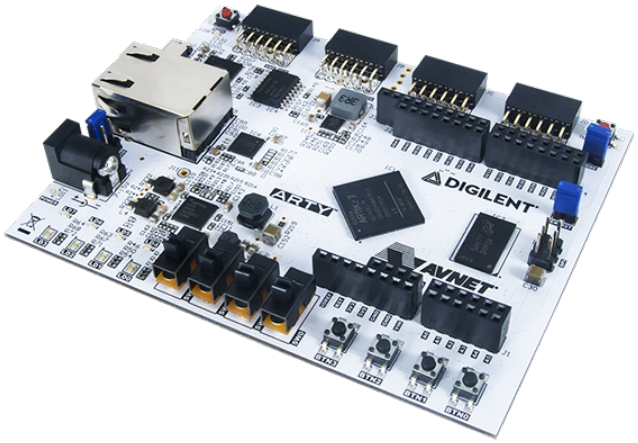
Flusso di Progettazione su FPGA

Introduzione



Lo scopo di questa esercitazione di laboratorio è introdurre alla progettazione di logica digitale e all'implementazione tramite dispositivi Field Programmable Gate Array (FPGA). Utilizzeremo il linguaggio VHDL per descrivere l'hardware da realizzare e una scheda Digilent Arty 35 basata su un chip XC7A35TICSG324-1L prodotto da AMD (ex Xilinx).
Il software di riferimento per lo sviluppo di logica in dispositivi AMD è Vivado. Esistono strumenti equivalenti per FPGA Intel (ex Altera) e per altri produttori.

> ***Nota sulle versioni di Vivado**: Questa guida si basa sulla versione 2019.1 di Vivado, il flusso di lavoro presentato è praticamente invariato anche sulle versioni più moderne. Un progetto Vivado è vincolato ad una versione specifica, se viene aperto con un'altra versione è necessaria una procedura di up-(down-)grade la cui riuscita non è garantita.*

Eseguire Vivado

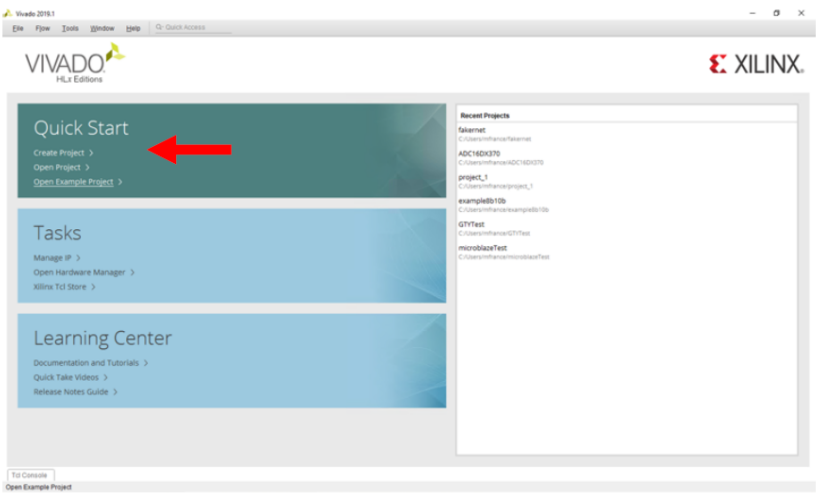
Una volta installato in Windows, l'applicazione può essere lanciata dal Desktop o dal menù delle applicazioni.

> ***Vivado su Linux**: Nel caso di installazione su Linux, occorre aggiornare le variabili di ambiente prima di poter lanciare l'eseguibile:*

```
$ source /opt/Xilinx/Vivado/2019.1/settings64.sh
$ vivado
```

Creazione di un nuovo progetto

La schermata di benvenuto di Vivado presenta molte opzioni, tra cui:

