

UNIVERSITÀ DI PISA



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Progetto di DSD

Riconoscitore di sequenza sicura

di Giovanni Pollina

Anno accademico 2015/2016

Indice

1. Problema

1.1. Requisiti.	3
-------------------------	---

2. Introduzione

2.1. Possibili applicazioni	4
2.1.1. Riconoscitori di sequenza binaria.	4
2.1.2 Riconoscitori di sequenza alfanumerica.	5
2.2. Possibili architetture	5
2.2.1. Architettura di Mealy.	5
2.2.2. Architettura di Moore	6
2.3. Algoritmo proposto	7

3. Architettura

3.1. Schema a blocchi con registri	8
3.2. Modello strutturale.	8
3.3. Codice VHDL	9
3.4 Codice TestBench VHDL	11

4. Test Plan

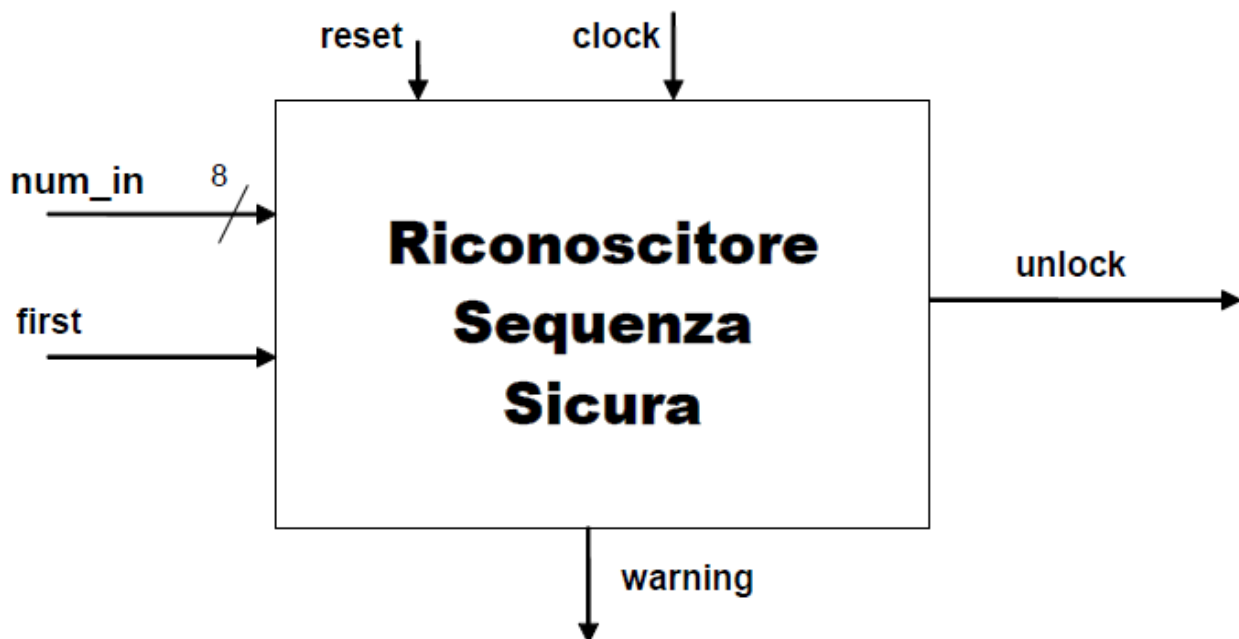
4.1. Architettura di testing	14
4.2. Code coverage.	17

5. Sintesi

5.1. Architettura di base di un FPGA.	22
5.2. Struttura della Zybo Board.	23
5.3. Wrapper per il riconoscitore di sequenza	24
5.4. Spazio occupato e frequenza massima.	26
5.5. Generazione del programming file.	28

1 Problema

Si chiede di progettare ed implementare in VHDL un riconoscitore di sequenza con le seguenti caratteristiche: il riconoscitore è sincrono; la sequenza è composta da 5 numeri espressi su 8 bit (ed in particolare è: 36, 19, 56, 101, 73); il primo numero significativo deve essere accompagnato da un segnale di *first*; le cifre successive devono essere presentate al componente con la cadenza di un ciclo di clock ciascuna; il segnale di *first* deve essere alto solo durante il primo numero, altrimenti il riconoscitore mette il segnale di *warning* a 1 e smette di riconoscere sequenze fintanto che non viene resettato (a questo scopo c'è il piedino di *reset*); la durata di un riconoscimento è sempre pari a 5 cicli di clock (anche nel caso in cui il primo numero fosse sbagliato); se, dopo 5 cicli di clock, la sequenza viene riconosciuta, al ciclo successivo al riconoscimento dell'ultimo numero, il piedino *unlock* viene messo a 1 per un ciclo di clock altrimenti il piedino *warning* viene messo a 1 per un ciclo di clock; se si fallisce il riconoscimento per 3 volte consecutive, il piedino *warning* viene messo costantemente ad 1 e il riconoscitore smette di funzionare fintanto che non viene resettato. Al reset, entrambi i piedini *unlock* e *warning* sono messi a 0. Il segnale di reset può essere a scelta attivo alto oppure attivo basso.



2 Introduzione

2.1 Possibili applicazioni

I riconoscitori di sequenza hanno il compito di controllare se una data sequenza di ingressi appartiene ad un insieme predefinito di sequenze, stabilito a priori dal progettista in base a determinati scopi. In particolare i riconoscitori di sequenza binaria controllano flussi di bit, cioè sequenze di '0' e '1', per intercettare quelle accettabili.

2.1.1 Riconoscitori di sequenza binaria

Questi riconoscitori lavorano con segnali digitali che racchiudono informazioni elementari, per questo sono solitamente sottosistemi, piccoli moduli, facenti parte di una aggregazione di funzioni che svolgono compiti di riconoscimento elementari ma importanti nel complesso.

Possibili usi di questi riconoscitori possono essere:

- Riconoscimento delle sequenze di sincronismo nelle trasmissioni I dati nei canali di comunicazione digitale viaggiano come sequenza di '0' e '1', tra i pacchetti di dati vengono inviati dei segnali di sincronismo (che servono a stabilire, mantenere e controllare la comunicazione tra i dispositivi); le sequenze di sincronismo sono configurazioni prefissate di bit, organizzate in modo da essere facilmente riconoscibili e da non interferire con i dati. Lo scopo è appunto riconoscere tale sequenza e discriminarla dai dati utili.
- Controlli a distanza, telecomandi Anche in questo caso l'informazione è codificata in forma digitale, i vari comandi impartiti vengono codificati ciascuno con una diversa sequenza di bit; l'automa riconoscitore riconosce le diverse sequenze distribuendo gli ordini all'organo a cui è destinato il comando.
- Controllo dell'integrità di un'onda Ad esempio il segnale di clock: è una successione ordinata di '0' e '1' del tipo 01010101... l'automa in questo caso deve controllare la sequenza ed emettere un segnale di allarme in caso si presenti un bit che altera la sequenza corretta.

2.1.2 Riconoscitori di sequenza alfanumerica

Gli automi riconoscitori alfanumerici trattano lettere e numeri al posto dei bit.

- Questi automi trovano applicazione in ambito informatico nei sistemi che interpretano le istruzioni di programmazione: devono riconoscere localizzare le istruzioni di programmazione all'interno di un insieme predefinito di parole chiave che costituiscono la sintassi del linguaggio.
- Un altro campo di applicazione è l'analisi del linguaggio. Ad esempio un sistema in grado di riconoscere una parola, oppure una sequenza di lettere all'interno di una parola e di sostituirla con un'altra sequenza di lettere prefissata. All'interno di questa categoria si trovano i sistemi in grado di analizzare un testo composto da simboli alfanumerici e riconoscere un particolare codice, ad es. una password. Un esempio è il riconoscimento della combinazione di un sistema di sicurezza come una cassaforte.

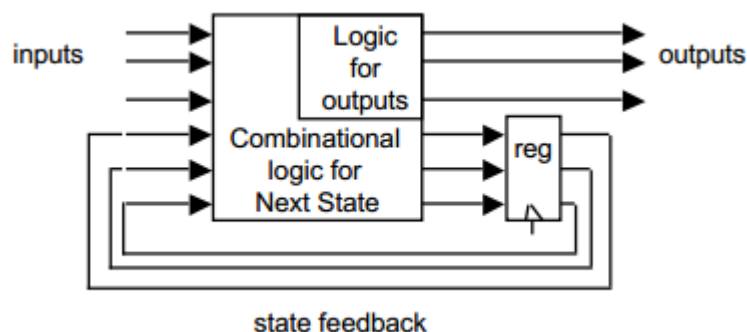
2.2 Possibili architetture

Un riconoscitore di sequenza è una macchina sequenziale sincrona governata da un segnale di clock, e pertanto si può realizzare seguendo le due architetture principali ovvero: l'architettura di Mealy e quella di Moore.

2.2.1 Architettura di Mealy

Nell'architettura di Mealy lo stato di uscita dipende direttamente dallo stato interno e da quello di ingresso e inoltre le variabili di uscita sono prelevate direttamente dalla rete combinatoria principale.

L'architettura della macchina di Mealy è riportata in figura. (Modello strutturale).

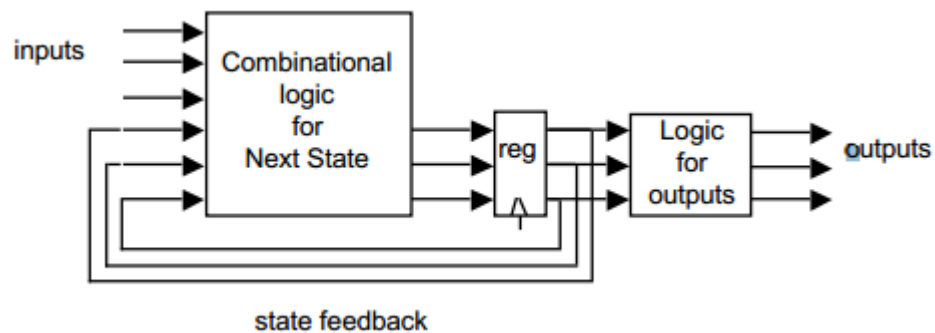


Come si nota in figura è presente una sola rete combinatoria che accetta in ingresso lo stato di input e quello interno presentato tramite un feedback che mantiene lo stato interno, è presente inoltre un registro formato da flip-flop jk che rende la rete di tipo sequenziale sincrona e che memorizza lo stato interno per un ciclo di clock.

