necessario mantenere stato del segnale in una variabile "up" perche'
-- In caso di reset parto con Delay, ma se sono in stato di Delay
-- Il circuito non e' sensibile e continua a contare all'infinito
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;

entity gen_pulse is
 generic (N : integer:=8);
 port(L : in std_logic_vector(N-1 downto 0); -- Dato dal contatore Length
 D : in std_logic_vector(N-1 downto 0); -- Dato dal contatore Delay
 WDelay : in std_logic_vector(N-1 downto 0); -- Dato dal registro regD
 WLength : in std_logic_vector(N-1 downto 0); -- Dato dal registro regL
 reset : in std_logic;
 clock : in std_logic;
 Dstart : out std_logic; -- Attivo alto, per avviare il counter per il Delay
 Lstart : out std_logic; -- Attivo alto, per avviare il counter per il Length
 Dsave : out std_logic; -- Attivo alto, segnala al registro sreg di salvare il nuovo dato Delay
 Lsave : out std_logic; -- Attivo alto, segnala al registro sreg di salvare il nuovo dato Length
 wave : out std_logic); -- Segnale di uscita
end gen_pulse;

architecture behavioural of gen_pulse is

begin
 gen:process(clock, reset)

 variable xorD: std_logic_vector(N-1 downto 0); -- Variabile di appoggio per risultato dello XOR per il
Delay variable xorL: std_logic_vector(N-1 downto 0); -- Variabile di appoggio per risultato dello XOR per il Length

```

variable Dsavev: std_logic; -- Variabile di appoggio per cambiare lo stato del segnale Save per il Delay
variable Lsavev: std_logic; -- Variabile di appoggio per cambiare lo stato del segnale Save per il Length
variable up: std_logic := '0'; -- Variabile di appoggio che indica lo stato del segnale di uscita: alto (1) o basso

(0)

begin
    if(reset = '0') then
        -- Al reset inizio con segnale di uscita basso
        if(up = '1') then
            -- Se lo stato e' alto
            Dstart <= '1'; -- Avvio contatore per segnale Delay (basso)
            Lstart <= '0'; -- E disattivo quello per lo stato Length (alto)
        else
            -- Se lo stato e' basso
            Dstart <= '0'; -- Disattivo il segnale Delay (basso)
            Lstart <= '1'; -- E avvio contatore per segnale Length (alto)
        end if;

        Dsavev := '1';
        Lsavev := '1';
        if(reset = '0') then -- PER RITARDO non sto programmando: manda tutto
            contemporaneamente
                Dsave <= Dsavev;
                Lsave <= Lsavev;
                wave <= '0'; -- L'impulso di uscita sara' 0
            end if;
        elsif(clock'event and clock='0') then

            Dsavev := '0';
            -- Metto a 0 i segnali si Save il i registri sreg, basta un impulso di un ciclo di clock
            Lsavev := '0'; -- Ad indicare che il registro deve salvare un nuovo valore
            if(Dsavev = '0' and Lsavev = '0') then
                -- PER RITARDO non sto programmando: manda tutto contemporaneamente
                Dsave <= Dsavev;
                Lsave <= Lsavev;
            end if;

            xorD := WDelay xor D; -- Confronto dei risultati per il Delay
            xorL := WLength xor L; -- Confronto dei risultati per il Length
            if(xorD = "00000000") then -- Se uguali, ha raggiunto il tempo, avvio l'altro
                contatore
                    Dstart <= '0';
                    Lstart <= '1'; -- Delay terminato, avvio Length
                    Lsavev := '1';
                    if(xorD = "00000000") then
                        -- PER RITARDO non sto programmando: manda tutto contemporaneamente
                        Lsave <= Lsavev; -- segnale
                    end if;
                    wave <= '1'; -- Segnale di uscita alto
                    up := '1'; -- Cambio stato
                else
                    if(xorL = "00000000") then -- Se uguali, ha raggiunto il tempo, avvio l'altro
                        contatore
                            Dstart <= '1'; -- Length terminato, avvio Delay
                            Lstart <= '0';
                            Dsavev := '1';
                            if(xorL = "00000000") then -- PER RITARDO non sto programmando: manda tutto
                                contemporaneamente
                                    Dsave <= Dsavev; -- segnale
                                end if;
                                wave <= '0'; -- Segnale di uscita basso
                                up := '0'; -- Cambio stato
                            end if;
                        end if;
                    end process gen;
                end behavioural;