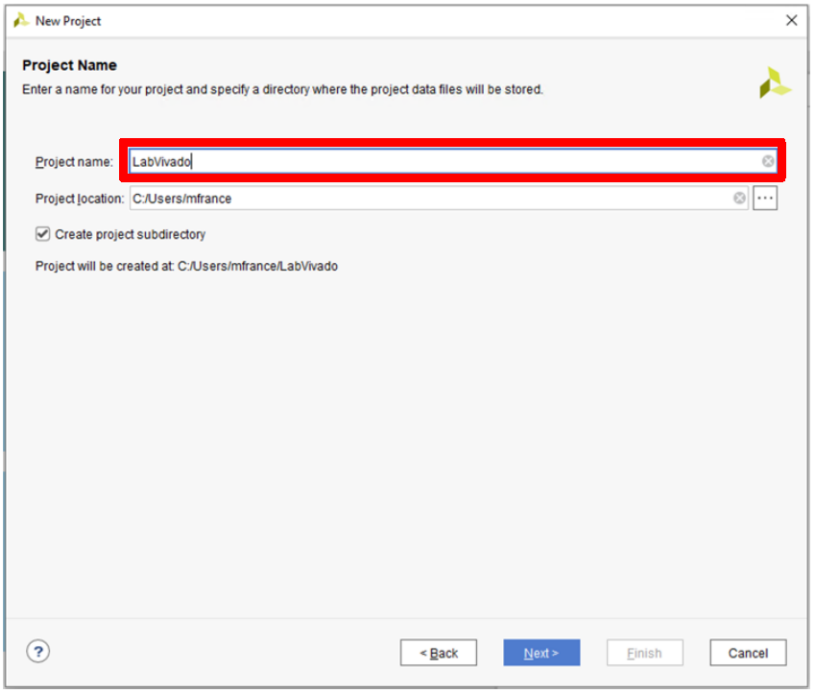


- **Create Project**: apre una procedura guidata per la creazione di un nuovo progetto;
- **Open Project**: permette di aprire un progetto esistente (da un file XPR) creato in precedenza o ottenuto da altre fonti;
- **Open Hardware Manager**: può essere utilizzato per programmare direttamente una scheda utilizzando un apposito programmatore.

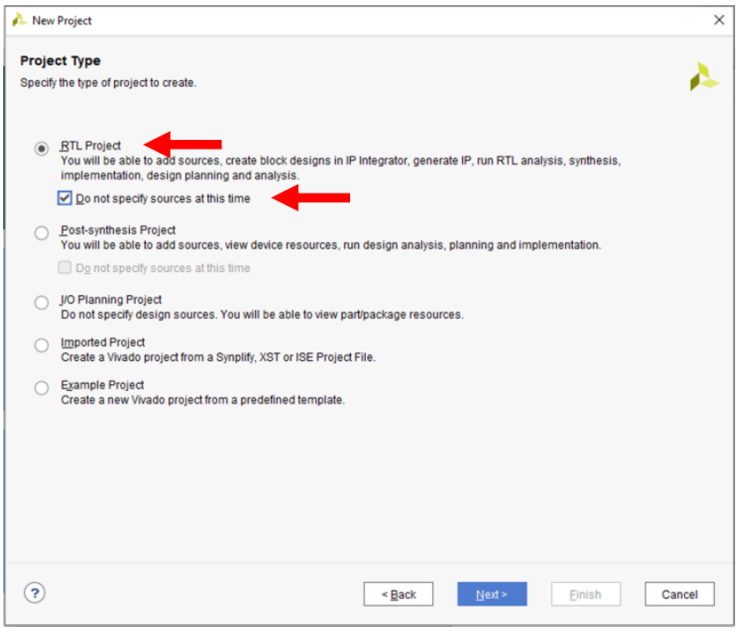
Procediamo con il pulsante **Create Project**.

Il primo passo è l'assegnazione di un **nome al progetto**. Vivado utilizzerà questo nome per generare l'insieme delle cartelle che descrivono il progetto.