2.2.2 Architettura di Moore

Nell'architettura di Moore invece lo stato di uscita dipende solo dallo stato interno attuale e inoltre sono presenti due reti combinatorie, una di ingresso e una di uscita.

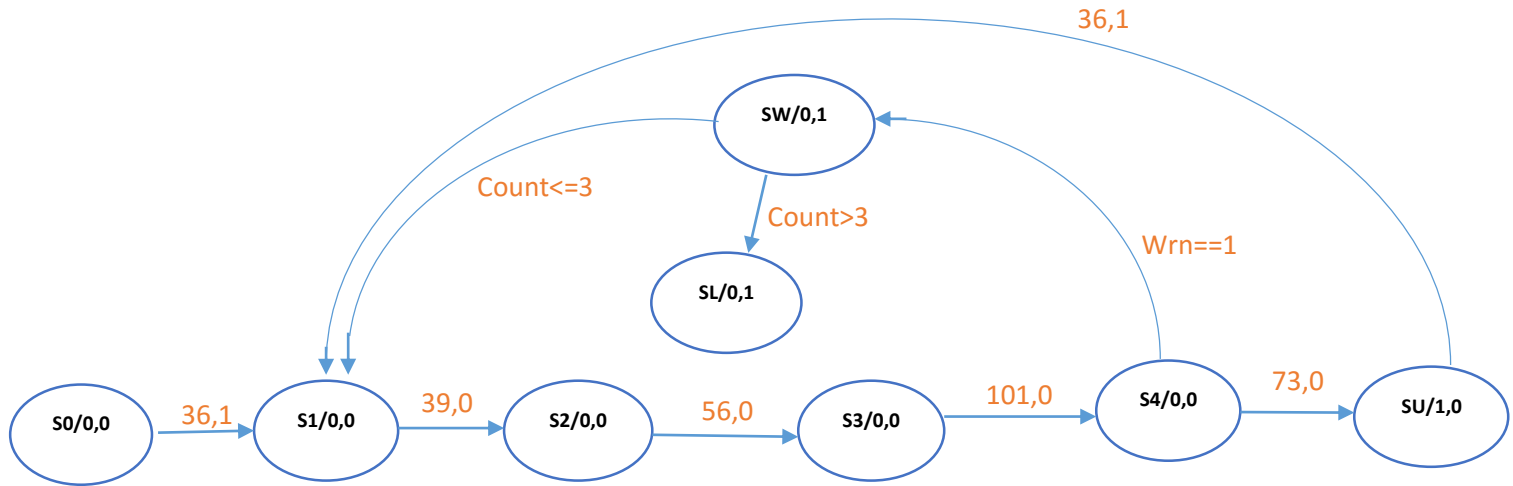
L'architettura della macchina di Moore è riportata in figura. (Modello strutturale).



Nel nostro caso verrà proposto un approccio basato su macchina di Moore ovvero una macchina in cui lo stato di uscita dipende ad ogni istante univocamente dallo stato interno.

2.3 Algoritmo proposto

Per descrivere il funzionamento del riconoscitore utilizzeremo un diagramma di stato, più precisamente quello relativo alla macchina di Moore dove in ogni stato viene rappresentato lo stato interno e quello di uscita.



Nel grafico, i pallini rappresentano gli stati che vengono rappresentati con la notazione: Stato interno/Unlock pin, Warning pin ; per esempio SU/1,0 significa Unlocked State con Unlock pin attivo alto e Warning pin attivo basso.

Le frecce rappresentano i passaggi di stato e su di esse è rappresentato lo stato di ingresso necessario per effettuare un passaggio di stato interno.

Nel grafico vengono omesse per semplicità gli archi che partono da ogni nodo allo stato SL, infatti se in qualsiasi stato che non sia S0, SU o SW si ha il bit di ingresso First bit attivo alto si passa direttamente allo stato locked SL.

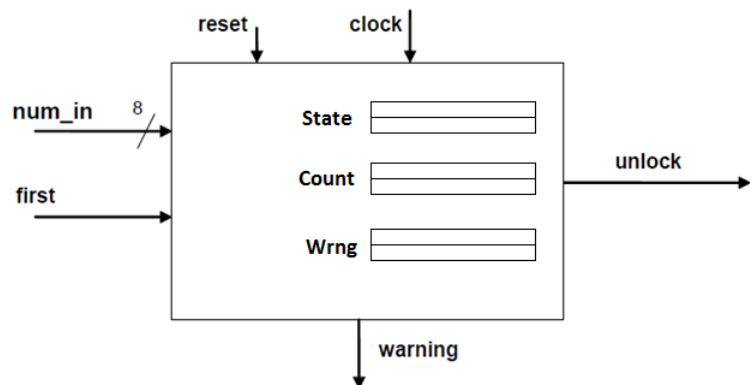
Tipicamente dopo il reset iniziale si raggiunge lo stato S1 ogni 5 cicli di clock (quindi al sesto ciclo si è in stato S1), inoltre al 5 ciclo di clock può essere raggiunto lo stato SU che corrisponde allo stato di sblocco che presenta uno stato di uscita (Unlock bit =1 ;Warning bit=0), lo stato SW che notifica che la sequenza appena presentata è errata e presenta uno stato di uscita (Unlock bit =0 ;Warning bit=1), o lo stato SL che è uno stato di blocco da cui si può uscire solamente tramite reset e che presenta uno stato di uscita costante (Unlock bit =0 ;Warning bit=1).

3 Architettura

3.1 Schema a blocchi con registri

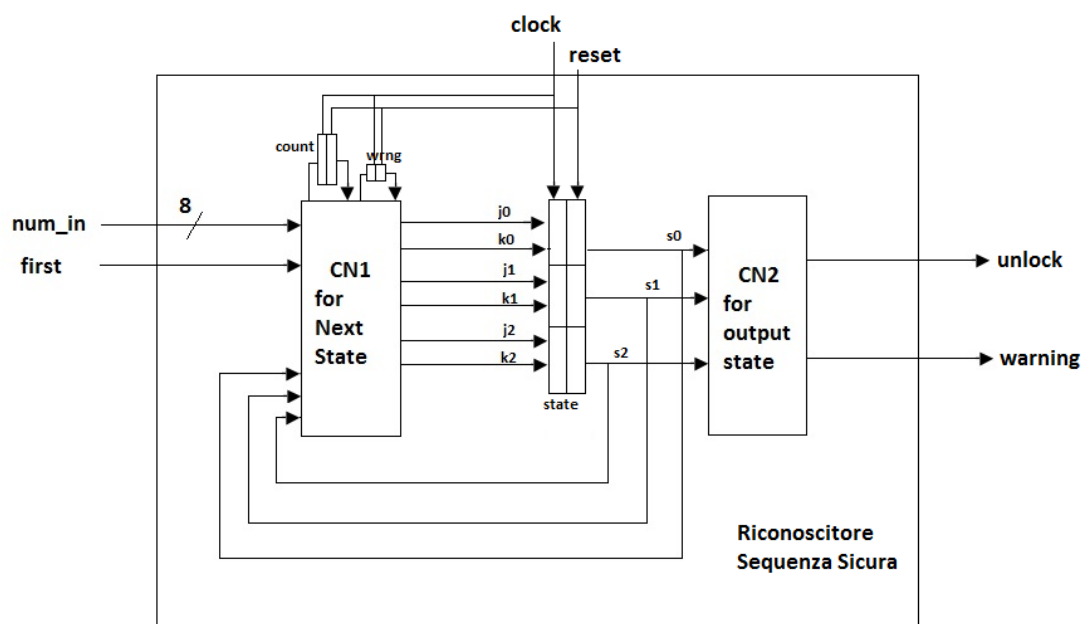
La figura sottostante mostra una visione funzionale del dispositivo, esso è composto da 3 registri interni: il registro State che tiene conto dello stato interno attuale (S0, S1), il registro Count che conta il numero di fallimenti nel riconoscere la sequenza presentata (dopo 3 fallimenti si va in stato locked), e infine il registro Wrng che è a 1 solo bit (quindi un solo FF-JK) che corrisponde ad un flag che si setta se un numero nella sequenza è errato (si attiva quando viene rilevato il primo numero errato nella sequenza), nello stato S4 sarà questo flag a determinare il passaggio allo stato SU (Unlocked - Sequenza riconosciuta) o a SW (Stato warning – Sequenza non riconosciuta).

Inoltre non è possibile direttamente collegare il pin warning al registro wrng perché il pin warning deve attivarsi per un solo ciclo di clock alla fine dell'acquisizione dell'intera sequenza, e non al primo numero errato nella sequenza.



3.2 Modello strutturale

Il modello strutturale interno del riconoscitore di sequenza viene mostrato in figura sottostante. Nel modello vengono codificati gli 8 stati interni disponibili su 3 bit: s0, s1, s2 che sono le uscite dei 3 flip flop jk che formano il registro di stato.