```

```

-----
-- TestBench (gen_pulse)
--
-- File name : tbgen_pulse.vhd
-- Purpose   : Test per il generatore di impulso
--           :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Genera l'impulso
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbgen_pulse is

end tbgen_pulse;

architecture gen_pulse_test of tbgen_pulse is

    component gen_pulse

        generic (N : integer:=8);
        port( L      : in std_logic_vector(N-1 downto 0); -- Dato dal contatore Length
              D      : in std_logic_vector(N-1 downto 0); -- Dato dal contatore Delay
              WDelay  : in std_logic_vector(N-1 downto 0); -- Dato dal registro regD
              WLength : in std_logic_vector(N-1 downto 0); -- Dato dal registro regL
              reset   : in std_logic;
              clock   : in std_logic;
              Dstart  : out std_logic;                    -- Attivo alto, per avviare il counter per il Delay
              Lstart  : out std_logic;                    -- Attivo alto, per avviare il counter per il Length
              Dsave   : out std_logic;                    -- Attivo alto, segnala al registro sreg di salvare il nuovo
dato Delay
              Lsave   : out std_logic;                    -- Attivo alto, segnala al registro sreg di salvare il nuovo
dato Length
              wave    : out std_logic);                  -- Segnale di uscita
        end component;

    -----
    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

    -----initial value-----
-- INPUT SIGNALS

    signal D      : std_logic_vector(N-1 downto 0) := "00000000";
    signal L      : std_logic_vector(N-1 downto 0) := "00000000";
    signal WDelay  : std_logic_vector(N-1 downto 0) := "00000010";
    signal WLength : std_logic_vector(N-1 downto 0) := "00000011";
    signal clock   : std_logic := '0';
    signal reset   : std_logic := '1'; -- reset attivo basso

-- OUTPUT SIGNALS

    signal Dstart : std_logic;
    signal Lstart : std_logic;
    signal Dsave  : std_logic;
    signal Lsave  : std_logic;

```

```

signal wave      : std_logic;

signal clk_cycle      : integer;           --number of clk's periods
signal Testing      : boolean := True;

begin

i_gen_pulse : gen_pulse generic map(N=>8)
    port map(L, D, WDelay, WLength, reset, clock, Dstart, Lstart, Dsave, Lsave, wave);

-----

-- Generates clk

clock    <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clock)
    variable count: integer:= 0;
begin
    clk_cycle <= (count+1)/2;

    -- Simulo contatori, WDelay = 2, WLength = 3
    case count is
        when 2 =>      reset <= '0';           -- Resetta il circuito
        when 4  =>      reset <= '1';
        when 5  =>      D <= "00000000"; L <= "00000000";    -- Simulo ricezione dati dai contatori
        when 7 =>      D <= "00000001"; L <= "00000000";
        when 9  =>      D <= "00000010"; L <= "00000000";

        when 13 =>     D <= "00000000"; L <= "00000001";    -- Cambio di stato
        when 15 =>     D <= "00000000"; L <= "00000010";
        when 17 =>     D <= "00000000"; L <= "00000011";
        when 19 =>     D <= "00000001"; L <= "00000000";
        when 21 =>     D <= "00000010"; L <= "00000000";

        when 25 =>     D <= "00000000"; L <= "00000001";    -- Cambio stato
        when 27 =>     D <= "00000000"; L <= "00000010";
        when 29 =>     D <= "00000000"; L <= "00000011";

        when 31 =>     D <= "00000001"; L <= "00000000";

        when (TestLen - 1) => Testing <= False;
        when others => null;
    end case;
    count:= count + 1;
end process Test_Proc;

END gen_pulse_test;