***Importante**: Assicuratevi che NON ci siano spazi nel nome o nel percorso del progetto. Questo potrebbe causare problemi nelle fasi successive*

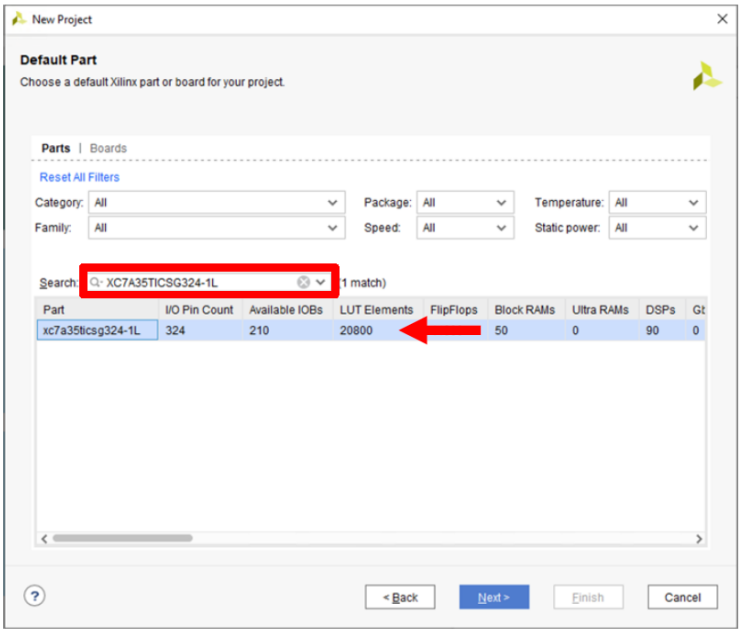


Al momento di selezionare il tipo di progetto, scegliete **RTL Project** (RTL = Register Transfer Logic) e spuntate la casella *Do not specify sources at this time* . Ciò evita le schermate che permettono di inserire file già esistenti.



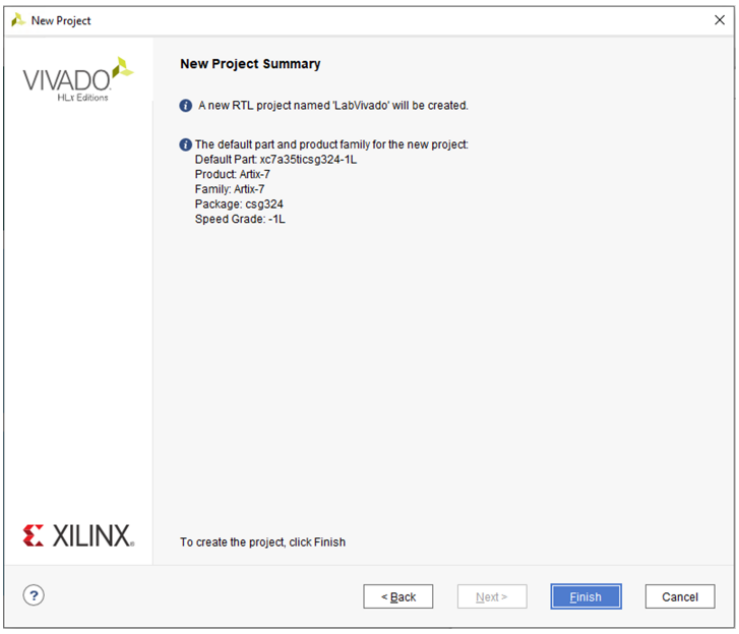
Vivado deve conoscere il componente FPGA che desideriamo utilizzare. I file compilati prodotti al termine del flusso di progetto saranno compatibili solo con il componente scelto. L'impostazione può essere cambiata nei settings del progetto.
Se la scheda di sviluppo è commerciale, probabilmente apparirà nella tab **Boards**. Così facendo si importeranno anche alcune definizioni aggiuntive del produttore, relative alla FPGA contenuta sulla scheda.

Se la vostra scheda non appare qui seguite le istruzioni sul sito del produttore.