3.3 Codice VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity seq is
    generic (N : INTEGER:=8);
    port(clk : in std_logic;
         reset : in std_logic;
         first : in std_logic;
         num_in : in std_logic_VECTOR (N-1 downto 0);
         unlock : out std_logic;
         warning : out std_logic
        );
end seq;

architecture process_0 of seq is
    type state_type is (S0, S1, S2, S3, S4, SW, SL, SU);           --stati disponibili
    signal state : state_type;  --registro di stato
    signal count : integer range 0 to 3; --contatore dei fallimenti di riconoscimento ==> per 3 fail si va
    --nello stato di blocco SL
    signal wrng : std_logic; --flag che indica se c'è stato un errore di riconoscimento nella sequenza

begin

    Seq_rec: process (reset,clk) --processo che viene richiamato quando si verifica una variazione
    in reset o clk
    begin
        if (reset='1') then count <=0; state <=S0; wrng<='0';unlock<='0';warning<='0'; --azzerò
        tutte le uscite e i registri

        elsif (clk'EVENT and clk='1') then --cnt<=count;--campiona sul fronte di salita

        case state is

            --Stato S0 --
            when S0 => unlock<='0';warning<='0';
            if (num_in="00100100" and first='1') then state<=S1;    --00100100 == 36 --A1
            --prossimo ciclo di clock si passa allo stato S1
            elsif ((num_in>"00100100" or num_in<"00100100" ) and first='1' ) then state <=S1;
            wrng<='1';
            else state<=SL;
            end if;

            --Stato S1 --
            when S1 => state <=S2; unlock<='0';warning<='0';
            if (num_in="00010011" and first='0') then state <=S2;    --00010011==19
            else state <=S2 ; wrng<='1';
            end if;

            if(first='1') then state<=SL; end if; --se ho ancora il pin first attivo alto mi
            --blocco sullo stato locked SL al prossimo ciclo di clock

            --Stato S2 --
            when S2 => unlock<='0';warning<='0';
            if (num_in="00111000" and first='0') then state <=S3;    --00111000==56
            else state<=S3 ; wrng<='1';
            end if;

            if(first='1') then state<=SL; end if;

            --Stato S3 --
            when S3 =>unlock<='0';warning<='0';
            if (num_in="01100101" and first='0') then state <=S4;    --01100101==101
            else state <= S4 ; wrng<='1';
            end if;

            if(first='1') then state<=SL; end if;
```

della prossima sequenza

quando il pin reset non diventa attivo alto

```
--Stato S4 --
when S4 => unlock<='0';warning<='0';
if (num_in="01001001" and first='0' and wrng='0') then state <=SU; --01001001==73

else state <= SW; unlock<='0';warning<='1';
end if;

if(first='1') then state<=SL; end if;

--Stato SU => Unlocked --
when SU => unlock<='1';warning<='0';
if (num_in="00100100" and first='1') then state <=S1; --passo alla campionazione

elsif ((num_in>"00100100" or num_in<"00100100" ) and first='1') then state <= S1;
wrng<='1';
else state<=SL; warning<='1'; --se il bit di first non è a 1 mi blocco
end if;

--Stato SW => Warning --
when SW=> unlock<='0';warning<='1';
case count is

    when 0 to 2 =>
        if (num_in="00100100" and first='1') then state<=S1;count<=count+1;
        elsif((num_in>"00100100" or num_in<"00100100" ) and first='1') then
            state<=S1;wrng<='1'; count<=count+1;
        elsie state<=SL; warning<='1'; --se il bit di first non è a 1 mi blocco
        end if;

    when others => unlock<='0';warning<='1'; state<=SL;

end case;

--Stato SL => Locked --
when SL => state <=SL; unlock<='0';warning<='1';--mi blocco sullo stato SL fin

end case;
end if;
end process;

end;
```

3.4 Codice TestBench VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity seq_test is
end seq_test;

--}} End of automatically maintained section

architecture seq_test of seq_test is

component seq

    generic (N : INTEGER:=8);
    port(clk : in std_logic;
    reset : in std_logic;
    first : in std_logic;
    num_in: in std_logic_VECTOR (N-1 downto 0);
    unlock: out std_logic;
    warning: out std_logic);
end component seq;

-----
CONSTANT N      : INTEGER := 8;      -- Bus Width
CONSTANT MckPer : TIME    := 10 ms;  -- Master Clk period
CONSTANT TestLen : INTEGER := 340;    -- No. of Count (MckPer/2) for test

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

SIGNAL clk : std_logic := '0';
SIGNAL first : std_logic := '0';
SIGNAL x : std_logic_vector(N-1 downto 0):="00000000";
SIGNAL reset : std_logic:='0';

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

SIGNAL unlock : std_logic;
SIGNAL warning : std_logic;
SIGNAL clk_cycle : INTEGER;
SIGNAL Testing: Boolean := True;
--signal cnt: integer;

begin

    I : Seq PORT MAP (clk,reset,first,x,unlock,warning);

    -- Generates clk

clk    <= NOT clk AFTER MckPer/2 WHEN Testing ELSE '0';
```

```

begin
    I : Seq PORT MAP (clk,reset,first,x,unlock,warning);

    -- 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 3 => reset <= '1'; x<="00100100";
        WHEN 5 => reset <= '0'; x<="00100100";first<='1';
            WHEN 7 => reset <= '0'; x<="00010011";first<='0';
            WHEN 9 => reset <= '0'; x<="00111000";first<='0';
            WHEN 11 => reset <= '0'; x<="01100101";first<='0';
            WHEN 13 => reset <= '0'; x<="01001001";first<='0';

            --2° round
            WHEN 15 => reset <= '0'; x<="00100100";first<='1';
            WHEN 17 => reset <= '0'; x<="00110000";first<='0';
        WHEN 19 => reset <= '0'; x<="00100101";first<='0';
            WHEN 21 => reset <= '0'; x<="00110011";first<='0';
            WHEN 23 => reset <= '0'; x<="00111001";first<='0';
            WHEN 25 => reset <= '0'; x<="01100100";first<='1';
            WHEN 27 => reset <= '0'; x<="01001010";first<='0';
            WHEN 29 => reset <= '0'; x<="01101000";first<='0';
            WHEN 31 => reset <= '0'; x<="11001000";first<='0';
            WHEN 33 => reset <= '0'; x<="11001000";first<='0';
            WHEN 35 => reset <= '0'; x<="11001000";first<='1';
            WHEN 37 => reset <= '0'; x<="11001000";first<='0';
            WHEN 39 => reset <= '0'; x<="11001000";first<='0';
            WHEN 41 => reset <= '0'; x<="11001000";first<='0';
            WHEN 43 => reset <= '0'; x<="11001000";first<='0';
            WHEN 45 => reset <= '0'; x<="11001000";first<='1';
            WHEN 47 => reset <= '0'; x<="11001000";first<='0';
            WHEN 49 => reset <= '0'; x<="11001000";first<='0';

            WHEN (TestLen - 1) => Testing <= False;
            WHEN OTHERS => NULL;
        END CASE;

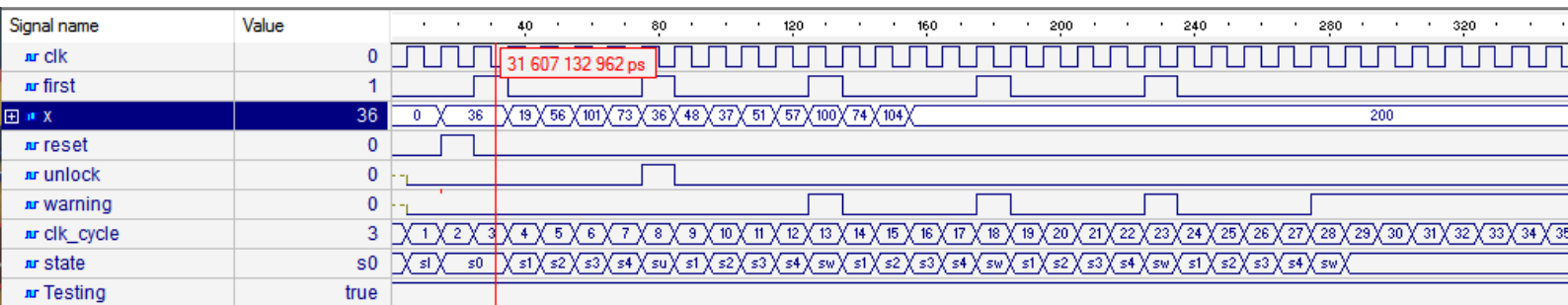
        count:= count + 1;
    END PROCESS Test_Proc;
end seq_test;

```

4 Test Plan

Ciò che è stato realizzato è un semplice riconoscitore di sequenza a 5 valori.

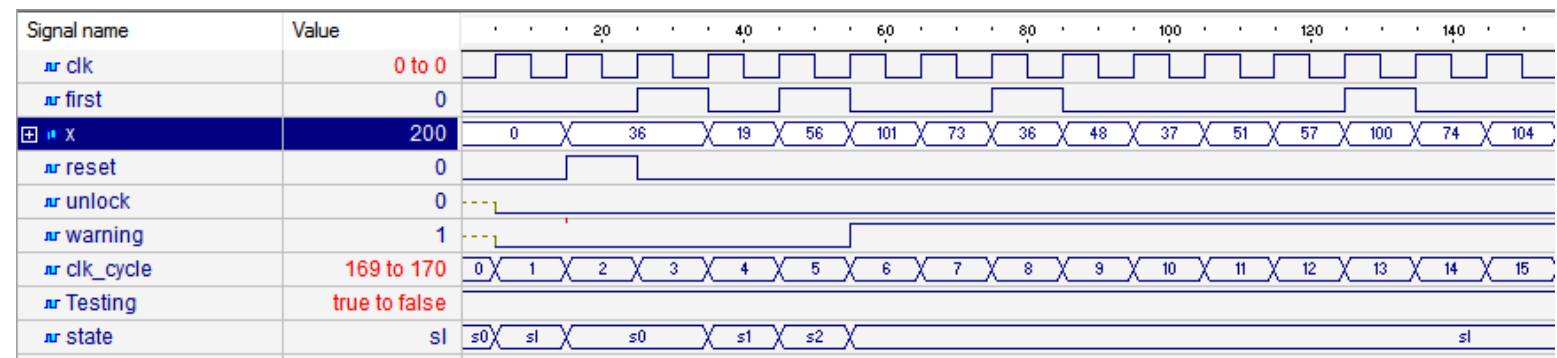
Nella figura sottostante viene rappresentato un diagramma che descrive il funzionamento del dispositivo al variare degli stati di ingresso.



Come si nota dal diagramma, dopo aver dato il reset (al successivo ciclo di clock), si inizia con il riconoscimento della sequenza, nel primo ciclo di clock successivo a quello di reset si deve avere il pin di first attivo alto e su num_in (X) il primo numero della sequenza.

Nell'esempio viene inserita la prima sequenza corretta e le successive errate, si nota infatti che dopo la prima sequenza si ha uno sblocco, infatti il bit di uscita unlock viene messo a 1 per il ciclo di clock immediatamente successivo a quello relativo all'ultima cifra della sequenza. Successivamente si hanno tre sequenze errate che portano a 3 warning e la quarta sequenza errata porta ad un blocco del dispositivo fino al prossimo reset.

