

# Introduzione al VHDL

Claudio Pagnamenta & Simone Buvoli  
responsabile: Franco Crivelli

# Indice:

<b>1. Introduzione .....</b>	<b>p. 03</b>
1.1 Cos'è il VHDL .....	p. 04
1.2 Breve storia .....	p. 04
1.3 Perché utilizzare VHDL.....	p. 04
<b>2. I concetti del linguaggio VHDL.....</b>	<b>p. 05</b>
2.1 I segnali.....	p. 06
2.1.1 Breve definizione di variabile .....	p. 06
2.1.2 Breve definizione di segnale .....	p. 06
2.1.3 I differenti tipi di segnali.....	p. 07
2.1.3.1 Il tipo bit.....	p. 07
2.1.3.2 Il tipo std_logic.....	p. 08
2.1.3.3 Il vettore .....	p. 08
2.1.3.4 Assegnazione del valore in un segnale o in un vettore.....	p. 08
2.2 I moduli.....	p. 09
2.2.1 L'entità.....	p. 09
2.2.1.1 Sintassi generica dell'entità di un modulo .....	p. 09
2.2.1.2 Esempi di un'entità.....	p. 10
2.2.2 L'architettura.....	p. 11
2.2.2.1 La sintassi generica di un'architettura.....	p. 11
<b>3. Le istruzioni elementari .....</b>	<b>p. 12</b>
3.1 Le istruzioni sequenziali.....	p. 13
3.2 L'istruzione scelta (case) .....	p. 13
3.3 L'istruzione condizionale (if) .....	p. 14
<b>4. Esempi di programmi.....</b>	<b>p. 15</b>
4.1 Esempi di logica combinatoria .....	p. 16
4.1.1 Esempio 1: controllo di un'entrata .....	p. 16
4.1.2 Esempio 2: controllo di un'entrata vettoriale.....	p. 17
4.1.3 Esempio 3: TIMER sequenziale .....	p. 19
4.1.4 Esempio 4: Semaforo sequenziale .....	p. 20
<b>5. Trasferimento dal programma alla logica .....</b>	<b>p. 21</b>
5.1 Creazione di un progetto VHDL in "Synplify - Synplicity" .....	p. 22
5.2 L'ambiente di programmazione in "Synplify - Synplicity" .....	p. 24
5.3 Creazione di un progetto in "IspDS+ - Lattice" .....	p. 26
5.4 Scaricamento nella logica con "ispDS+ Download" .....	p. 31

# Introduzione

## 1.1 Cos'è il VHDL

Il VHDL è un linguaggio di descrizione hardware.

L'abbreviazione VHDL deriva da *VHSIC* (Very High Speed Integrated Circuit) e Hardware Description Language (linguaggio di descrizione hardware per circuiti a velocità d'integrazione molto alta).

## 1.2 Storia

All'inizio degli anni '80, il dipartimento della difesa americana (DOD) desiderava creare un linguaggio standard di descrizione e di documentazione dei sistemi hardware anziché utilizzare un linguaggio logico, per avere un'indipendenza con i loro fornitori.

È per questo che *DOD* creerà due linguaggi che si assomiglieranno fortemente: *ADA* (logico) e *VHDL* (hardware - fisico).

Lo standard VHDL prenderà piede nel 1987 e verrà normalizzato *dall'IEEE* (Institute of Electrical and Electronics Engineers).

Nel 1993 una nuova revisione dell'IEEE ha permesso di sfruttare a pieno le capacità del linguaggio VHDL, in particolare per:

- La sintesi automatica dei circuiti a partire dalla descrizione
- La verifica delle costruzioni temporanee (prototipi)
- La verifica formale dell'equivalenza dei circuiti

## 1.3 Perché utilizzare VHDL

Il VHDL è un linguaggio standard riconosciuto da tutti i venditori di strumenti *CAE* (*Computer Aided Electronics*). Questo gli permette di continuare ad essere usato e allo stesso tempo permette agli industriali di lanciarsi su uno strumento duraturo.

Tecnicamente è un linguaggio potente, moderno e generale che assicura un'eccellente lettura.

# I concetti del linguaggio VHDL

## 2.1 I segnali

Il *segnale* è specifico del linguaggio di programmazione hardware, è presente per modellare le informazioni che transitano tra i componenti fisici (porte) ed è connesso permanentemente per l'intermediario di una o più funzioni logiche. Si distingue dalla *variabile* per il fatto che in VHDL, il segnale è connesso istantaneamente ad altre variabili.

### 2.1.1 Breve definizione di variabile

La *variabile* è l'immagine di un oggetto al quale possiamo associare in ogni momento il valore che noi vogliamo.

*Esempio:*

```
variable A, B : bit;  
  
begin  
    A := B;  
end;
```

Il valore della variabile B è stato messo nella variabile A. Questa è un'operazione istantanea, dopo questo procedimento però non c'è più nessun legame tra le due variabili (A e B).

### 2.1.2 Breve definizione di segnale

Il *segnale*, nel linguaggio VHDL, è l'immagine di un'*entrata* (in) o di un'*uscita* (out) di un componente logico. Il collegamento tra uno o più segnali in un segnale ha un legame diretto.

Il collegamento di un segnale si fa tramite l'operatore "<=".

*Esempio:*

```
signal A, B, Y : Bit;  
  
begin  
    Y <= A or B;  
end;
```

L'esempio qui sopra è caratterizzato dalla porta "OR" logica.

Sottolineiamo il fatto che il collegamento di un segnale in un altro viene eseguito solamente alla conclusione del processo, mentre con le variabili è istantaneo.

## 2.1.3 I differenti tipi di segnali

Esistono molti tipi di segnali, i più utilizzati sono:

- Tipo bit
- Tipo std\_logic
- Tipo std\_logic\_vector

### 2.1.3.1 Tipo Bit

Il segnale del tipo *bit* è lo standard in VHDL per un segnale. Il bit ha solamente un valore che può essere uno dei due livelli logici:

- '0', stato logico forzato a 0
- '1', stato logico forzato a 1

#### Osservazioni:

Non bisogna confondere i segnali di tipo *bit* con i segnali del tipo *booleano*.

Il tipo bit è associato a un segnale a meno che il tipo booleano non ha rappresentazioni fisiche, esso è utilizzato per i risultati delle operazioni razionali:

- = (uguale)
- /= (diverso)
- > (maggiore)
- < (minore)
- >= (maggiore o uguale)
- <= (minore o uguale).

Per capire meglio le differenze tra *tipo bit* e *tipo booleano* guardare l'esempio qui sotto:

Il codice qui sotto è sbagliato nel linguaggio VHDL:

```
signal A, B, Y : bit;

begin
    y <= (A >= B);
end;
```

Evidenziamo il fatto che il risultato dell'operazione  $A \geq B$  è di tipo booleano, dunque non può essere assegnato al segnale *y* che è di tipo bit. Se vogliamo rendere questa operazione corretta in VHDL, dobbiamo scriverla così:

```
signal A, B, Y : bit;

begin
    y <= '1' when (A >= B) else '0';
end;
```

Sottolineiamo questa volta che il valore assegnato al segnale *y* è di tipo *bit*.

### 2.1.3.2 Tipo *std\_logic*

Il tipo *std\_logic* è molto utilizzato perché ha la possibilità di prendere vari stati logici:

- '0', stato logico forzato a 0
- '1', stato logico forzato a 1
- 'X', stato logico indefinito
- 'L', stato logico non forzato a 0
- 'H', stato logico non forzato a 1
- 'W', stato logico non forzato a indefinito
- 'Z', stato di alta impedenza
- 'U', segnale non inizializzato
- '-', stato non importante

Per questi differenti stati logici, il tipo *std\_logic* è utile per descrivere delle porte logiche a tre stati o delle uscite per circuiti a collettore aperto.