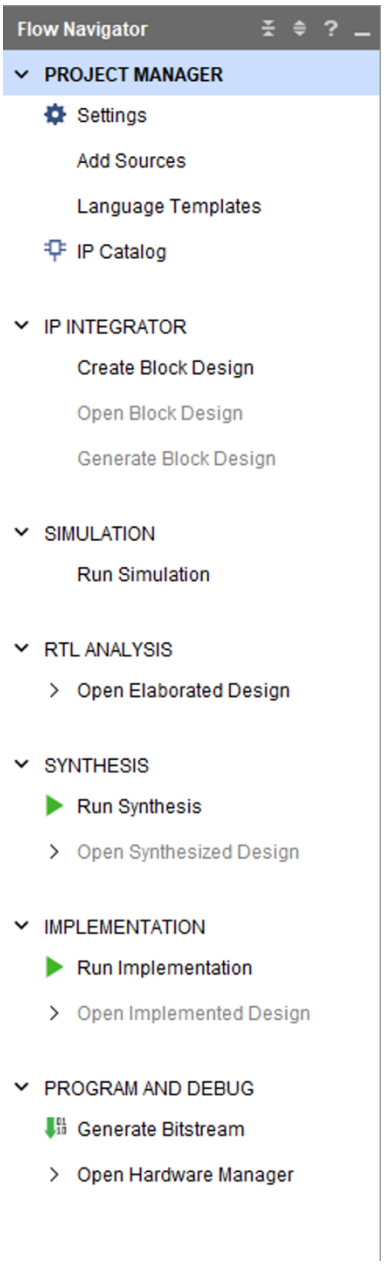


Nel caso più generico possiamo semplicemente scegliere il modello Artix-7 XC7A35TICSG324-1L nella tab **Parts.**

L’ultima schermata riassume le impostazioni del nuovo progetto. Premendo **Finish** l’ambiente di sviluppo verrà inizializzato con il progetto appena creato.