```

```

-----
-- (Structural)
--
-- File name : reg.vhd
-- Purpose  : Generatore di impulsi casuali
--          :
-- Library  : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Quando il reset e' attivo basso, il registro memorizza 0
-- Altrimenti, se il segnale Load e' attivo basso memorizza il dato in gresso
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity reg is
    generic (N : integer:=8);
    port( input      : in std_logic_vector(N-1 downto 0); -- Dato di ingresso, proveniente dal bus dati
          Load       : in std_logic;                    -- Segnale Load attivo basso: l'uscita prende l'ingresso
          reset       : in std_logic;                    -- Reset attivo basso
          output      : out std_logic_vector(N-1 downto 0)); -- Dato di uscita del registro, lo invia al sreg
end reg;

architecture behavioural of reg is
begin
    reg_process(Load, reset)
    begin
        if (reset = '0') then -- Reset attivo basso, l'uscita prende 0
            output <= "00000000";
        elsif (Load'event and Load='0') then -- Load 0
            for i in N-1 downto 0 loop
                output(i) <= input(i); -- L'uscita prende l'ingresso
            end loop;
        end if;
    end process reg;
end behavioural;

```



```

-----
-- TestBench (reg)
--
-- File name : tbreg.vhd
-- Purpose   : Test per il registro
--           :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Ogni volta che il segnale Load diventa attivo basso memorizza
-- un nuovo dato in ingresso e lo inoltra allo slave
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbreg is

end tbreg;

architecture reg_test of tbreg is

    component reg

        generic (N : integer:=8);

        port( input   : in std_logic_VECTOR (N-1 downto 0);          -- Bus dati in ingresso
              Load     : in std_logic;                               -- Segnale di ingresso attivo basso: memorizza
              l'input
              reset     : in std_logic;                               -- Segnale di reset, attivo basso
              output    : out std_logic_vector (N-1 downto 0));      -- Bus dati per l'uscita

    end component;

-----

    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time     := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

-----initial value-----
-- I N P U T   S I G N A L S

    signal Load : std_logic := '1';          -- Inizializzo ad 1, non memorizza il dato
    signal input : std_logic_VECTOR (N-1 downto 0):="00000001"; -- Segnale dato in ingresso inizializzato ad 1
    signal reset : std_logic := '1';         -- reset attivo basso

-- O U T P U T   S I G N A L S

    signal output : std_logic_vector (N-1 downto 0);

    signal clock : std_logic := '0';
    signal clk_cycle : integer;                --number of clk's periods
    signal Testing: boolean := True;

begin

    i_reg : reg generic map(N=>8)
        port map(input, Load, reset, output);

```

```

-----

-- Generates clk

clock    <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clock)
    variable count: integer:= 0;
begin
    clk_cycle <= (count+1)/2;

    case count is
        when 2 => reset<='0';           -- resetto il circuito
        when 4 => reset <='1';
        when 5 => input <="11111111";    -- nuovo dato
        when 6 => Load <= '0';          -- memorizzo nel registro
        when 8 => Load <= '1';
        when 11 => input <="11000111";   -- nuovo dato
        when 12 => Load <= '0';          -- memorizzo nel registro
        when 14 => Load <= '1';
        when 20 => reset<='0';           -- resetto il circuito
        when 22 => reset<='1';
        when 26 => input <="11001111";   -- nuovo dato
        when 28 => Load <= '0';          -- memorizzo nel registro
        when 30 => Load <= '1';
        when (TestLen - 1) => Testing <= False;
        when others => null;
    end case;
    count:= count + 1;
end process Test_Proc;

END reg_test;