### 2.1.3.3 Il vettore

VHDL offre la possibilità di descrivere un insieme di segnali sotto forma di *vettore*. La definizione di un *bus di 3 bit* si può fare nel seguente modo:

```
signal Bus_3_bits: std_logic_vector (2 downto 0);
```

Il tipo *vettore* può essere sia del tipo *bit*, che del tipo *std\_logic*, basta specificare il tipo nella definizione, che è *std\_logic\_vector*, o *bit\_vector*.

Il numero di segnali che possono comporre il vettore è specificato nei parametri di quest'ultimo (X & Y).

```
signal parametro_vettore: std_logic_vector (X downto Y);
```

### 2.1.3.4 Assegnazione del valore in un segnale o in un vettore

L'assegnazione di uno stato logico a un segnale si effettua tramite l'uso dell'apostrofo ('stato\_logico'), mentre l'assegnazione di uno stato logico di un vettore si effettua tramite l'uso delle virgolette ("stato\_logico").

```
signal_1_bit      <= '0';  
vector_3_bits    <= "001";
```

## 2.2 I moduli

Il modulo in VHDL è un oggetto con il quale descriviamo un sistema numerico andando da una semplice porta logica a un sistema complesso.

Tutte le librerie in VHDL contengono al minimo un modulo che è composto da una rappresentazione esterna di esso (entità) e da una interna (architettura).

### 2.2.1 L'entità

L'entità è una vista esterna del modulo, esse definisce tutte le entrate e le uscite.

#### 2.2.1.1 Sintassi generica dell'entità di un modulo

```
entity nome_del_modulo is  
  
    port (   nome_entrata_1: in      tipo_del_segnaled;  
           nome_entrata_2: in      tipo_del_segnaled;  
           nome_uscita_1:  in      tipo_del_segnaled);  
  
end nome_del_modulo;
```

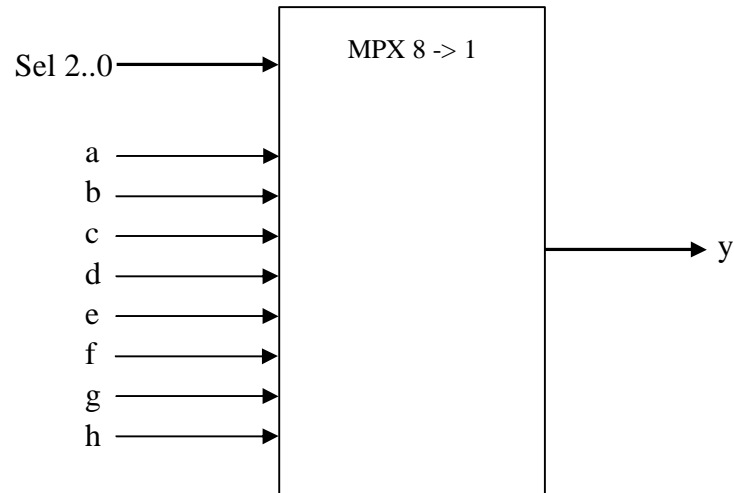
#### Osservazioni:

Le entrate e le uscite di un modulo sono sempre rappresentate tramite delle porte, ne esistono tre diversi tipi:

- *in*            porta d'entrata
- *out*          porta d'uscita
- *inout*        porta bidirezionale (usata per i bus)

### 2.2.1.2 Esempi di un'entità

Per essere più concreti, rappresentiamo ora l'entità di un *multiplexer 8 -> 1* con 3 bit di selezione (Sel):



#### Vista esterna del multiplexer nel linguaggio VHDL:

```
entity example1 is
    port (sel
           A, B, C, D, E, F, G, H : in std_logic;
           Y                       : out std_logic);
end example1;
```

## 2.2.2 L'architettura

L'architettura descrive i componenti del modulo. Un'architettura può essere descritta in tre maniere differenti:

- Flusso di dati: descrizione del modulo tramite un'equazione booleana.
- Comportamentale: descrizione del modulo confrontando il comportamento dei segnali uno in rapporto con l'altro.
- Strutturale: descrizione del modulo confrontando unicamente le interconnessioni tra i diversi moduli esistenti.

### 2.2.2.1 La sintassi generica di un'architettura

```
architecture tipo_d_architettura of nome_del_modulo is
```

```
begin
```

```
    istruzioni_parallele;
```

```
    process (lista_della_sensibilità)
```

```
        istruzione_sequenziale;
```

```
    end process;
```

```
end tipo_d_architettura;
```

I sistemi elettronici funzionano in maniera parallela. È per questo che esistono queste istruzioni nel linguaggio VHDL.

Tuttavia non è sempre possibile lavorare in modo parallelo. In questo caso dovremmo lavorare in modo sequenziale, dove un'operazione viene eseguita quando è terminata la precedente.

Le operazioni sequenziali sono raggruppate all'interno del processo (process). Quest'ultimo, però, lavora in modo parallelo ad altri processi.

# Le istruzioni elementari

### 3.1 Le istruzioni sequenziali

Tutte le istruzioni sequenziali *devono* essere incluse in un processo.

### 3.2 L'istruzione scelta (case)

La sintassi generica di un'istruzione di scelta è la seguente:

```
case expression is
    when valore_1           => uscita_1;
    when valore_2           => uscita_2;
    when valore_3 to valore_4 => uscita_3;
    when valore_6 downto valore_5 => uscita_4;
    .
    .
    when others             => uscita_default;
end case;
```

Riprendiamo l'esempio della descrizione di un multiplexer 8 -> 1 (con 3 linee di selezione e le entrate da 4 a 7 messe a 0).

```
architecture mpx of esempio 4 is

begin

    process (sel, A, B, C, D) begin

        case sel is
            when "000"   => Y <=A;
            when "001"   => Y <=B;
            when "010"   => Y <=C;
            when "011"   => Y <=D;
            when others  => Y <='0';
        end case;

    end process;

end mpx;
```

### 3.3 L'istruzione condizionale (if)

La sintassi generica di un'istruzione condizionale è la seguente:

```
if condizione_booleana_1 then
    istruzione_1;
elsif condizione_booleana_2 then
    istruzione_2;

else
    istruzione_default;
end if;
```

Rappresentiamo l'esempio di descrizione di un multiplexer 84 -> 1 (con 3 linee di selezione e le entrate da 4 a 7 messe a 0).

```
architecture mpx of esempio_5 is
begin

    process (sel, A, B, C, D) begin

        Y <= '0'                -- inizializzazione obbligata al fine di non aggiungere
                                -- dei latch durante la sintesi.

        if (sel = "000") then
            Y <= A;

        elsif (sel = "001") then
            Y <= B;

        elsif (sel = "010") then
            Y <= C;

        elsif (sel = "011") then
            Y <= D;

        end if;

    end process;

end mpx;
```

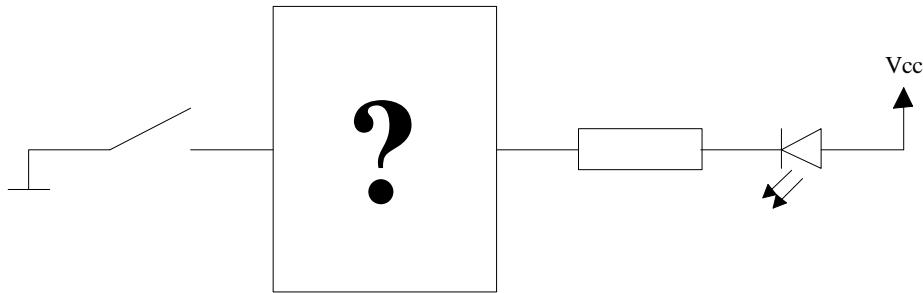
# Esempi pratici

## 4.1 Esempi di logica combinatoria

Per concludere questo breve corso di programmazione, proponiamo ora degli esempi pratici, così da consolidare un po' le basi...