Nell'esempio sottostante invece viene riportato il particolare caso di funzionamento nel quale il first bit diventa attivo alto in un ciclo di clock diverso a quello di campionamento del primo valore della sequenza, infatti diventa attivo per il 4° valore della sequenza ed essa non era ancora finita; si nota dunque che al ciclo di clock successivo si passa permanentemente allo stato di blocco SL con conseguente blocco del warning pin sullo stato di uscita '1'.



Il test appena effettuato non rappresenta però un test perfettamente affidabile dato che i risultati mostrati sono corretti solo per un sottoinsieme di tutti i possibili stati di ingresso, inoltre non si possono testare tutti i possibili stati di ingresso dato che le possibili sequenze sono ben:

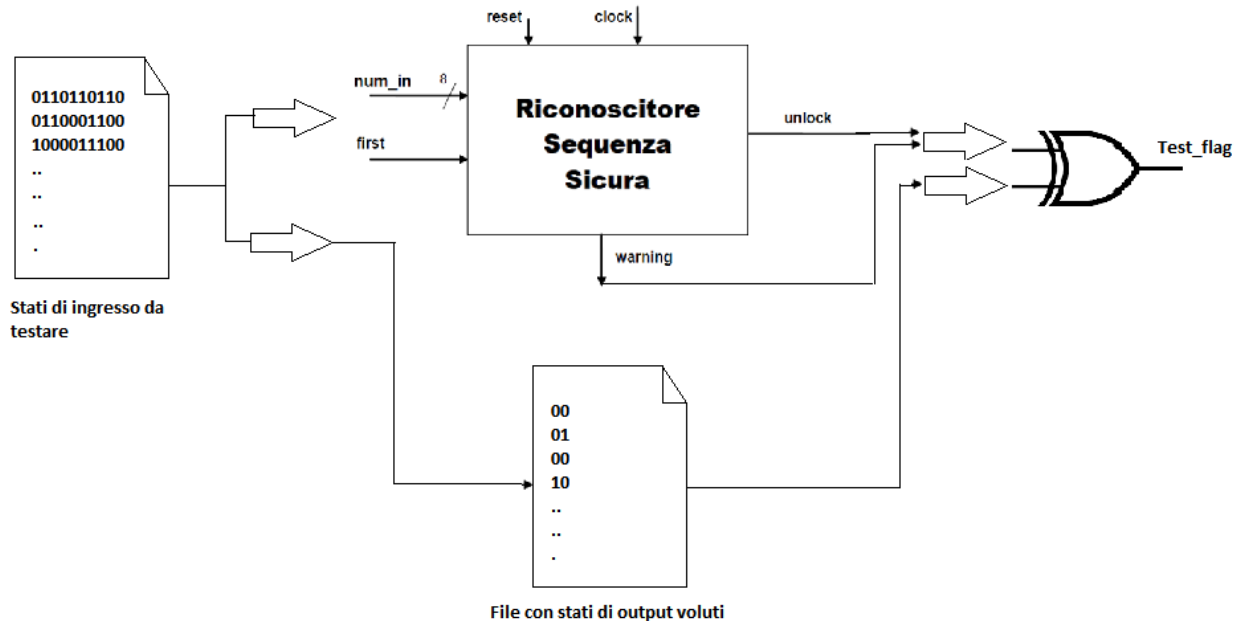
$$256^5 = 1.099.511.627.776$$

Quindi decisamente una quantità di possibili stati di ingresso eccessiva e non tutti testabili su un normale calcolatore.

Per questo procederemo con un approccio che si basa sul testare stati di ingresso generati in modo random in modo da coprire buona parte dei possibili stati di ingresso.

4.1 Architettura di Testing

Per testare il corretto funzionamento del dispositivo useremo l'architettura riportata in figura:

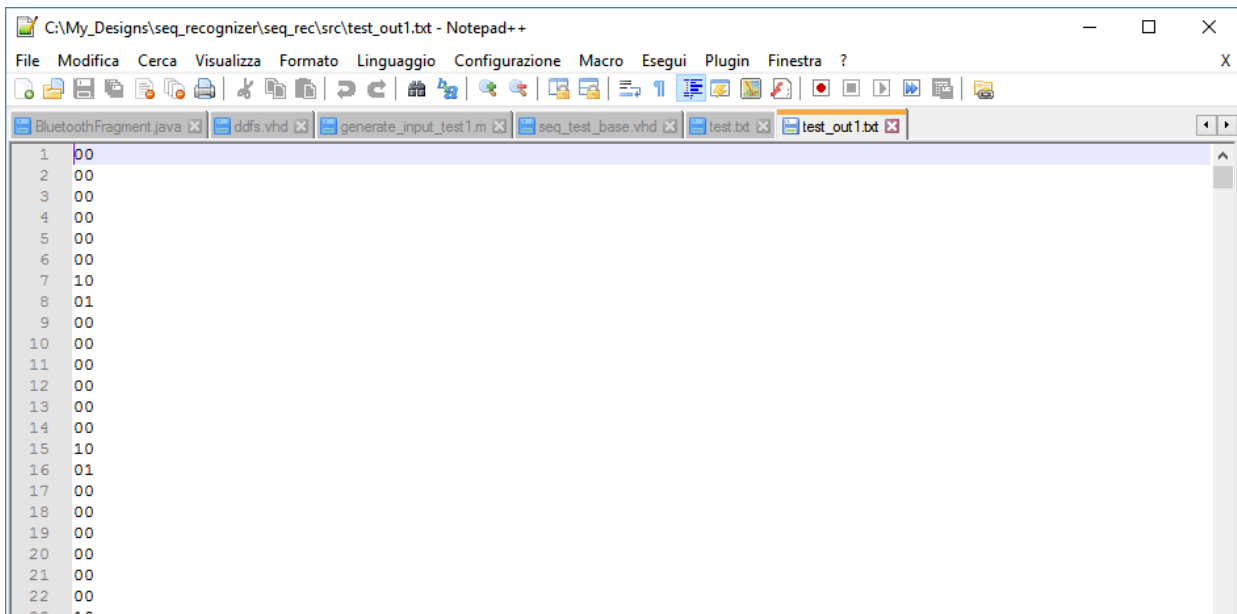


Verranno inizializzati due file quindi, nel primo vengono inseriti, tramite un semplice programmino scritto in Matlab, gli stati d'ingresso generati in modo random, nel secondo file gli stati di output che ci aspettiamo di trovare. Nelle immagini sottostanti è presente un'anteprima dei due file.

```

C:\My_Designs\Sequence_Recognizer\Seq\src\test.txt - Notepad++
File Modifica Cerca Visualizza Formato Linguaggio Configurazione Macro Esegui Plugin Finestra ?
BluetoothFragment.java ddfs.vhd generate_input_test1.m seq_test_base.vhd test.txt test_out1.txt
25 1100101000
26 1111010100
27 1010011100
28 0000100100
29 1101100101
30 1110111000
31 1010110110
32 1100000101
33 1011110100
34 0110010000
35 1010011100
36 0010110000
37 1011010001
38 0000100000
39 0100011100
40 0000110000
41 0001100100
42 1101001001
43 1011000100
44 0101000100
45 1111001000
46 0000100100
47 0111000001
48 0110000100
49 1100001100
50 1100101100
51 0011000000
52 0111110101
53 0111001000
54 1010010110

```



The screenshot shows a Notepad++ window with the title bar 'C:\My_Designs\seq_recognizer\seq_rec\src\test_out1.txt - Notepad++'. The menu bar includes File, Modifica, Cerca, Visualizza, Formato, Linguaggio, Configurazione, Macro, Esegui, Plugin, Finestra, and ?. The toolbar contains various icons for file operations. The tab bar shows several open files: BluetoothFragment.java, ddfs.vhd, generate_input_test1.m, seq_test_base.vhd, test.bt, and test_out1.txt. The main text area displays a list of 22 lines of binary data, each preceded by a line number from 1 to 22. The sequence is: 1 00, 2 00, 3 00, 4 00, 5 00, 6 00, 7 10, 8 01, 9 00, 10 00, 11 00, 12 00, 13 00, 14 00, 15 10, 16 01, 17 00, 18 00, 19 00, 20 00, 21 00, 22 00.

Il test genera più di 2500 numeri casuali (stati di ingresso) per un totale di 500 sequenze, la variabile test_flag in uscita dallo XOR comparatore resta nello stato Undefined se il test è stato superato con successo, mentre viene settata ad uno nel caso di Fallimento.

Il test ha sempre dato esito positivo per i 2500 cicli di clock testati.

Di seguito viene riportato lo script matalab per la generazione di stati di ingresso.