```

```

-----
-- (Structural)
--
-- File name : sreg.vhd
-- Purpose  : Generatore di impulsi programmabile
--          :
-- Library  : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Quando il valore save diventa attivo alto, il valore di uscita prende il valore
-- Di ingresso, ma, se quest'ultimo e' zero viene posto ad 1
-- Questo affinche' tutto il circuito quando resettato possa generare
-- Un impulso di lunghezza 1 e ritardo 1
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity sreg is
    generic (N: integer:=8);
    port(
        input : in std_logic_vector(N-1 downto 0); -- Valore di ingresso, proveniente dal reg
        save : in std_logic;                        -- Segnale attivo alto, proveniente dal gen_pulse
        output : out std_logic_vector(N-1 downto 0)); -- Valore di uscita
end sreg;

architecture behavioural of sreg is
begin
    sreg:process(save)--clock, save)
    begin
        if(save'event and save='1')then -- L'uscita prende l'ingresso quando il segnale save diventa 1
            if(input = "00000000")then -- Se zero, l'uscita prende 1
                output <= "00000001";
            elsif(input /= "00000000") then -- Se diverso da 0, l'uscita prende l'ingresso
                output <= input;
            end if;
        end if;
    end process sreg;
end behavioural;

```

```

-----
-- TestBench (sreg)
--
-- File name : tbsreg.vhd
-- Purpose   : Test per il registro
--           :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Ogni volta che il segnale Load diventa attivo basso memorizza
-- un nuovo dato in ingresso e lo inoltra allo slave
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbsreg is

end tbsreg;

architecture sreg_test of tbsreg is

    component sreg

        generic (N : integer:=8);

        port(
            input : in std_logic_vector(N-1 downto 0); -- bus dati in ingresso
            save  : in std_logic;                       -- attivo alto (per un ciclo di clock), memorizza il dato in
ingresso
            clock : in std_logic;
            output : out std_logic_vector(N-1 downto 0)); -- bus dati di uscita

        end component;

    -----

    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

    -----initial value-----
-- INPUT SIGNALS

    signal save : std_logic := '0'; -- inizializzato a 0, non memorizzo
    signal input : std_logic_VECTOR (N-1 downto 0):="00000001";

-- OUTPUT SIGNALS

    signal output : std_logic_vector (N-1 downto 0);

    signal clock : std_logic := '0';
    signal clk_cycle : integer; --number of clk's periods
    signal Testing: boolean := True;

begin

    i_sreg : sreg generic map(N=>8)
        port map(input, save, clock, output);

```

```

-----

-- Generates clk

clock    <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clock)
    variable count: integer:= 0;
begin
    clk_cycle <= (count+1)/2;

    case count is
        when 2 => save<='1';           -- memorizza il dato
        when 4 => save <='0';
        when 5 => input<="00000000";   -- nuovo dato
        when 6 => save<='1';           -- memorizza il dato
        when 8 => save<='0';
        when 9 => input<="00010000";   -- nuovo dato
        when 10 => save<='1';          -- memorizza il dato
        when 12 => save<='0';
        when 14 => input<="00011000";  -- nuovo dato
        when 16 => save<='1';          -- memorizza il dato
        when 18 => save<='0';
        when 19 => input<="00000011";  -- nuovo dato
        when 22 => save<='1';          -- memorizza il dato
        when 24 => save<='0';
        when 25 => input<="00000000";  -- nuovo dato
        when 26 => save<='1';          -- memorizza il dato
        when 28 => save<='0';

        when (TestLen - 1) => Testing <= False;
        when others => null;
    end case;
    count:= count + 1;
end process Test_Proc;

END sreg_test;