Il pannello **Flow Navigator**

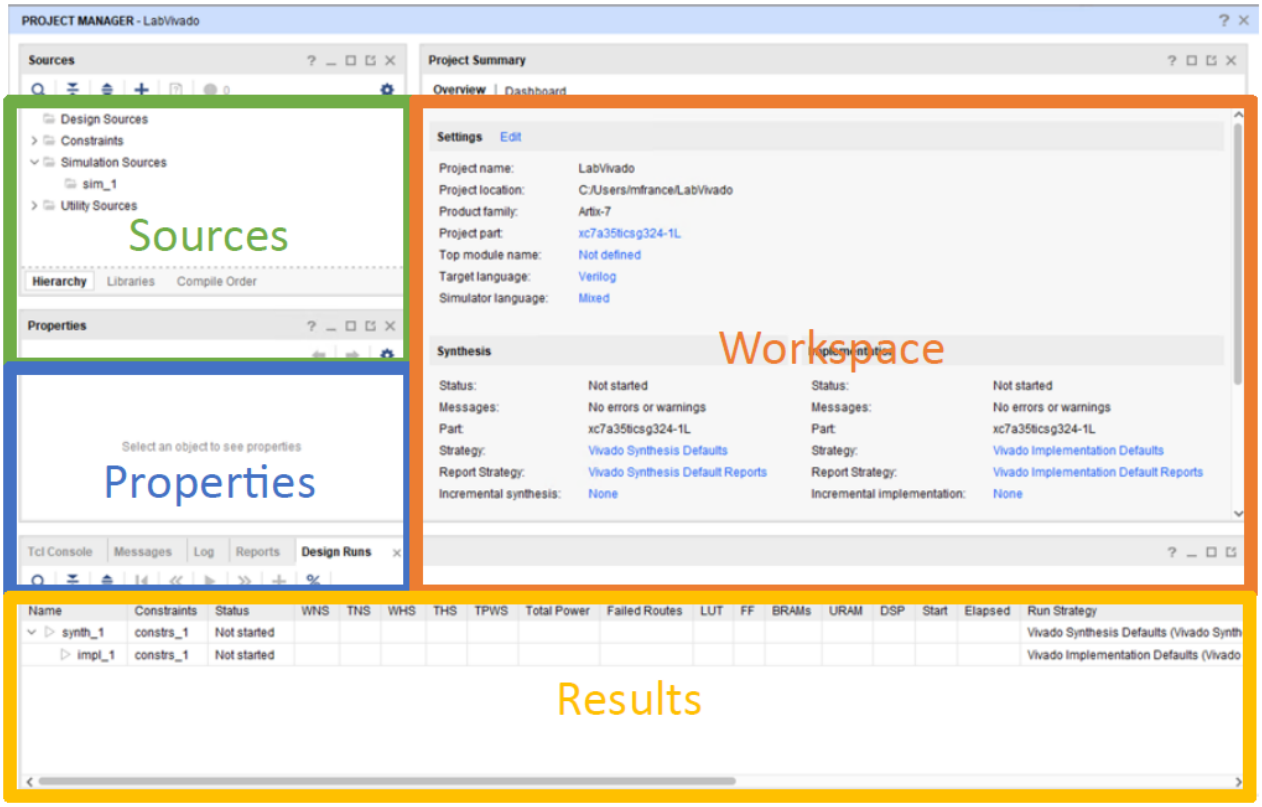


Il pannello sulla sinistra è probabilmente la finestra più importante da conoscere. Permette di passare tra i diversi sotto-tool di Vivado e procedendo dall'alto al basso segue gli step necessari a compilare un progetto.

Ogni step è caratterizzato da una sezione del pannello:

- **Project Manager**
  - accesso ai settaggi del progetto;
  - aggiunta di nuovi sorgenti e di componenti già sviluppati (IP) al progetto;
- **RTL Analysis**
  - generazione di una visualizzazione grafica del risultato dell'analisi dei sorgenti forniti;
- **Synthesis**
  - traduzione della descrizione dell'hardware RTL in una lista (gerarchica) di elementi logici (look-up-table, flip-flop, interconnessioni) disponibili nel dispositivo selezionato;
  - in questo step Vivado determina se il design rispetta i vincoli temporali e quante/quali risorse verranno utilizzate.
- **Implementation**
  - Assegnazione delle porte di ingresso/uscita
  - allocazione degli elementi logici sintetizzati, ottimizzazione dei collegamenti tra un elemento ed un altro;
  - verifica finale dei vincoli temporali includendo il tempo di propagazione dei segnali nei percorsi logici.
- **Program and Debug**
  - generazione dei file binari da caricare nel chip;
  - accesso all'Hardware Manager per l'effettiva comunicazione con l'FPGA.

Il Project Manager

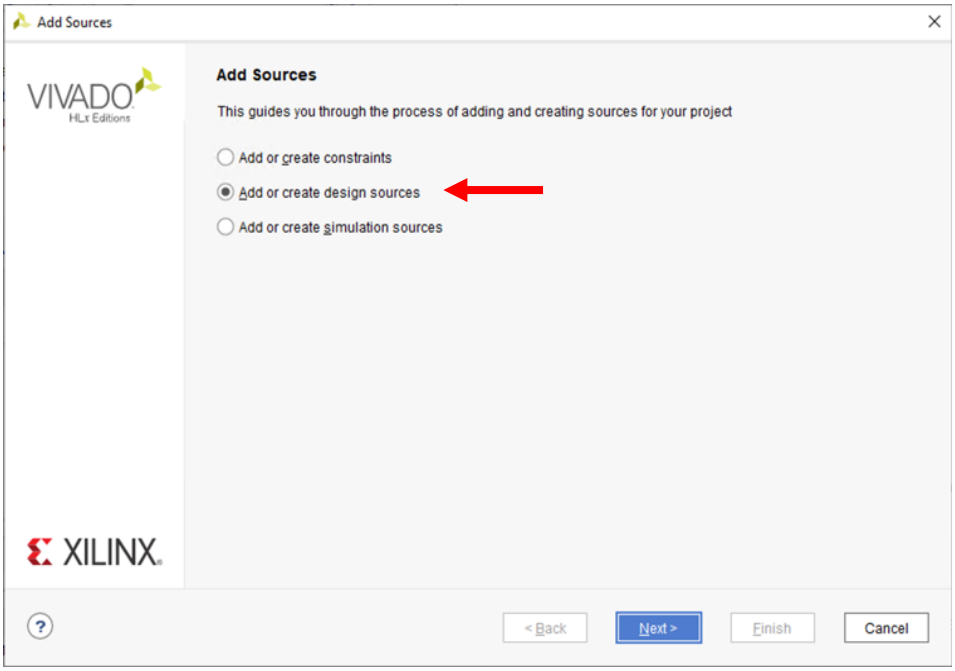


La parte di destra della schermata cambia in base alla visualizzazione corrente. La modalità iniziale di Project Manager è quella utilizzata per gestire i sorgenti (nel nostro caso dei file VHDL e XDC) del progetto e consiste di 4 sezioni indipendenti (“Sources”, “Properties”, “Workspace”, “Results”).