### 4.1.1 Esempio 1: controllo di un'entrata

Il primo esempio consiste nel monitorizzare lo stato di un'entrata. Lo stato di un interruttore viene visualizzato su un LED.



```
-- *****
-- * INT - LED.vhd      Visualizza lo stato di un interuttore su un LED.          *
-- *                                                            *
-- *                    Introduzione al VHDL...                               *
-- *                    By Claudio P. & Simone B.                             *
-- *****

library ieee;                                     -- carica le librerie std...
use          ieee.std_logic_1164.all;
use          ieee.std_logic_unsigned.all;

entity esempio_1 is                               -- definizione entita'
    port(
        input:  in      std_logic;                -- definizione entrate
        output: out     std_logic                 -- definizione uscite
    );
end esempio_1;                                   -- chiusura definizione entita'

architecture behave of esempio_1 is               -- definizione architettura esempio_1
    begin                                         -- da qui in poi, seguono tutte le...
                                                -- istruzioni di programma.

        output <= input;                          -- istruzione principale...
                                                -- quello che c'è in input (entrata)...
                                                -- mettilo in output (uscita)

    end behave;                                   -- chiusura architettura, fine programma.
```



```
        when "01111111" => output <= "0000000";      -- scrivi 8 sul display 7seg
        when others => output <= "0010000";         -- scrivi 9 sul display 7 seg, per difetto

    end case;                                         -- fine istruzione seq. scelta.

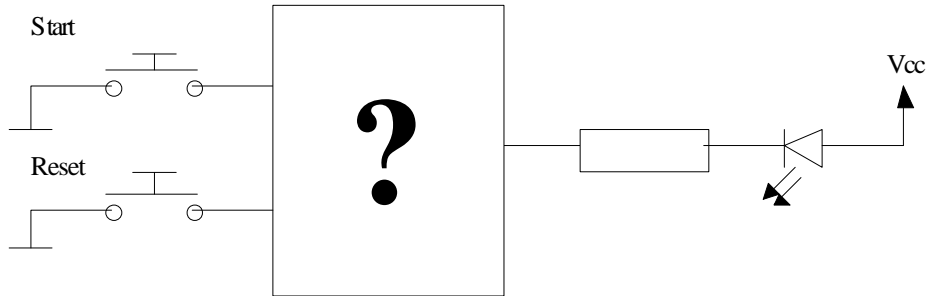
end process;                                         -- chiusura processo.

end behave;                                          -- chiusura architettura, fine programma.
```

### 4.1.3 Esempio 3: Timer sequenziale

Il terzo esempio consiste nel monitorizzare un'entrata e visualizzarne il suo stato tramite un LED. Quest'ultimo è temporizzato. (circa 5 sec.)

È anche presente un'entrata di reset. (dominante)



```
-- *****
-- * PULS - Timer LED          Dopo aver premuto un pulsante, si accende un LED per 5 s.      *
-- *                                                                    *
-- *                          Introduzione al VHDL...                          *
-- *                          By Claudio P. & Simone B.                          *
-- *****
```

```
library ieee; -- carica le librerie std...
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity esempio_3 is -- definizione entita'
    port(
        clkin: in std_logic; -- clock input di 1 MHz.
        reset: in std_logic;
        start: in std_logic; -- definizione entrata pulsante
        led_o: out std_logic -- definizione uscita LED
    );
end esempio_3; -- chiusura definizione entita'
```

```
architecture behave of esempio_3 is -- definizione architettura esempio_3
    -- definizione dei segnali
    signal clkout: std_logic;
    signal led: std_logic;

    signal counter: std_logic_vector (2 downto 0); -- definizione di un segnale vettoriale
    signal tempo: std_logic_vector (19 downto 0);

begin -- da qui in poi, seguono tutte le...
    -- istruzioni di programma.
    led_o <= led;

    process (clkin) begin -- se clkin cambia, allora...
        if (rising_edge(clkin)) then -- se clkin e' sul fronte di salita, allora...
            if (tempo = "00000000000000000000") then -- se tempo e' a 0, allora...
                tempo <= "11110100001001000000"; -- ricaricalo a 1'000'000 e metti il
                clkout <= '1'; -- segnale "clkout" a 1.
            else -- se no...
                tempo <= tempo - '1'; -- decrementalo tempo di 1 e...
                clkout <= '0'; -- lascia clkout a 0.
            end if;
        end if;
    end process;
end behave;
```

```
        end if;

    end if;
end process;                                -- chiusura processo

process (reset, start, clkout, counter) begin -- inizio processo

    if (reset = '0') then
        counter <= "000";
        led     <= '1';
    else

        if (start = '0') then
            led     <= '0';
            counter <= "000";

        else

            if (rising_edge(clkout)) then
                counter <= counter + '1';
            end if;

            if (counter > "101") then
                led     <= '1';
                counter <= "000";
            end if;
        end if;
    end if;

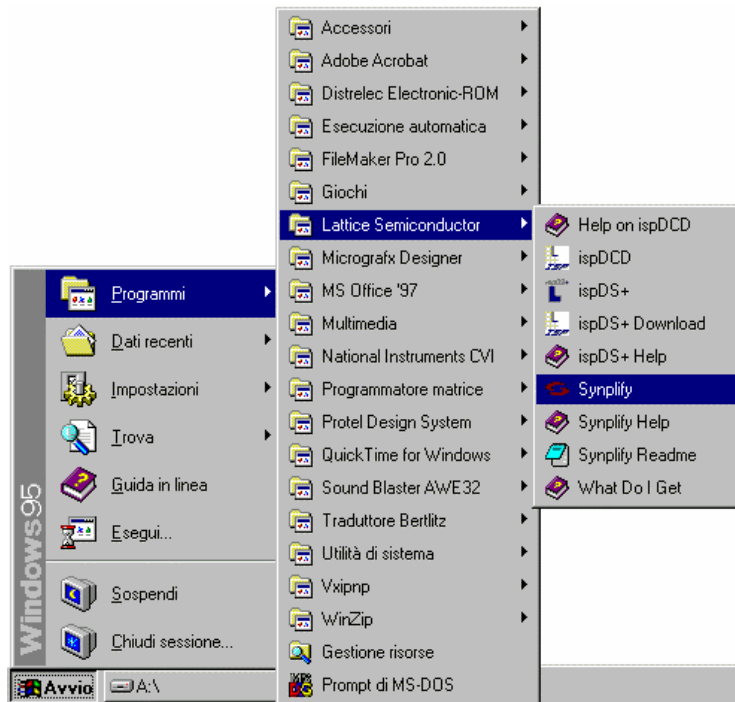
end process;

end behave;                                -- chiusura architettura, fine programma.
```

# **Trasferimento dal programma alla logica**

## 5.1 Creazione di un progetto VHDL in "Synplify - Synplicity"

Come prima cosa bisogna aprire il programma che permette di programmare in VHDL che nel nostro caso si chiama *Synplify*.

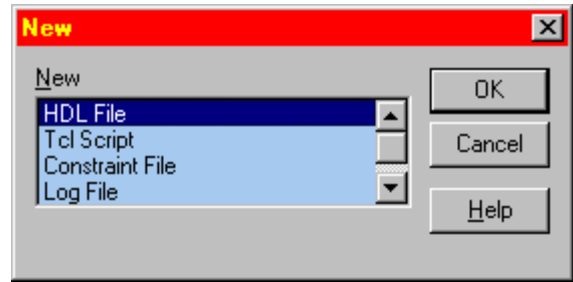
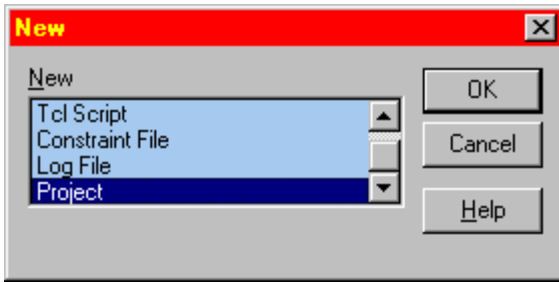


Una volta che il programma è stato aperto bisogna creare un nuovo File o un nuovo progetto (menu *File* opzione *new*).