```

```

-----
-- (Structural)
--
-- File name : counter.vhd
-- Purpose  : Generatore di impulsi programmabile
-- :
-- Library  : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Finche' start e' attivo alto, genera un dato in uscita, partendo da 0
-- e lo incrementa per ogni ciclo di clock
--
-----

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;

entity counter is

    generic (N : INTEGER:=8);
    port( start          : in std_logic;                -- Segnale attivo alto, genera numero, ricevuto dal
    gen_pulse
        clock           : in std_logic;
        reset           : in std_logic;                -- Al reset interrompe l'incremento e genera un
valore pari a 0
        sum             : out std_logic_vector (N-1 downto 0)); -- Segnale di output inviato al gen_pulse
end counter;

architecture structure of counter is
    component adder is -- componente adder
        generic (N : integer:=8);
        port(
            b          : in std_logic_VECTOR (N-1 downto 0); -- Segnale dato in ingresso da incrementare,
proveniente dal registro creg
            start      : in std_logic; -- Segnale attivo alto, finche' e' alto, incrementa
il valore ogni ciclo di clock
            clock      : in std_logic;
            s          : out std_logic_VECTOR (N-1 downto 0)); -- Segnale di uscita, il valore di ingresso e'
incrementato
        end component;

    component creg is
        generic (N: integer:=8);
        port(
            input : in std_logic_vector(N-1 downto 0);
            start : in std_logic;
            clock : in std_logic;
            reset : in std_logic;
            output : out std_logic_vector(N-1 downto 0);
            somma : out std_logic_vector(N-1 downto 0));
    end component;

    --signal clock : std_logic;
    --signal start : std_logic;
    signal s : std_logic_vector(N-1 downto 0);
    signal b : std_logic_vector(N-1 downto 0);
    signal output : std_logic_vector(N-1 downto 0);

begin
    i_creg: creg

```

```
port map(input => s, start => start, clock => clock, reset => reset, output => sum, somma => b);

i_adder: adder
port map(b => b, start => start, clock => clock, s => s);

end structure;
```

```

-----
-- (Structural)
--
-- File name : tbcounter.vhd
-- Purpose  : Generatore di impulsi programmabile
--          :
-- Library  : IEEE
-- Author(s) : Valerio Tanferna
-----

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

entity tbcounter is

end tbcounter;

architecture counter_test of tbcounter is

    component counter
        generic (N : INTEGER:=8);
        port( start          : in std_logic;
dal gen_pulse              : in std_logic;
                        clock      : in std_logic;
                        reset       : in std_logic;
genera un valore pari a 0
                        sum         : out std_logic_vector (N-1 downto 0));
        end component;
-- Segnale attivo alto, genera numero, ricevuto
-- Al reset interrompe l'incremento e
-- Segnale di output inviato al gen_pulse
-----

    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

-----initial value-----
-- INPUT SIGNALS

    signal start : std_logic := '0';
    signal clock : std_logic := '0';
    signal reset : std_logic := '1';
--clk initial value

-- OUTPUT SIGNALS

    signal sum : std_logic_vector (N-1 downto 0);

    signal clk_cycle : integer;
    signal Testing: boolean := True;
--number of clk's periods

    begin
        i_counter: counter generic map(N=>8)
        port map(start, clock, reset, sum);
-----

-- Generates clk

    clock <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

```



```

Test_Proc: process(clock)
    variable count: INTEGER:= 0;
begin
    clk_cycle <= (count+1)/2;

    case count is
        when 2 => start<='1';      -- Avvio il counter per vedere come evolve l'uscita per vari cicli di clock
        when 8 => start<='0';
        when 10 => start<='1';
        when 16 => start<='0';
        when 20 => start<='1';
        when 28 => start<='0';
        when 30 => start<='1';
        when 34 => start<='0';
        when (TestLen - 1) => Testing <= False;
        when others => null;
    end case;
    count:= count + 1;
end process Test_Proc;
end counter_test;