Gli stati di uscita corretti da far comparare a quelli da testare sono invece stati scritti manualmente in quanto si tratta di una forma d'onda periodica.

```
function generate_input_test1()
    mkdir('C:\My_Designs\seq_recognizer\seq_rec\src'); %creo la cartella log
    id=fopen('C:\My_Designs\seq_recognizer\seq_rec\src\test.txt' , 'wt');
    %apro il file test.txt e se non presente lo creo, ritorno il suo file
    descriptor in id
    number_it=2500;
    count=0;

    fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'10');

    c=round(rand(1)*255);
    str = dec2bin(c,8);
    fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'01');

    for i = 1:number_it

        if (mod(i,22)==0)
            fprintf(id,'%s%s\n',dec2bin(round(rand(1)*255),8),'10');
            fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'01');count=count+2;
        elseif (mod(i-count,5)==0)
            fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'01');
        else fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'00');
        end
    end

end
```

Per poter usare l'architettura di testing sopra illustrata, viene usato il seguente file VHDL.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE std.textio.all;
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;
library std;
use std.textio.all;

entity seq_test is
end seq_test;

--}} End of automatically maintained section

architecture seq_test of seq_test is

component seq

    generic (N : INTEGER:=8);
    port(clk : in std_logic;
    reset : in std_logic;
    first : in std_logic;
    num_in: in std_logic_VECTOR (N-1 downto 0);
    unlock: out std_logic;
    warning: out std_logic);
end component seq;

-----
CONSTANT N      : INTEGER := 8;      -- Bus Width
CONSTANT MckPer : TIME     := 10 ms; -- Master Clk period
CONSTANT TestLen: INTEGER  := 5000;  -- No. of Count (MckPer/2) for test

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

SIGNAL  clk      : std_logic := '0';
SIGNAL  first    : std_logic := '0';
SIGNAL  x        : std_logic_vector(N-1 downto 0) := "00000000";
SIGNAL  reset    : std_logic:= '0';

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

SIGNAL  unlock    : std_logic;
SIGNAL  warning   : std_logic;
SIGNAL  clk_cycle : INTEGER;
SIGNAL  Testing: Boolean := True;
SIGNAL  unlock_t  : std_logic;
SIGNAL  warning_t  : std_logic;
SIGNAL  test      : std_logic;
--signal cnt: integer;

begin

    I : Seq PORT MAP(clk,reset,first,x,unlock,warning);

    -- 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;
    file vec_file: text is in "test.txt";
    file out_t: text is in "test_out1.txt";

```



```

variable buf_in: line;
variable out_line: line;
variable testv: integer;
variable v_read: std_logic_vector(0 to 9);
variable test_out_read: std_logic_vector(0 to 1);
BEGIN

    -----
    clk_cycle <= (count+1)/2;

    if not endfile (vec_file) then

        if (count mod 2)>0 then
            readline (vec_file, buf_in);
            readline (out_t, out_line);
            read(buf_in,v_read);
            read(out_line,test_out_read);
            --num_in
            x(0)<=v_read(7);
            x(1)<=v_read(6);
            x(2)<=v_read(5);
            x(3)<=v_read(4);
            x(4)<=v_read(3);
            x(5)<=v_read(2);
            x(6)<=v_read(1);
            x(7)<=v_read(0);
            --reset in
            reset<=v_read(8);
            --first in
            first<=v_read(9);
            unlock_t<=test_out_read(0);
            warning_t<=test_out_read(1);
            test<= (unlock_t xor unlock) or (warning_t xor warning) or test;
        end if;

    end if;

```

4.2 Code Coverage

Un altro aspetto critico, e di fondamentale importanza nel test del codice VHDL, per un dispositivo elettronico, è quello del code coverage, ossia vedere quanta parte di codice viene usata nella simulazione, raggiungere ogni parte del codice nella simulazione infatti è utile per capire se ci sono parti superflue nel dispositivo o se una parte può essere non raggiungibile per un errore nel codice.

Per effettuare il code coverage viene usato il tool Code Coverage Viewer di Aldec Acitve-HDL

di seguito le immagini rappresentative del test.

Branch Coverage for selected item and all its children



- Branches executed, including children
- Branches not executed, including children

Branch count: 50

Branches executed, including children: 37 (74.00 %)

Branches not executed, including children: 13 (26.00 %)

Come si può notare nella simulazione con gli stati di ingresso generati dallo script matlab, solo il 74% del codice viene raggiunto, non è possibile stabilire con certezza quindi se il codice è esente da errori o se qualche parte di codice è superflua.

Nell'immagine sottostante una parte di codice dove le righe verdi sono quelle raggiunte dalla simulazione e quelle rosse, quelle non raggiunte.

Line	Count	BC	EC	Source
37	111*	1t 109f		<code>if num_in='00100100' and first='1' then state<=S1; --00100100==36 --Al prossimo ciclo di clock si passa lo stato</code>
38	325*	108t 1f		<code>elsif (num_in='00100100' or num_in='00100100') and first='1' then state <=S1 ; wmg<= '1' ;</code>
39	1			<code>else state<=SL;</code>
40	110			<code>end if;</code>
41	None			
42	None			<code>--Stato S1--</code>
43	1305*	435t		<code>when S1 => state <=S2; unlock<='0' waming<='0' ;</code>
44	436*	1t 434f		<code>if num_in='00010011' and first='0' then state <=S2; --00010011==15</code>
45	868*			<code>else state <=S2 ; wmg<= '1' ;</code>
46	435			<code>end if;</code>
47	None			
48	870*	0t 435f		<code>if first='1' then state<=SL waming<='1'; end if; --se ho ancora il pin first attivo allo mi blocca sullo stato locked SL al pros</code>
49	None			
50	None			<code>--Stato S2--</code>
51	870*	435t		<code>when S2 => unlock<='0' waming<='0' ;</code>
52	436*	1t 434f		<code>if num_in='00111000' and first='0' then state <=S3; --00111000==56</code>
53	868*			<code>else state<=S3 ; wmg<= '1' ;</code>
54	435			<code>end if;</code>
55	None			
56	870*	0t 435f		<code>if first='1' then state<=SL waming<='1'; end if;</code>
57	None			
58	None			<code>--Stato S3--</code>
59	870*	435t		<code>when S3 =>unlock<='0' waming<='0' ;</code>
60	437*	2t 433f		<code>if num_in='01100101' and first='0' then state <=S4; --01100101==101</code>
61	866*			<code>else state <= S4 ; wmg<= '1' ;</code>
62	435			<code>end if;</code>
63	None			
64	870*	0t 435f		<code>if first='1' then state<=SL waming<='1'; end if;</code>
65	None			
66	None			<code>--Stato S4--</code>
67	868*	434t		<code>when S4 => unlock<='0' waming<='0' ;</code>
68	434*	0t 434f		<code>if num_in='01001001' and first='0' and wmg='0' then state <=SU; unlock<='1' waming<='0' ; --01001001==73</code>
69	1302*			<code>else state <= SW unlock<='1' waming<='1' ;</code>
70	434			<code>end if;</code>
71	None			
72	868*	0t 434f		<code>if first='1' then state<=SL waming<='1'; end if;</code>
73	None			
74	None			<code>--Stato SU => Unlocked--</code>
75	0*	0t		<code>when SU => unlock<='0' waming<='0' ;</code>
76	0*	0t 0f		<code>if num_in='00100100' and first='1' then state <=S1; --passo alla campionazione della prossima sequenza</code>
77	0*	0t 0f		<code>elsif (num_in='00100100' or num_in='00100100') and first='1' then state <= S1 wmg<='1' ;</code>
78	0*			<code>else state<=SL waming<='1' ; --se il bit di first non è a 1 mi blocca</code>

Com'è possibile intuire bisogna estendere gli stati di ingresso presentati, aggiungendone altri che permettono di raggiungere tutte le parti del codice durante la simulazione.

Per questo useremo lo script matlab modificato, modificando di conseguenza anche lo stato di uscita voluto.

Ecco lo script matlab modificato per coprire al 100% il codice VHDL durante la simulazione.

```
function generate_input_test1()
    mkdir('C:\My_Designs\seq_recognizer\seq_rec\src'); %creo la cartella log
    id=fopen('C:\My_Designs\seq_recognizer\seq_rec\src\test.txt' , 'wt'); %apro
    il file test.txt e se non presente lo creo, ritorno il suo file descriptor in id
    number_it=2500;
    count=0;

    fprintf(id,'%s \n','0010010010');
    fprintf(id,'%s \n','0010010001');
    fprintf(id,'%s \n','0001001100');
    fprintf(id,'%s \n','0011100000');
    fprintf(id,'%s \n','0110010100');
    fprintf(id,'%s \n','0100100100');
    fprintf(id,'%s \n','0010010000');
    fprintf(id,'%s \n','0010010000');

    fprintf(id,'%s \n','0010010010');
    fprintf(id,'%s \n','0010010001');
    fprintf(id,'%s \n','0001001100');
    fprintf(id,'%s \n','0011100000');
    fprintf(id,'%s \n','0110010100');
    fprintf(id,'%s \n','0100100100');
    fprintf(id,'%s \n','0010010000');
    fprintf(id,'%s \n','0010010000');

    fprintf(id,'%s \n','0010010010');
    fprintf(id,'%s \n','0010010001');
    fprintf(id,'%s \n','0001001100');
    fprintf(id,'%s \n','0011100000');
    fprintf(id,'%s \n','0110010100');
    fprintf(id,'%s \n','0100100100');
    fprintf(id,'%s \n','0010000001');
    fprintf(id,'%s \n','0010010000');

    fprintf(id,'%s \n','0010010010');
    fprintf(id,'%s \n','0010010001');
    fprintf(id,'%s \n','0000001100');
    fprintf(id,'%s \n','0011100000');
    fprintf(id,'%s \n','0110010100');
    fprintf(id,'%s \n','0100100100');
    fprintf(id,'%s \n','0010000000');
    fprintf(id,'%s \n','0010010000');

    %%

    fprintf(id,'%s \n','0010010010');
    fprintf(id,'%s \n','0010010001');
    fprintf(id,'%s \n','0001001101');
    fprintf(id,'%s \n','0011100000');
    fprintf(id,'%s \n','0110010100');
    fprintf(id,'%s \n','0100100100');
```