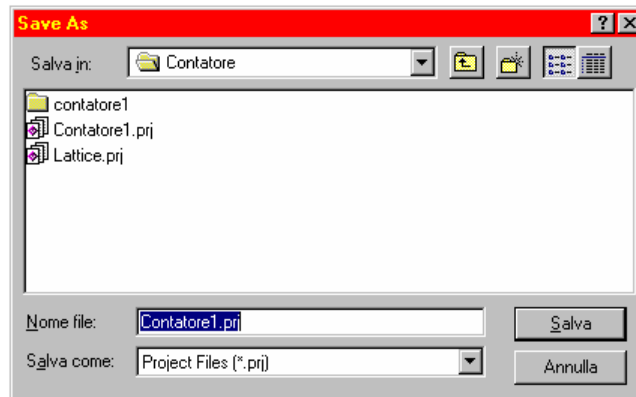


Per creare un nuovo progetto bisogna selezionare *Project*, mentre per creare un nuovo file sorgente si selezionerà *HDL File*.

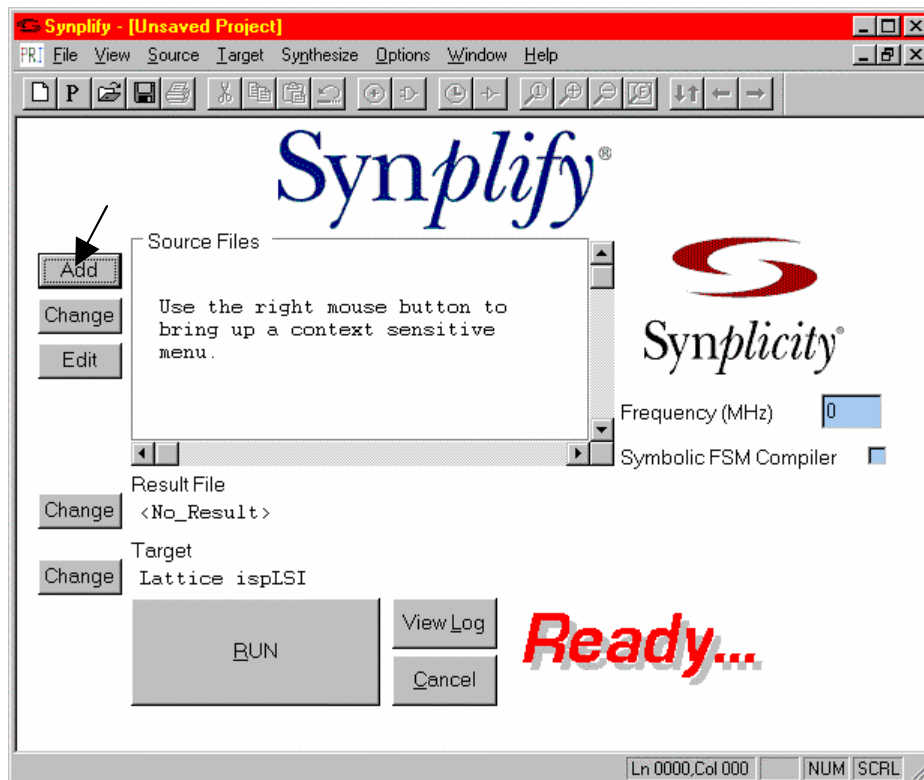
Solitamente prima si crea un nuovo progetto e poi un nuovo file.



Dopo questa operazione si aprirà una finestra dove si potrà scegliere il percorso ed il nome del file.



Una volta creato il progetto e il file di programma VHDL bisogna aggiungere al progetto il file di programma premendo il tasto Add.

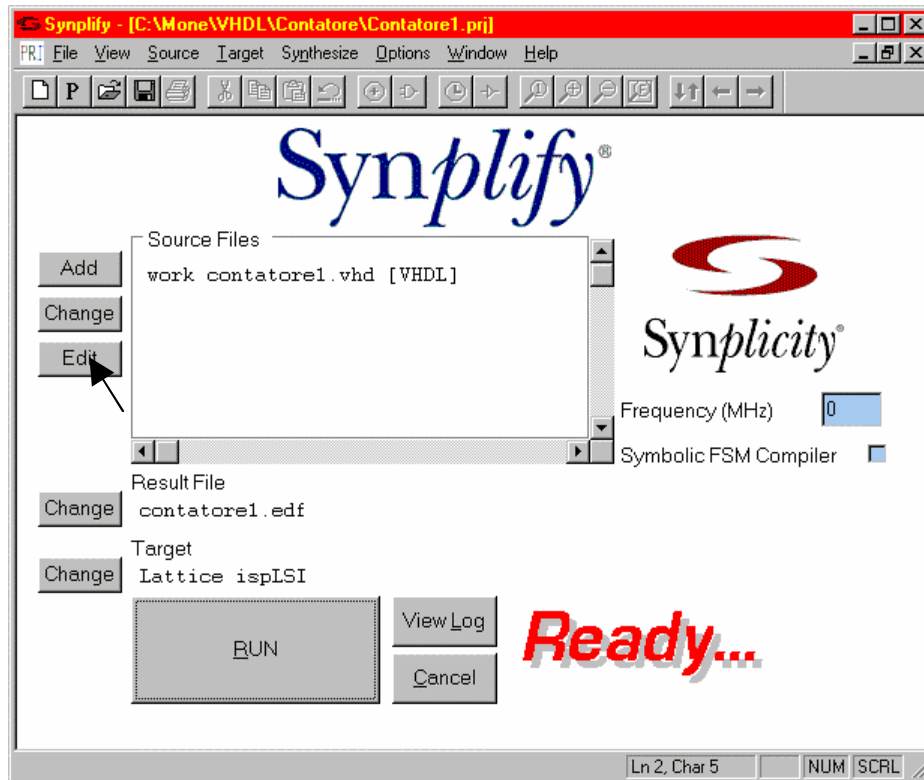


## 5.2 L'ambiente di programmazione "Synplify - Synplicity"

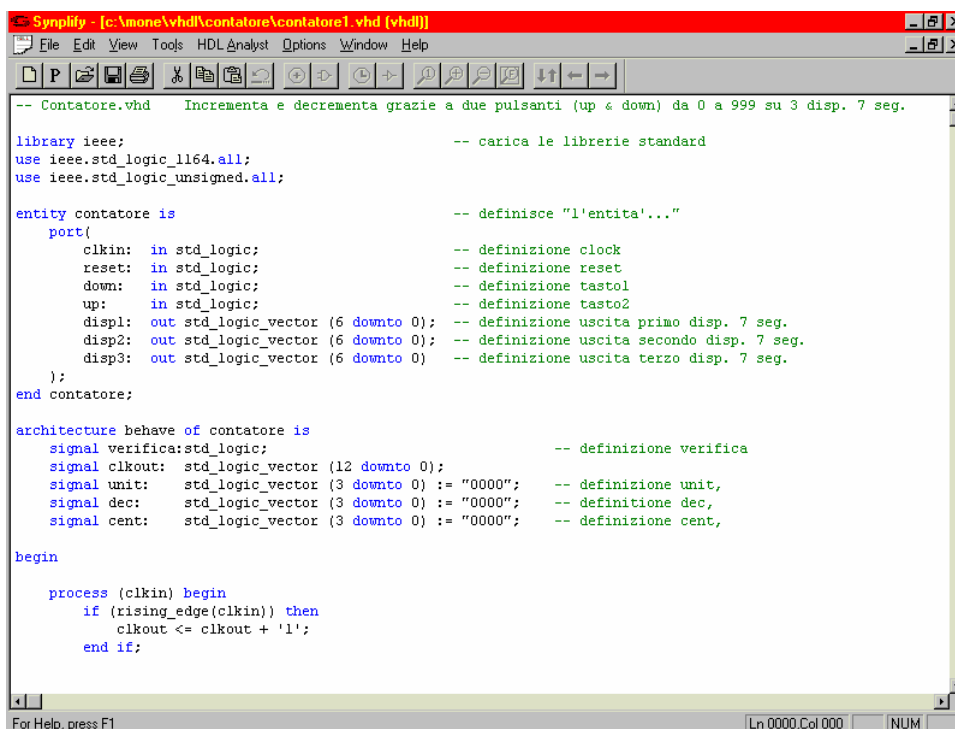
Quando il file è stato aggiunto al progetto si presenta la seguente situazione.