```

```

-----
-- (Structural)
--
-- File name : adder.vhd
-- Purpose   : Counter register
--           :
-- Library    : IEEE
-- Author(s)  : Valerio Tanferna
-----

--
-- Incrementa di 1 il valore b dato in ingresso e lo manda in output s
-- ogni ciclo di clock finche' start e' attivo alto
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity adder is

    generic (N : integer:=8);
    port(
        b      : in std_logic_vector (N-1 downto 0);  -- Segnale dato in ingresso da incrementare, proveniente dal
        registro creg
        start    : in std_logic;                        -- Segnale attivo alto, finche' e' alto, incrementa il valore
        ogni ciclo di clock
        clock     : in std_logic;
        s        : out std_logic_vector (N-1 downto 0)); -- Il valore di uscita incrementato, collegata alla porta di input del
        creg
    end adder;

architecture behavioural of adder is

begin

    inc:process(start, clock, b)
        variable C:std_logic;                        -- variabile di appoggio per il carry
        variable a:std_logic_vector(N-1 downto 0);   -- variabile di appoggio
        begin
            C:='0';                                -- carry inizializzato ad 1
            a:="00000001";                          -- Incremento sempre il valore di ingresso di 1
            if(start = '0')then                      -- Quando start e' basso il valore di uscita e' 0
                s<="00000000";
            elsif(clock='0')then                    -- Se il clock e' 0 fai la somma (incrementa di 1)
                for i in 0 to N-1 loop
                    -- Calculate bit sum using carry from previous step, then carry out
                    s(i)<= a(i) xor b(i) xor C;
                    C:= (a(i) and b(i)) or (a(i) and C) or (b(i) and C);
                end loop;
            end if;

        end process inc;

end behavioural;

```

```

-----
-- (Structural)
--
-- File name : tbadder.vhd
-- Purpose   : Counter register
--           :
-- Library    : IEEE
-- Author(s) : Valerio Tanferna
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbadder is

end tbadder;

architecture adder_test of tbadder is

    component adder

        generic (N : integer:=8);
        port(
            b      : in std_logic_vector (N-1 downto 0);  -- Segnale dato in ingresso da incrementare, proveniente dal
registro creg
            start   : in std_logic;                        -- Segnale attivo alto, finche' e' alto, incrementa il valore
ogni ciclo di clock
            clock   : in std_logic;
            s      : out std_logic_vector (N-1 downto 0)); -- Il valore di uscita incrementato, collegata alla porta di input del
creg
        end component;

    -----
    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 24;   -- No. of Count (MckPer/2) for test

    -----initial value-----
-- INPUT SIGNALS

    signal clk : std_logic := '0';
    signal b   : std_logic_VECTOR (N-1 downto 0):="00000000";
    signal start : std_logic :='0';
    signal clock : std_logic :='0';

-- OUTPUT SIGNALS

    signal s : std_logic_vector (N-1 downto 0);

    signal clk_cycle : integer;
    signal Testing: Boolean := True;

begin

    i_adder : adder generic map(N=>8)
        port map(b, start, clock, s);

    -----

    -- Generates clk

```

```

    clk    <= not clk after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clk)
    variable count: integer:= 0;
begin
    clk_cycle <= (count+1)/2;

    case count is
        when 2 => start <= '1';    -- Avvio, il valore verra' incrementato
        when 4 => start <= '0';
        when 6 => b <= "00000010"; -- Nuovo valore
        when 10 => start <= '1';    -- Avvio, il valore verra' incrementato
        when 14 => start <= '0';

        when (TestLen - 1) => Testing <= False;
        when others => null;
    end case;

    count:= count + 1;
end process Test_Proc;

END adder_test;

```

```

-----
-- (Structural)
--
-- File name : creg.vhd
-- Purpose   : Counter register
--          :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Al reset o quando il segnale di start e' a 0, genera un output con valore 0
-- Quando start e' attivo alto manda in uscita il valore in ingresso
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity creg is
    generic (N: integer:=8);
    port(
        input : in std_logic_vector(N-1 downto 0); -- Segnale di ingresso incrementato, ricevuto
        dall'adder
        start : in std_logic;                        -- Segnale start attivo alto
        clock : in std_logic;                        -- Segnale di clock
        reset : in std_logic;                        -- Segnale di reset, attivo basso,
        l'output generato sara' 0
        output : out std_logic_vector(N-1 downto 0); -- Segnale di output mandato al
        generatore di impulsi, collegato all'uscita del counter
        somma : out std_logic_vector(N-1 downto 0) ); -- Segnale di output inviato all'adder
        per incrementarlo, collegato all'adder
    end creg;

architecture behavioural of creg is
begin
    creg:process(clock, start, reset)
    begin
        if(start = '0' or reset = '0')then          -- Se start o il reset e' attivo basso, in uscita il valore sara' 0
            output <= "00000000";
            somma <= "00000000";
        elsif(clock'event and clock = '1')then      -- Al clock il valore di uscita prendera' il valore di ingresso
            output <= input;
            somma <= input;
        end if;
    end process creg;
end behavioural;