I file HDL che descrivono il progetto sono contenuti all’interno dei **Design Sources**, eventuali vincoli da applicare in fase di compilazione e sintesi si trovano sotto **Constraints** e infine file aggiuntivi utilizzati in simulazioni sono inclusi sotto **Simulation Sources**.

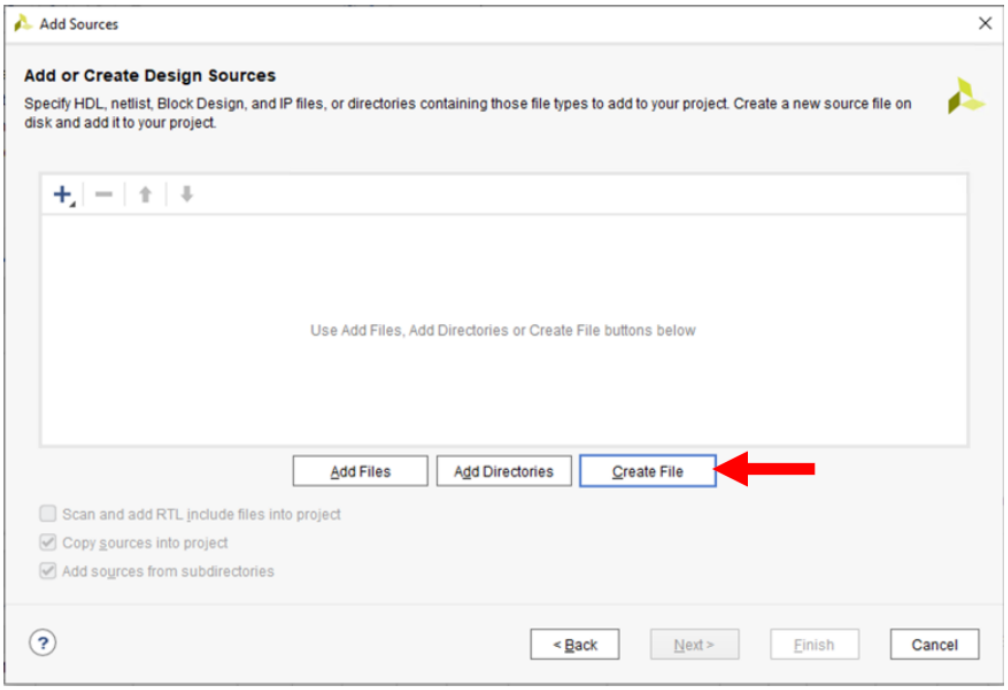
Dato che nessun file è attualmente aperto, il Workspace mostra il sommario del progetto con alcune informazioni di base

In nostro primo sorgente



Il primo file VHDL che aggiungeremo eseguirà alcune operazioni combinatorie tra due interruttori e mostrerà il risultato tramite i LED. Nuovi sorgenti possono essere generati premendo Add Sources tramite il "Flow Navigator" o premendo sul pulsante + all'interno del pannello "Sources".

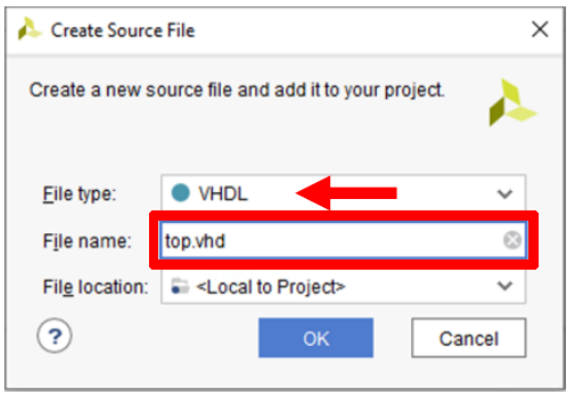
Dopo aver avviato la procedura guidata scegliamo il tipo di sorgente che vogliamo creare.