Si seleziona il file VHDL che si vuole editare (vedi casella *Source files*).

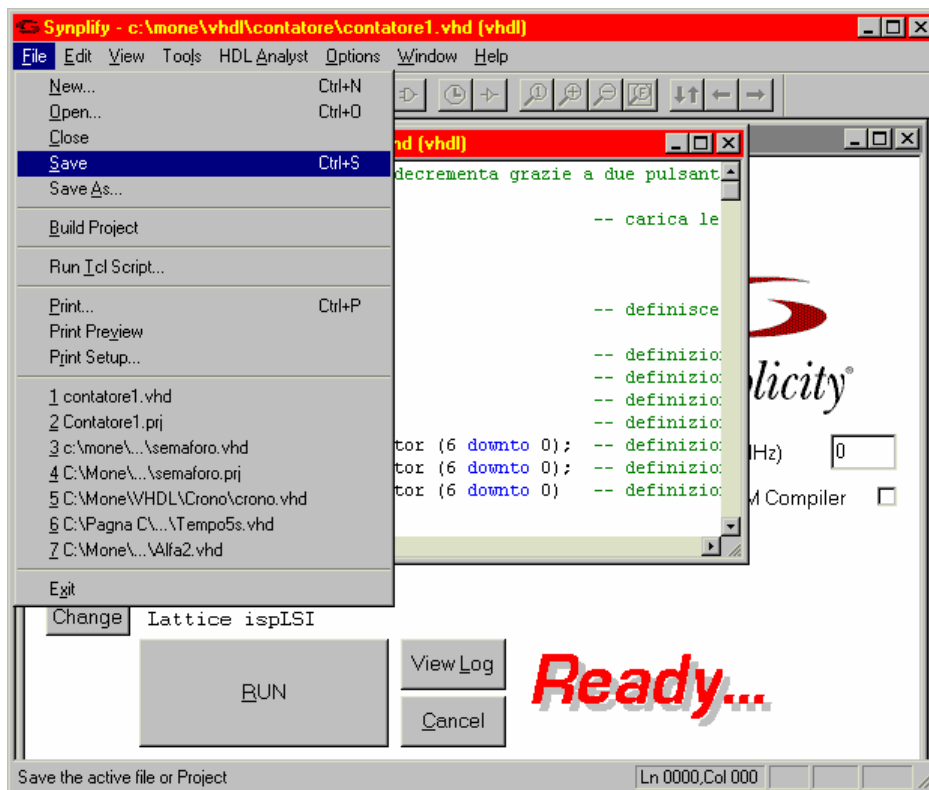
Nel nostro caso, il file da modificare si chiama "contatore1.vhd". Per editarlo si preme il pulsante *Edit*.



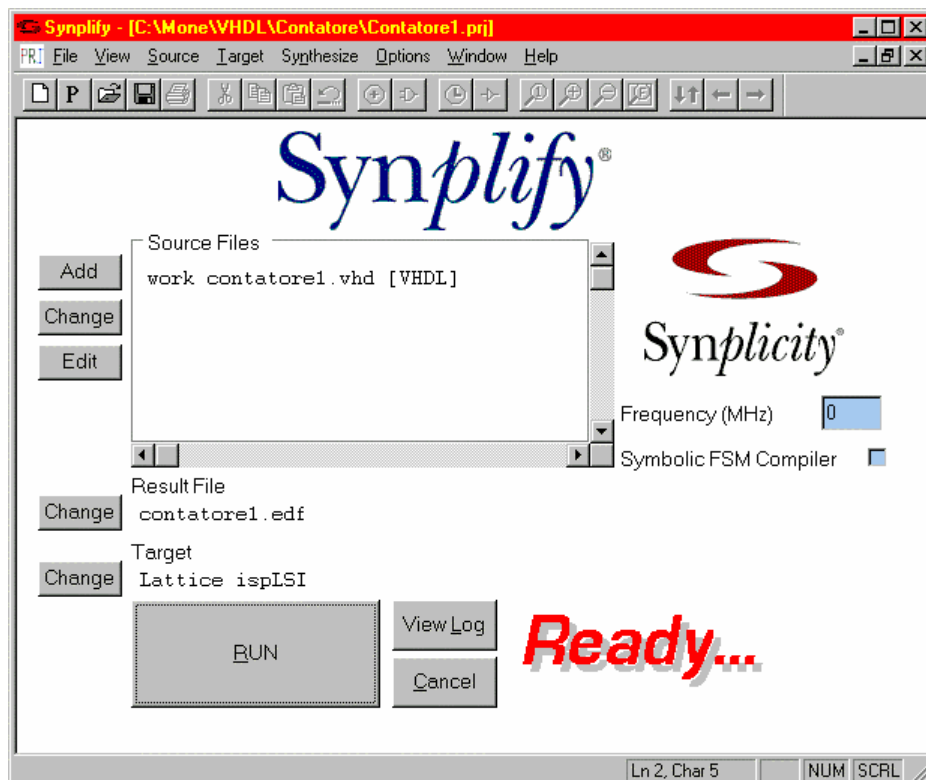
Una volta eseguita questa operazione si aprirà una finestra che sarà poi il nostro ambiente di lavoro (programmazione VHDL).



Per salvare il programma si seleziona il menu *File* e si seleziona *Save*.

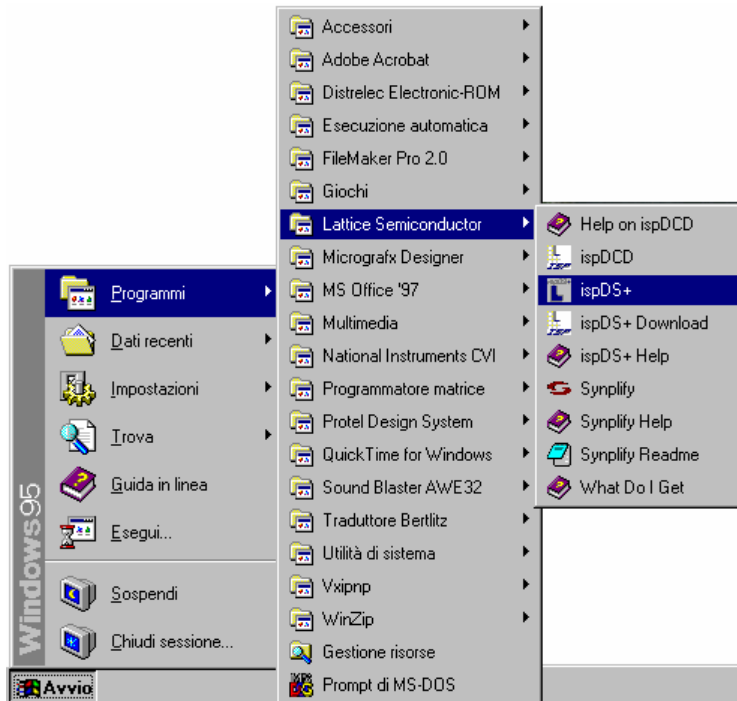


Una volta scritto il programma e salvato, bisogna pre-compilarlo. Per farlo, basta preme il pulsante *RUN*.