```

fprintf(id,'%s \n','0010010000');
fprintf(id,'%s \n','0010010000');

fprintf(id,'%s \n','0010010010');
fprintf(id,'%s \n','0010010001');
fprintf(id,'%s \n','0001001100');
fprintf(id,'%s \n','0011100001');
fprintf(id,'%s \n','0110010100');
fprintf(id,'%s \n','0100100100');
fprintf(id,'%s \n','0010010000');
fprintf(id,'%s \n','0010010000');

fprintf(id,'%s \n','0010010010');
fprintf(id,'%s \n','0010010001');
fprintf(id,'%s \n','0001001100');
fprintf(id,'%s \n','0011100000');
fprintf(id,'%s \n','0110010101');
fprintf(id,'%s \n','0100100100');
fprintf(id,'%s \n','0010010000');
fprintf(id,'%s \n','0010010000');

fprintf(id,'%s \n','0010010010');
fprintf(id,'%s \n','0010010001');
fprintf(id,'%s \n','0001001100');
fprintf(id,'%s \n','0011100000');
fprintf(id,'%s \n','0110010100');
fprintf(id,'%s \n','0100100101');
fprintf(id,'%s \n','0010010000');
fprintf(id,'%s \n','0010010000');

fprintf(id,'%s \n','0010010010');
fprintf(id,'%s \n','0010010001');
fprintf(id,'%s \n','0001001100');
fprintf(id,'%s \n','0011100000');
fprintf(id,'%s \n','0110010100');
fprintf(id,'%s \n','0100100100');
fprintf(id,'%s \n','0010010001');
fprintf(id,'%s \n','0010010000');

fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'10');

c=round(rand(1)*255);
str = dec2bin(c,8);
fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'01');

for i = 1:number_it

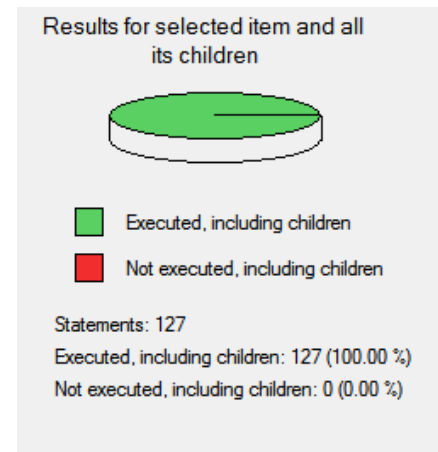
    if (mod(i,22)==0) fprintf(id,'%s%s
\n',dec2bin(round(rand(1)*255),8),'10'); fprintf(id,'%s%s
\n',dec2bin(round(rand(1)*255),8),'01');count=count+2;
    elseif (mod(i-count,5)==0) fprintf(id,'%s%s
\n',dec2bin(round(rand(1)*255),8),'01');
    else fprintf(id,'%s%s \n',dec2bin(round(rand(1)*255),8),'00');
    end
end

end

```

Nelle figure sottostanti si ha il risultato del secondo Code Coverage Test.

Come si può notare la simulazione copre tutto il codice, ovvero ogni statement del codice viene eseguito, il che significa che ogni parte del codice è raggiungibile e che non ci sono parti superflue.



Code Coverage Viewer - C:/My_Designs/seq_recognizer/seq_rec/coverage/results.ccl					
File Edit View Help					
All visible					
Browse By: Blocks					
Hierarchy	CC [%]	Seq.vhd	Details		
Root : seq_rec.Dummy top-level...		Line	Count	BC	EC
seq_test : seq_rec.seq_test ...	10	34	None		
seq_rec.seq (process_0)	10	35	None		
Test_Proc	10	36	234*	117t	
line__59	10	37	128*	11t 106f	
		38	316*	105t 1f	
		39	1		
		40	117		
		41	None		
		42	None		
		43	1305*	435t	
		44	444*	9t 426f	
		45	852*		
		46	435		
		47	None		
		48	872*	1t 434f	
		49	None		
		50	None		
		51	860*	430t	
		52	439*	9t 421f	
		53	842*		
		54	430		
		55	None		
		56	862*	1t 429f	
		57	None		
		58	None		
		59	856*	428t	
		60	436*	8t 420f	
		61	840*		
		62	428		
		63	None		
		64	858*	1t 427f	
		65	None		
		66	None		
		67	854*	427t	
		68	442*	5t 422f	
		69	1266*		
Source					
--State S0--					
when S0 => unlock<= 0; warning<= 0;					
if num_in= '00100100' and first= 0 then state<=S1; --00					
elsif (num_in= '00100100' or num_in<= '00100100') and fir					
else state<=SL;					
end if;					
--State S1--					
when S1 => state <=S2; unlock<= 0; warning<= 0;					
if num_in= '00100100' and first= 0 then state <=S2; --					
else state <=S2; wmg<= 1;					
end if;					
if first= 1 then state<=SL; warning<= 0; end if; --se ho anco					
--State S2--					
when S2 => unlock<= 0; warning<= 0;					
if num_in= '00100100' and first= 0 then state <=S3; --00					
else state<=S3; wmg<= 1;					
end if;					
if first= 1 then state<=SL; warning<= 0; end if;					
--State S3--					
when S3 => unlock<= 0; warning<= 0;					
if num_in= '00100100' and first= 0 then state <=S4; --01					
else state <= S4; wmg<= 1;					
end if;					
if first= 1 then state<=SL; warning<= 0; end if;					
--State S4--					
when S4 => unlock<= 0; warning<= 0;					
if num_in= '00100100' and first= 0 and wmg= 0 then stat					
else state <= SW unlock<= 0; warning<= 1;					

Il dispositivo pertanto risulta funzionare correttamente, e i test eseguiti danno tutti esito positivo.

5 Sintesi

Questo capitolo rappresenta l'ultima fase della progettazione del riconoscitore di sequenza, in questa fase di sintesi useremo il tool Xilinx ISE per la sintesi automatica.

Questo tool infatti, a partire dal codice sorgente in VHDL, effettua la traduzione e i relativi passaggi che servono per creare fisicamente il dispositivo su un FPGA.

Per la sintesi viene usata la Zybo Board, essa è una scheda elettronica programmabile e proprietaria di Xilinx inc.

Al suo interno la Zybo Board contiene una FPGA di 4.400 slice.

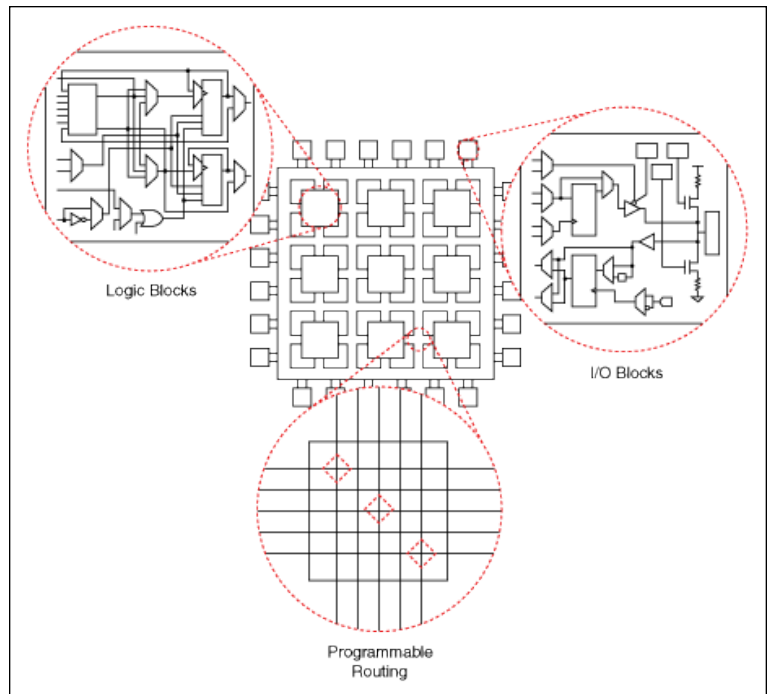
5.1 Architettura base di un FPGA

Un FPGA è composta da tanti blocchi chiamati CLB (Configurable Logic Block) che sono programmabili e definiscono le operazioni logiche, gli I/O Block che essendo anch'esse programmabili definiscono le operazioni di I/O.

Inoltre i CLB sono formati da Flip-Flop e dalle Look-up-table (LUTs).

I FF sono dei registri a un bit e possono trovarsi in uno dei due possibili stati (Set or Reset).

Le LUT invece sono dei blocchi logici che memorizzano una predefinita lista di output per ogni combinazione possibile di input (è una rete combinatoria dunque).



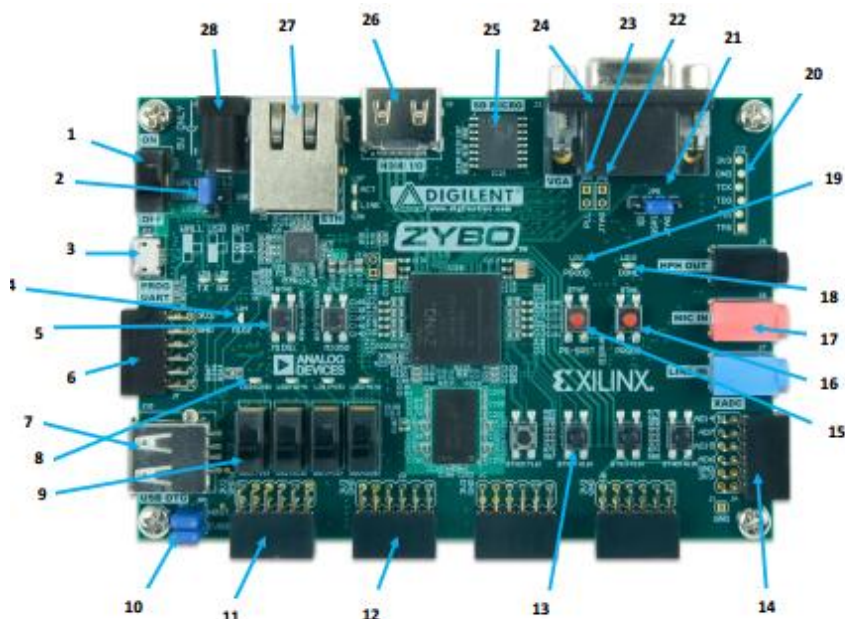
I CLB nell'FPGA sono inoltre suddivisi in slice (dei piccoli blocchi logici), ogni slice contiene un set di LUTs e FF: nella Zybo board per esempio sono presenti 4.400 slices dove ogni slice contiene 4 LUTs con input a 6-bit e 8 FF registers.

5.2 Struttura della Zybo Board

Dalla documentazione ufficiale della Zybo Board si possono ricavare le caratteristiche, di sotto alcune immagini che rappresentano la scheda.

Per la sintesi useremo la porta di ingresso Standard Pmod a 12 pin di cui 2 utilizzati per l'alimentazione VCC e 2 per la massa GND. (Porta num. 6)

Come output useremo i led num. 8 rappresentati in figura. Assegneremo un led per il pin di warning e uno per il pin di unlock.



Porta Standard Pmod:

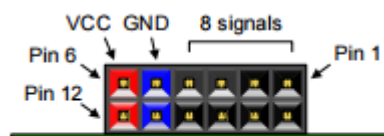


Figure 16. Pmod diagram.

Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

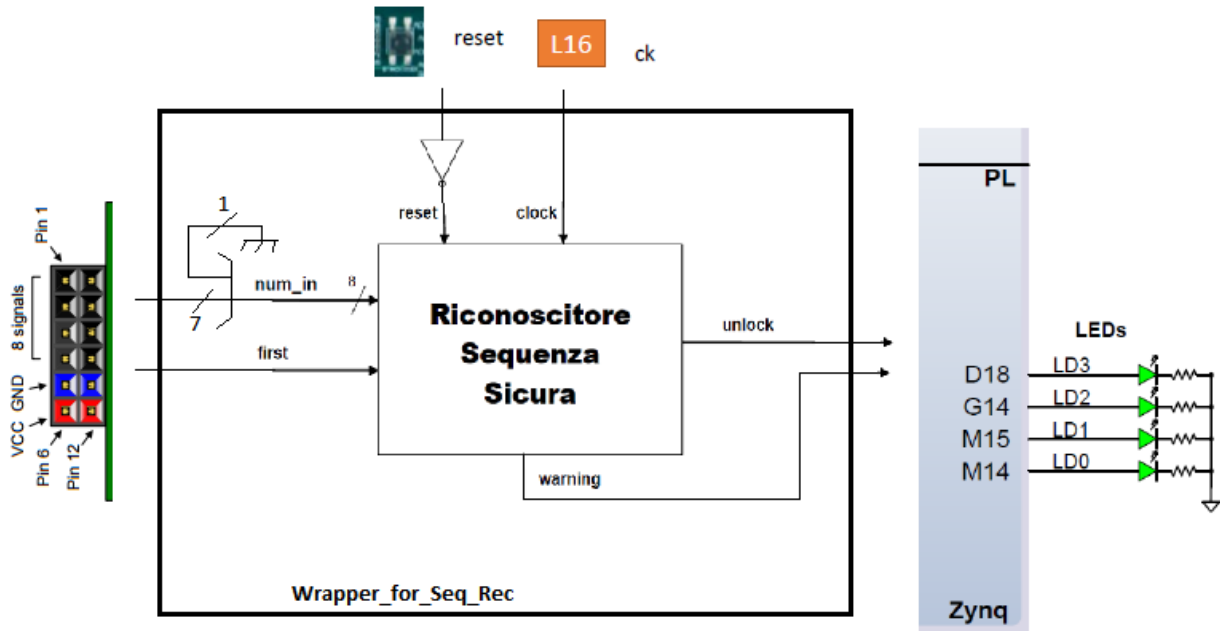
Configurazione pin della porta standard Pmod per la parte PL (Programmable Logic) dell'FPGA.

Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Porta usata da noi

5.3 Wrapper per il riconoscitore di sequenza

La Zybo Board ha la caratteristica di avere il reset attivo basso (si resetta da 1 -> 0) e inoltre tramite la porta Pmod standard si hanno solo 8 pin d'ingresso pilotabili, quindi bisogna porre il bit più significativo di num_in del riconoscitore a 0 per poter usare i 9 pin dello stato d'ingresso.



Codice VHDL del wrapper:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity wrapper is
    generic (N : INTEGER:=8);
    port(clk_1 : in std_logic;
        reset_1 : in std_logic;
        first_1 : in std_logic;
        num_in_1: in std_logic_VECTOR (N-2 downto 0);
        unlock_1: out std_logic;
        warning_1: out std_logic
    );
end wrapper;

architecture wrapper of wrapper is
    component Seq
        port(clk : in std_logic;
            reset : in std_logic;
            first : in std_logic;
            num_in: in std_logic_VECTOR (N-1 downto 0);
            unlock: out std_logic;
            warning: out std_logic
        ); end component Seq;