```

```

-----
-- TestBench (creg)
--
-- File name : tbcreg.vhd
-- Purpose   : Test per il registro contatore
--           :
-- Library   : IEEE
-- Author(s) : Valerio Tanferna
-----

--
-- Ogni volta che il segnale Load diventa attivo basso memorizza
-- un nuovo adto in ingresso e lo inoltra allo slave
--
-----

library IEEE;
use IEEE.std_logic_1164.all;

entity tbcreg is

end tbcreg;

architecture creg_test of tbcreg is

    component creg

        generic (N : integer:=8);
        port(
            input : in std_logic_vector(N-1 downto 0); -- Segnale di ingresso incrementato, ricevuto
            start : in std_logic;                        -- Segnale start attivo alto
            clock : in std_logic;                        -- Segnale di clock
            reset : in std_logic;                        -- Segnale di reset, attivo basso,
            output : out std_logic_vector(N-1 downto 0); -- Segnale di output mandato al
            somma : out std_logic_vector(N-1 downto 0) ); -- Segnale di output inviato all'adder

        dall'adder
        l'output generato sara' 0
        generatore di impulsi, collegato all'uscita del counter
        per incrementarlo, collegato all'adder
        end component;

    constant N      : integer := 8;    -- Bus Width
    constant MckPer : time    := 200 ns; -- Master Clk period
    constant TestLen : integer := 500;  -- No. of Count (MckPer/2) for test

    -----initial value-----
    -- INPUT SIGNALS

    signal start : std_logic := '0';
    signal input : std_logic_VECTOR (N-1 downto 0):="00000001";

    -- OUTPUT SIGNALS

    signal output : std_logic_vector (N-1 downto 0);
    signal somma : std_logic_vector (N-1 downto 0);

    signal reset : std_logic := '1';
    signal clock : std_logic := '0';
    signal clk_cycle : integer;
    signal Testing: boolean := True;
    --number of clk's periods

```

```

begin

i_creg : creg generic map(N=>8)
  port map(input, start, clock, reset, output, somma);

-----

-- Generates clk

clock    <= not clock after MckPer/2 when Testing else '0';

-- Runs simulation for TestLen cycles;

Test_Proc: process(clock)
  variable count: integer:= 0;
begin
  clk_cycle <= (count+1)/2;

  case count is
    when 2 => start<='1';           -- In uscita il valore di ingresso
    when 4 => start <='0';
    when 5 => input<="00000000";    -- Nuovo valore
    when 6 => start<='1';           -- In uscita il valore di ingresso
    when 8 => start<='0';
    when 9 => input<="00010000";    -- Nuovo valore
    when 10 => start<='1';          -- In uscita il valore di ingresso
    when 12 => start<='0';
    when 15 => input<="00011000";   -- Nuovo valore
    when 16 => start<='1';          -- In uscita il valore di ingresso
    when 18 => start<='0';
    when 19 => input<="00000011";   -- Nuovo valore
    when 22 => start<='1';          -- In uscita il valore di ingresso
    when 24 => start<='0';
    when 25 => input<="00000000";   -- Nuovo valore
    when 26 => start<='1';          -- In uscita il valore di ingresso
    when 28 => start<='0';

    when (TestLen - 1) => Testing <= False;
    when others => null;
  end case;
  count:= count + 1;
end process Test_Proc;

end creg_test;

```