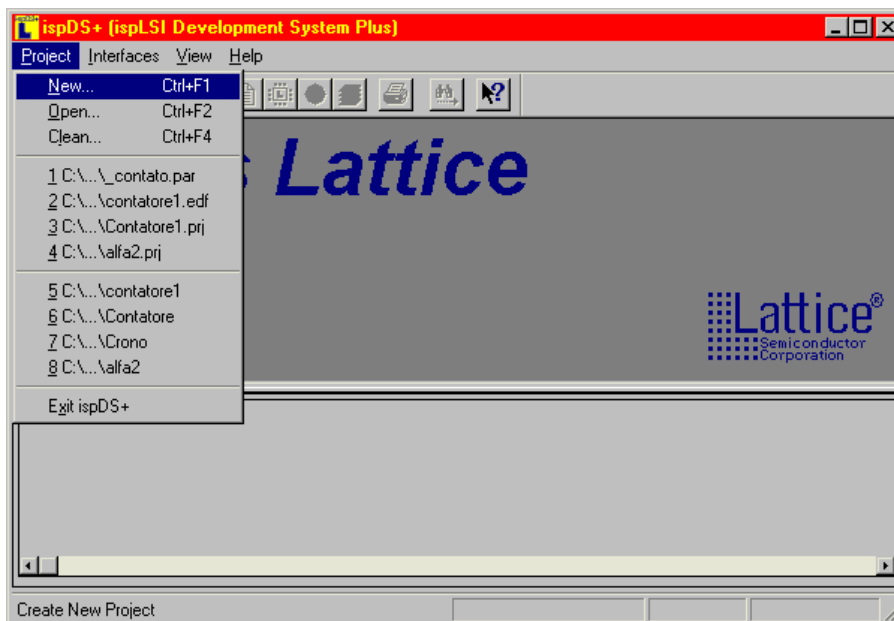


### 5.3 Creazione di un progetto in "ispDS+ - Lattice"

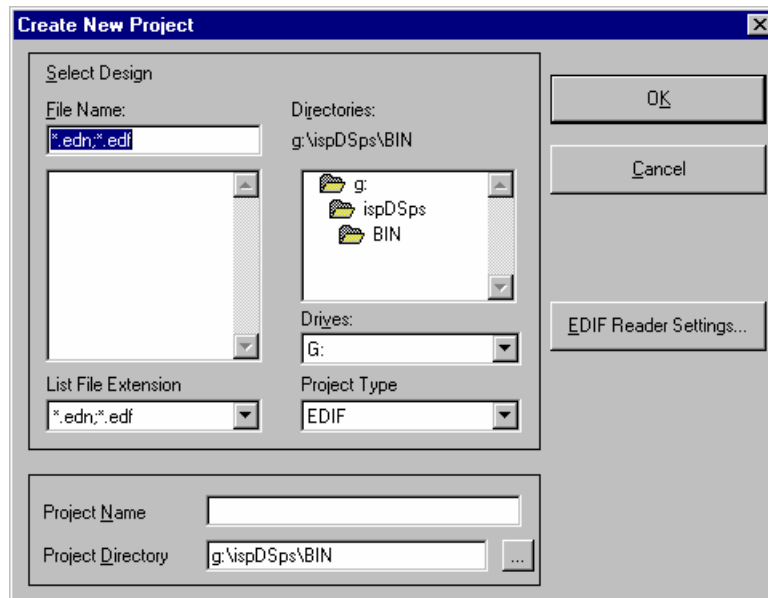
Una volta scritto il programma, si procede con la compilazione di quest'ultimo. Infatti, la logica (PLD) non comprende il codice VHDL, ma bensì quello Jedec.  
 Per fare questo bisogna aprire un programma in grado di compilare i file VHDL, che nel nostro caso si chiamerà *ispDS+*.



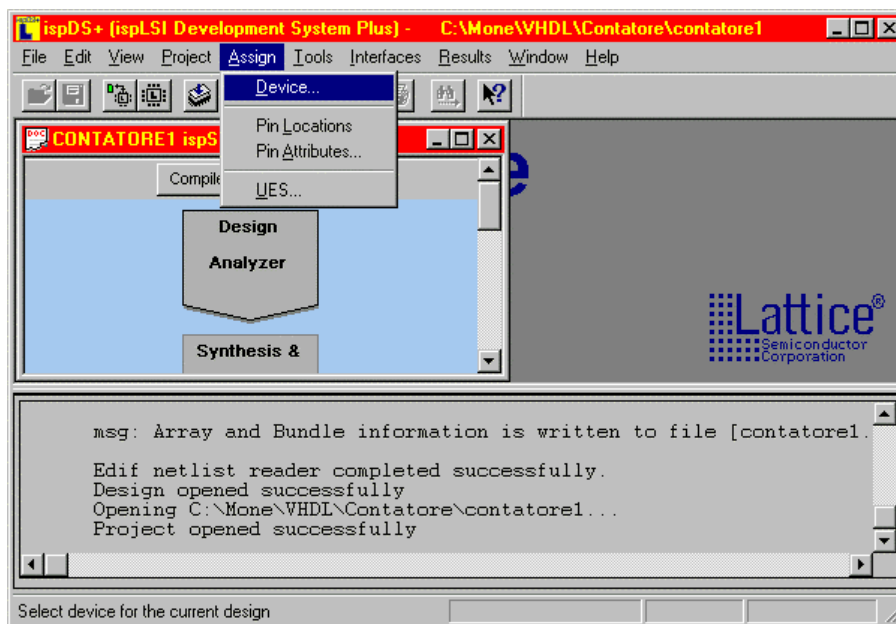
Una volta aperto è necessario creare un nuovo progetto, che sarà quello definitivo prima dello scaricamento nella logica.  
 Per fare ciò bisognerà andare sul menu *Project* e selezionare l'opzione *New...*



Subito dopo verrà visualizzata una finestra dove verrà chiesto il percorso del file compilato (estensione *.edf*).

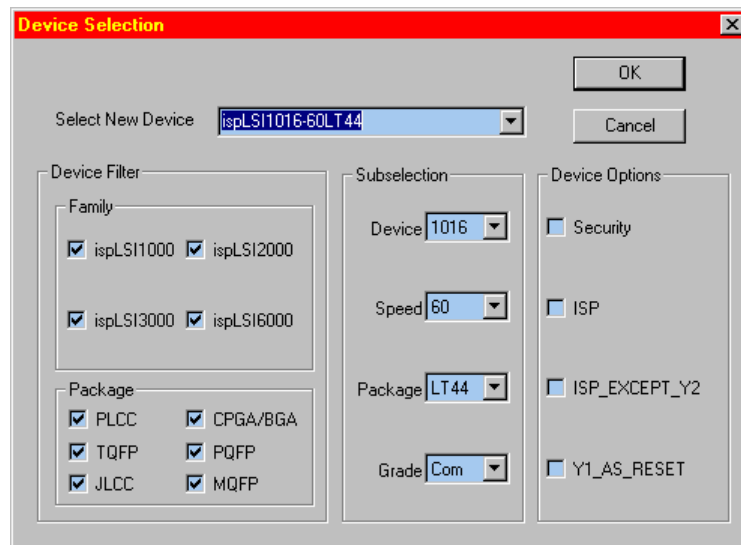


Fatto questo, si procede con la definizione del tipo di PLD utilizzata. Per fare selezionare il menu *Assign* e scegliere *Device*.