```



```

signal not_reset: std_logic;
signal num_in_adapter: std_logic_vector (N-1 downto 0);

begin
    num_in_adapter(7) <= '0';
    num_in_adapter(6 downto 0) <= num_in_1(6 downto 0);
    not_reset <= not reset_1;
    L1: Seq port
map(clk_1, not_reset, first_1, num_in_adapter, unlock_1, warning_1);


end wrapper;

```

Durante la sintesi nel tool Xilinx Ide viene riportato un solo warning:

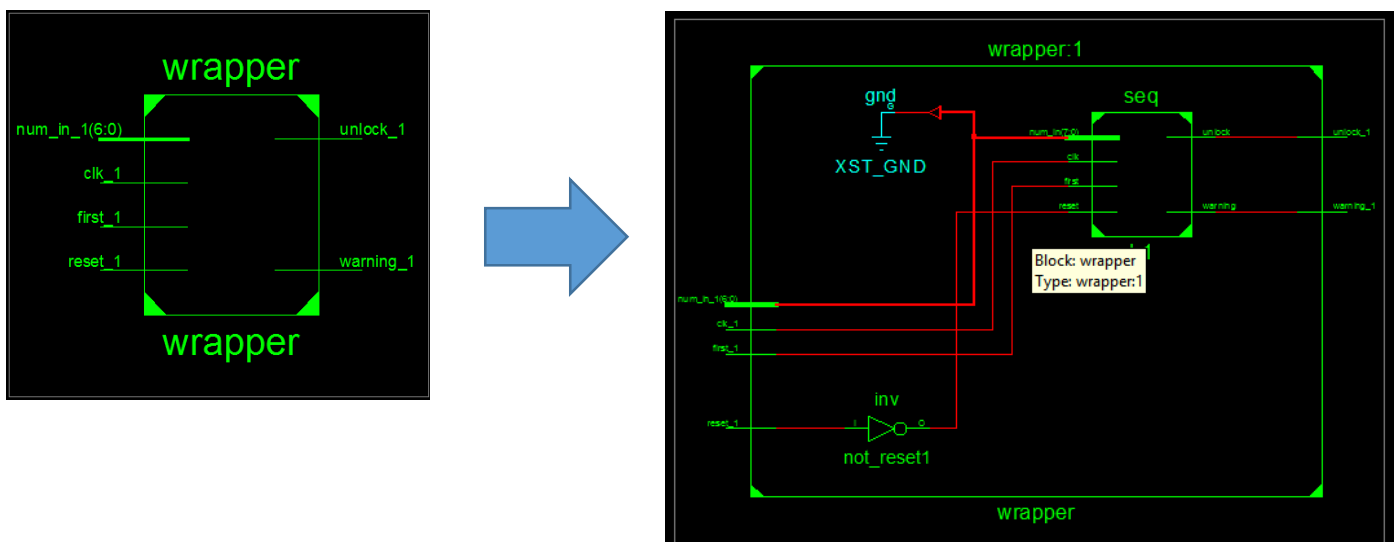
wrapper Project Status (02/10/2016 - 16:21:54)			
Project File:	Sequence_recognizer.xise	Parser Errors:	No Errors
Module Name:	wrapper	Implementation State:	Placed and Routed
Target Device:	xc7z010-3dkg400	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	1 Warning (0 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Il warning è il seguente:

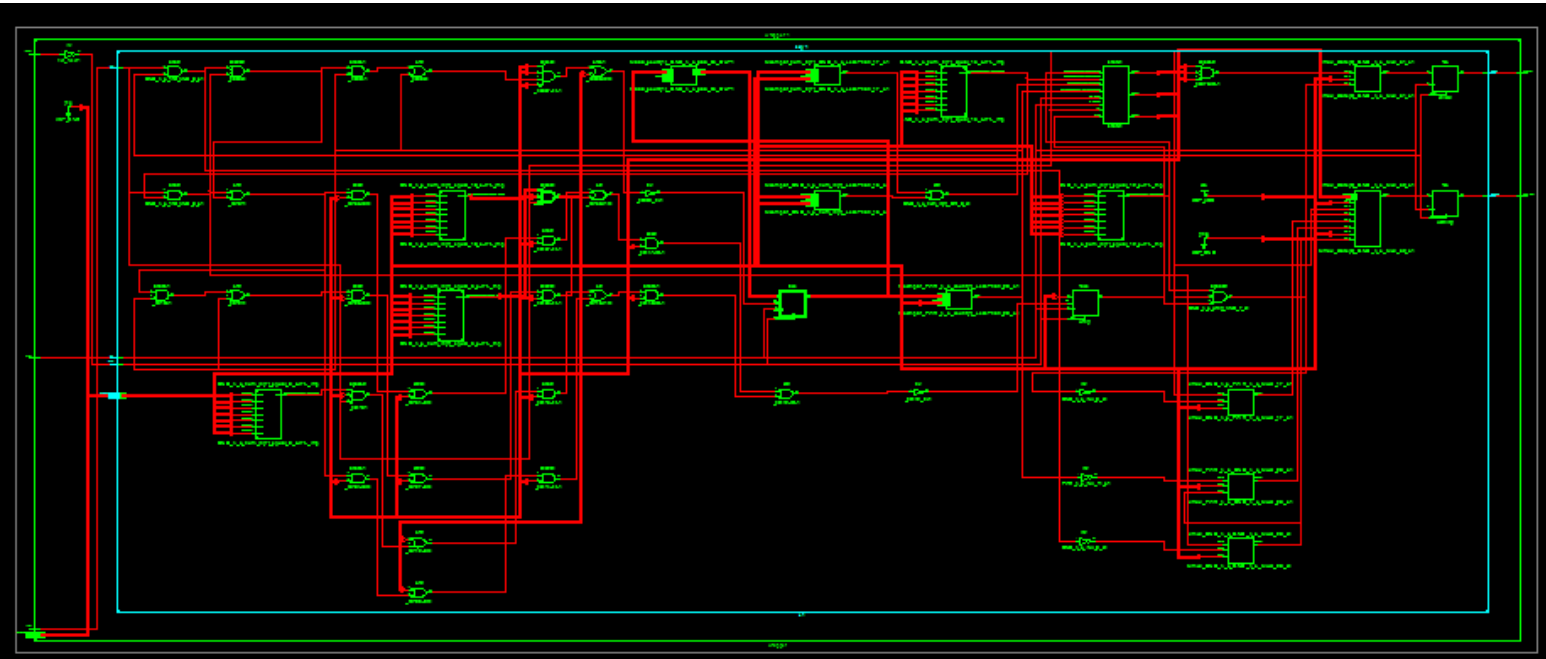
map		PhysDesignRules:2500 - This design does not have a PS7 block. Instantiate the PS7 block in order to ensure proper fabric tie-offs and correct operation of the processing system7.
-----	---	--

e sta ad indicare che nella sintesi del dispositivo non viene usata la parte PS (Processing system) dell'FPGA, cosa di cui siamo ben consapevoli.