A questo punto si presenta una finestra dove si può scegliere il tipo di logica (vedi lista *Select New Device*).

Fatto questo, dare dare conferma premendo il pulsante "OK".

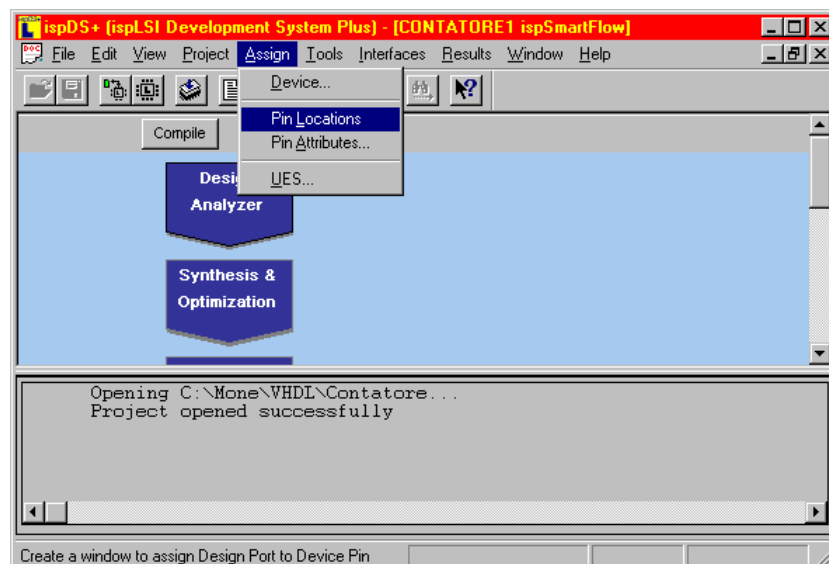


La casella *Security* serve per bloccare la lettura (magari in un secondo tempo) del programma contenuto nella logica.

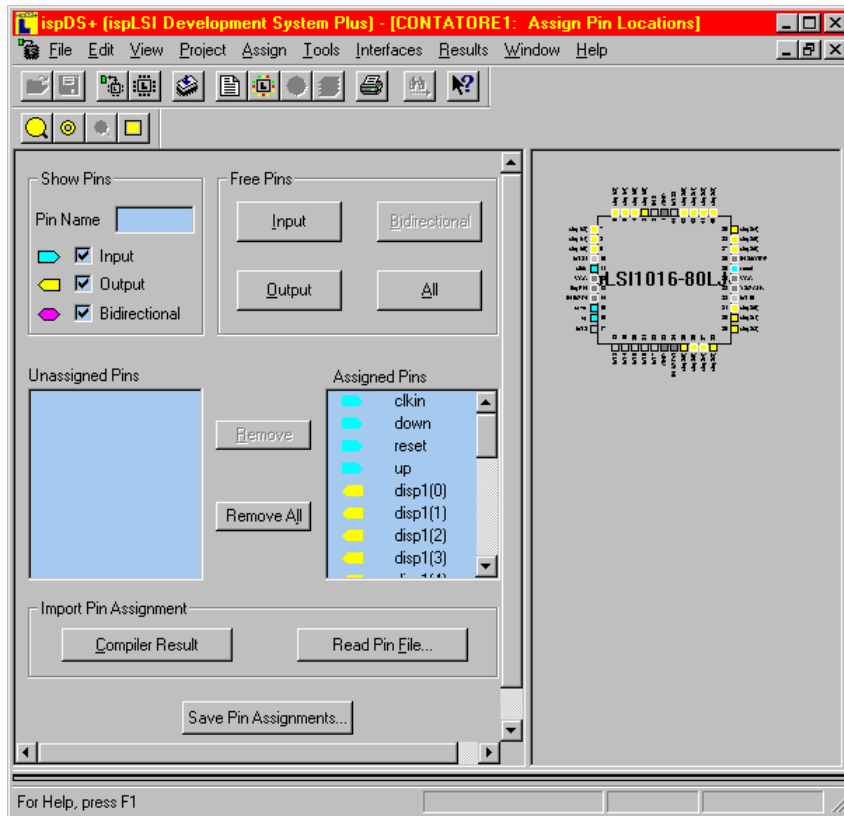
La casella *ISP* serve per riservare i pin della logica alla funzione *ISP* (*In System Programming*).

La casella *Y1\_AS\_RESET* serve per riservare il pin *Y1* come reset hardware.

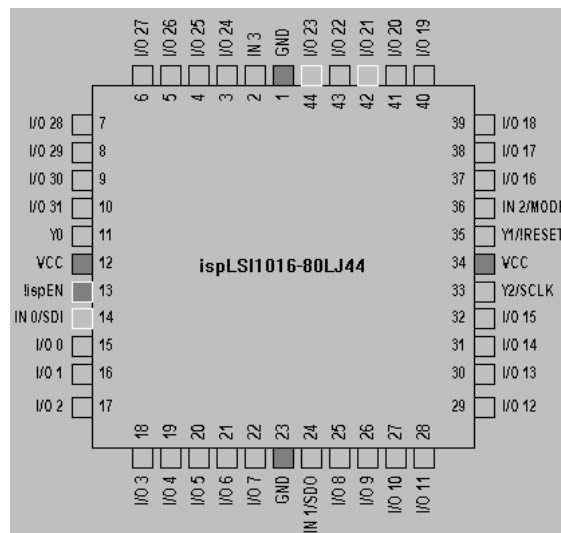
Come secondo passo bisogna definire le funzioni dei vari piedini della logica. Per fare questo bisogna andare sul menu "Assign" e selezionare "Pin Locations".



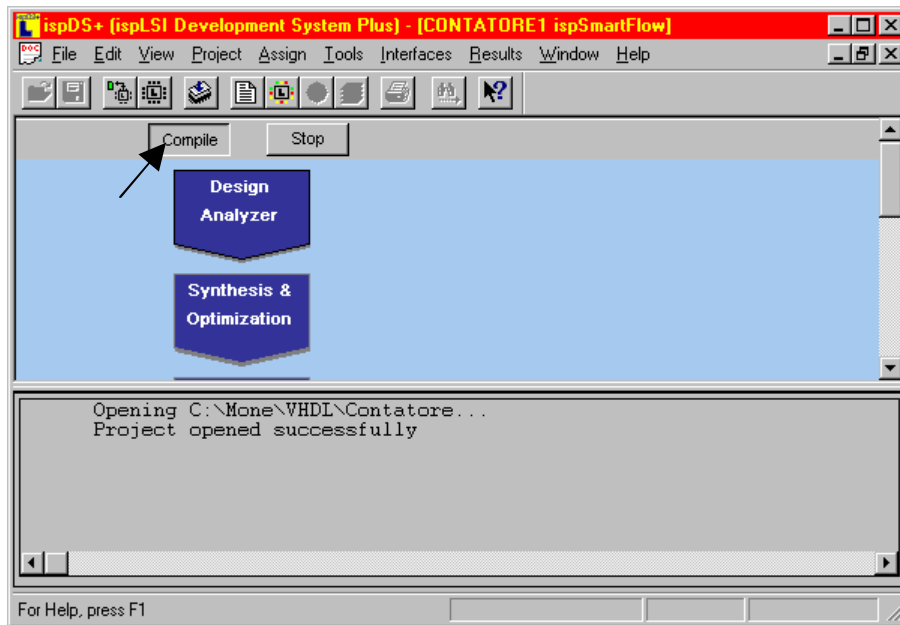
A questo punto si presenta una finestra simile dove si potrà definire ogni input e output utilizzati nella programmazione.