Sezione Synthesize -XST (View RTL Schematic)



Visione globale di View RTL Schematic:



5.4 Spazio occupato e frequenza massima:

Dal Design Overview summary si può notare che lo spazio occupato sull'FPGA è veramente minimo, infatti di un totale di 4.400 slice disponibili sull'FPGA, il nostro dispositivo ne utilizza solo 7 (meno dell'1%), utilizza inoltre 19 LUTs e 8 Flip-Flop.

Device Utilization Summary					[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Registers	8	35,200	1%		
Number used as Flip Flops	8				
Number used as Latches	0				
Number used as Latch-thrus	0				
Number used as AND/OR logics	0				
Number of Slice LUTs	19	17,600	1%		
Number used as logic	19	17,600	1%		
Number using O6 output only	14				
Number using O5 output only	0				
Number using O5 and O6	5				
Number used as ROM	0				
Number used as Memory	0	6,000	0%		
Number used exclusively as route-thrus	0				
Number of occupied Slices	7	4,400	1%		
Number of LUT Flip Flop pairs used	19				
Number with an unused Flip Flop	13	19	68%		
Number with an unused LUT	0	19	0%		
Number of fully used LUT-FF pairs	6	19	31%		
Number of unique control sets	2				

La frequenza massima raggiungibile dal nostro riconoscitore di sequenza è invece di circa 683MHz come riportato nelle specifiche di sotto.

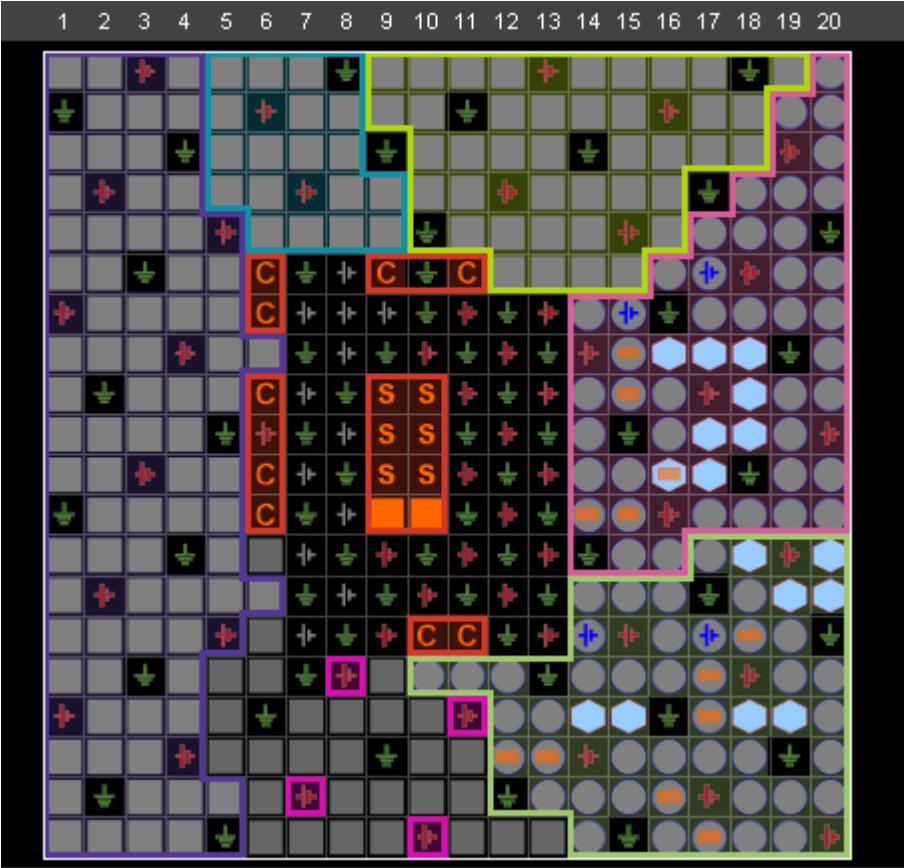
Timing Summary:

Speed Grade: -3

Minimum period: 1.463ns (Maximum Frequency: 683.434MHz)
Minimum input arrival time before clock: 2.005ns
Maximum output required time after clock: 0.511ns
Maximum combinational path delay: No path found

Specify I/O Pin – PlanAhead

In questa sezione vengono mappati i pin di I/O della Zybo board ai rispettivi pin di I/O del riconoscitore di sequenza, in modo da renderlo utilizzabile.

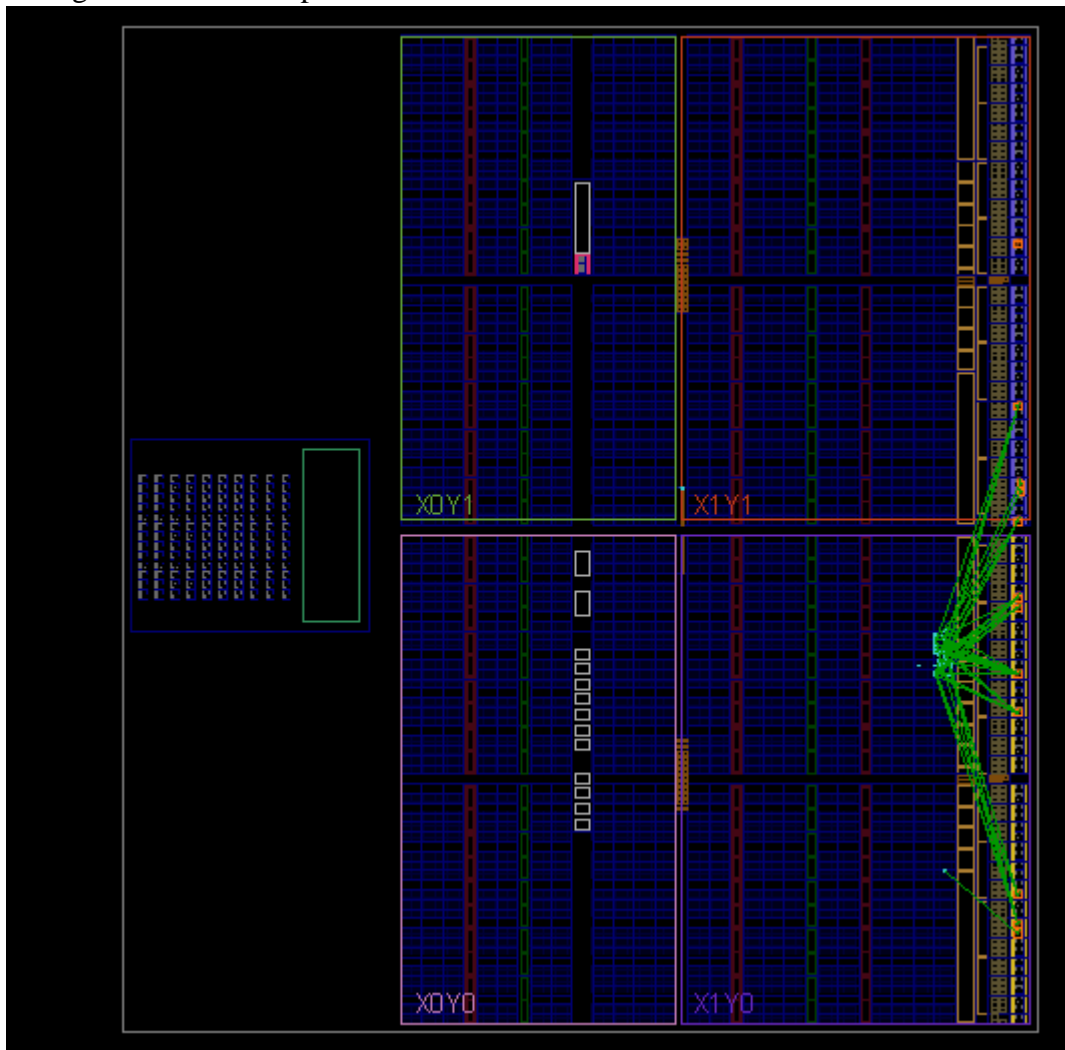


Voltaggio e mappaggio delle porte:

I/O Ports											
Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Stre...	Slew Type	Pull Type
All ports (12)											
num_in_1 (7)	Input					LVC MOS33*	3.300				NONE
num_in_1[6]	Input		T17			34 LVC MOS33*	3.300				NONE
num_in_1[5]	Input		U17			34 LVC MOS33*	3.300				NONE
num_in_1[4]	Input		V13			34 LVC MOS33*	3.300				NONE
num_in_1[3]	Input		H15			35 LVC MOS33*	3.300				NONE
num_in_1[2]	Input		J15			35 LVC MOS33*	3.300				NONE
num_in_1[1]	Input		W16			34 LVC MOS33*	3.300				NONE
num_in_1[0]	Input		V12			34 LVC MOS33*	3.300				NONE
Scalar ports (5)											
clk_1	Input		L16			35 LVC MOS33*	3.300				NONE
first_1	Input		Y17			34 LVC MOS33*	3.300				NONE
reset_1	Input		R18			34 LVC MOS33*	3.300				NONE
unlock_1	Output		M15			35 LVC MOS33*	3.300		12 SLOW		NONE
warning_1	Output		M14			35 LVC MOS33*	3.300		12 SLOW		NONE

Physical Placement on PlanAhead:

Di seguito come viene posizionato fisicamente il riconoscitore nell'FPGA.



5.5 Generazione del programming file

Questa è l'ultima tappa della sintesi del riconoscitore di sequenza, in questa fase viene generati il bitstream, ossia tutte le istruzioni da dare alla Zybo board per programmare l'FPGA.

```
Started : "Generate Programming File".  
Running bitgen...  
Command Line: bitgen -intstyle ise -f wrapper.ut wrapper.ncd
```



```
DRC detected 0 errors and 1 warnings. Please see the previously displayed  
individual error or warning messages for more details.  
Creating bit map...  
Saving bit stream in "wrapper.bit".  
Bitstream generation is complete.
```