Esempio di configurazione dei pin di una PLD



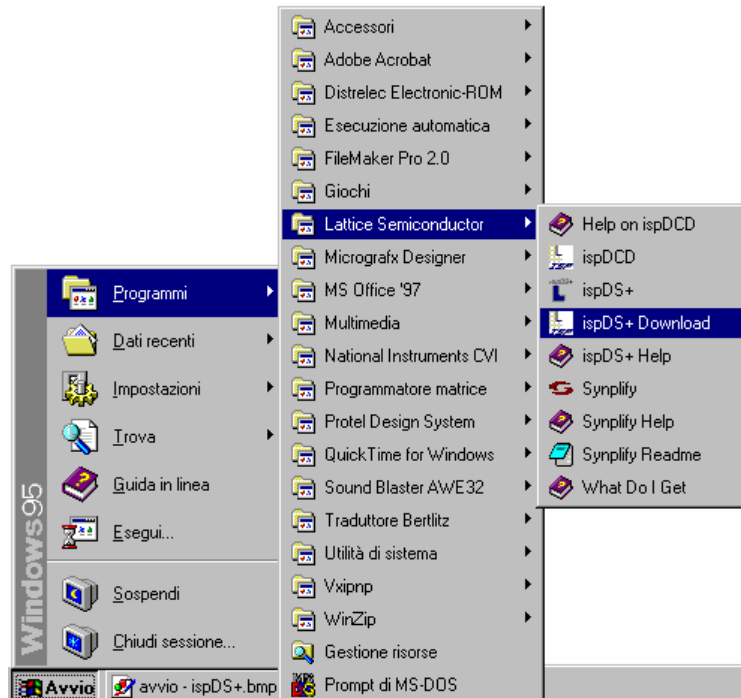
Arrivati a questo punto resta solo un'operazione da svolgere, ossia compilare il progetto. Per farlo basta premere il tasto *Compile*.



## 5.4 Scaricamento nella logica con "ispDS+ Download"

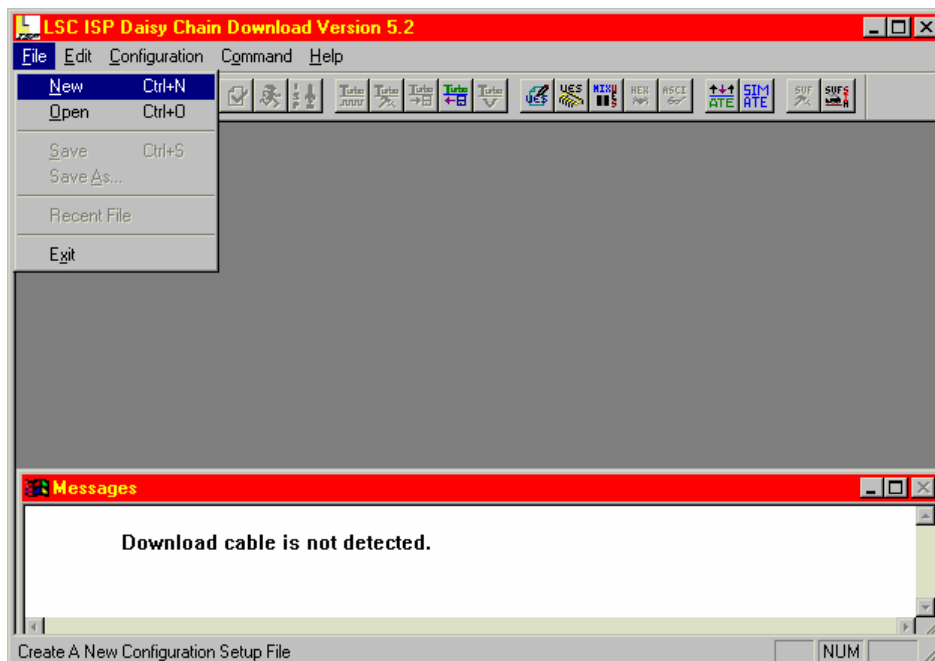
Una volta compilato il programma utente, bisogna aprire una utilità in grado di trasferire il tutto nella logica.

Nel nostro caso il programma in grado di fare questo si chiama *ispDS+ Download*.



La prima cosa da fare dopo aver avviato il programma è la creazione di un nuovo progetto.

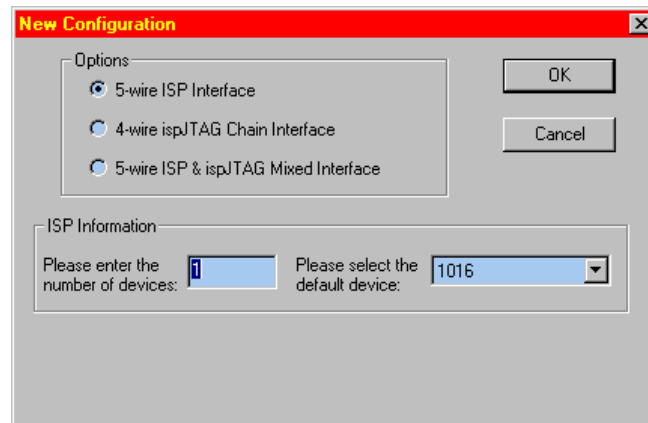
Si procede nel seguente modo:



Fatto questo, bisogna definire l'hardware utilizzato.

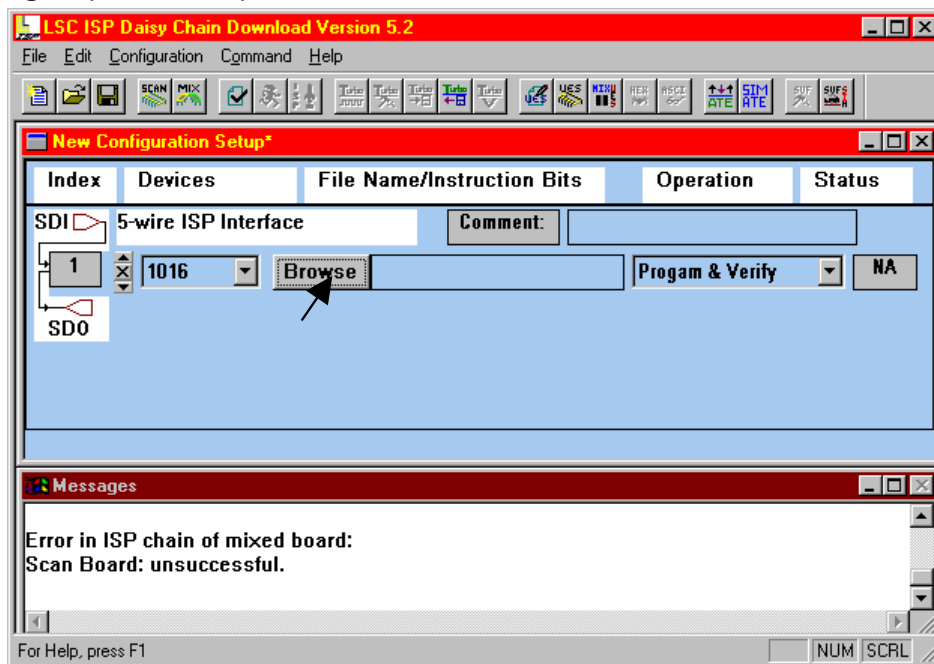
La nostra interfaccia è di tipo *5-wire ISP Interface*.

Nella casella *Please select the default device* si può selezionare il modello della PLD.



Dopo questa operazione, bisogna caricare il file da trasferire (Jedec).

Per farlo, bisogna premere il pulsante *Browse*.



Si aprirà una finestra dove si può selezionare il percorso del file.



Ora basta premere il pulsante *Run operation* ed il programma verrà trasferito nella logica.