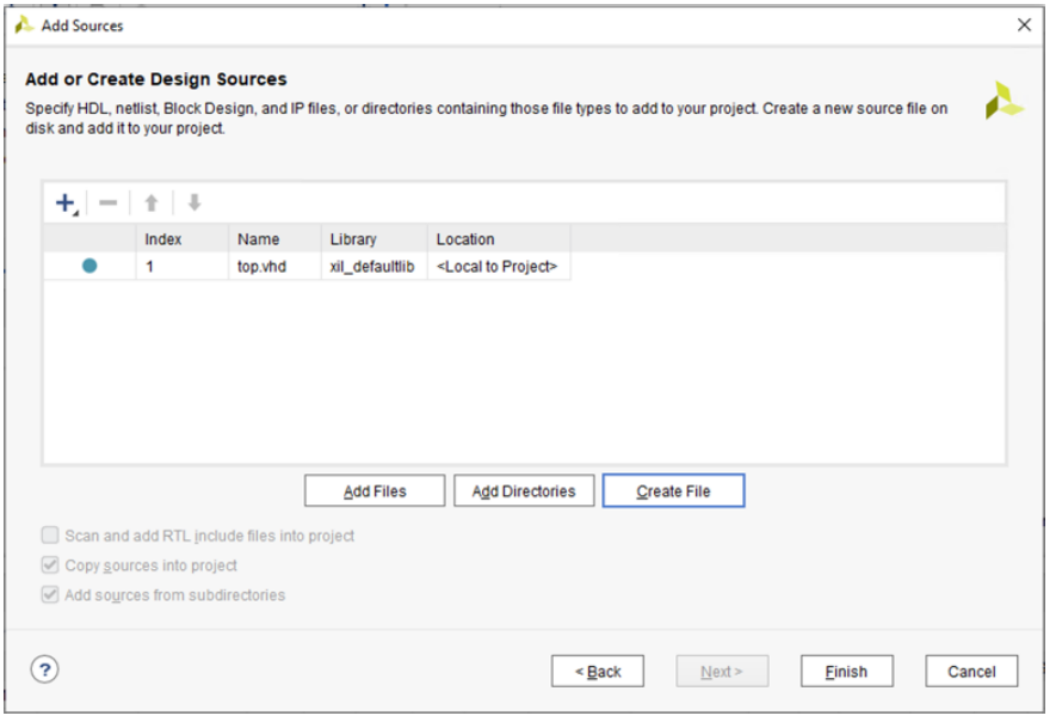
Il passo successivo permette di scegliere tra importare un file esistente o crearne uno nuovo premendo sul pulsante **Create File.**

Cliccando su Create File, si apre una finestra popup. Nel popup viene chiesto il tipo del file da creare ed il nome. Selezionate **VHDL** come tipo di file e scegliete il nome che preferite (top.vhd) aggiungendo l'estensione ".vhd".

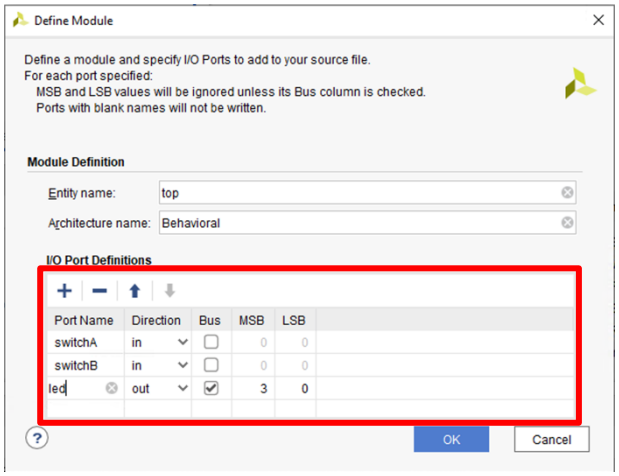
Come per il progetto, NON utilizzate spazi.



Una volta controllato che il nuovo file si trova effettivamente nella lista, premere **Finish**



Per ogni nuovo sorgente si aprirà una finestra che permette di definire in maniera veloce le porte del circuito descritto dal nostro file.



Create due porte di un singolo bit in ingresso e un vettore di 4 bit in uscita:

- “switchA”, direzione Input
- “switchB”, direzione Input
- “led”, direzione Output, Bus con MSB=3 LSB=0

Queste informazioni servono a pre-generare il sorgente, possono essere modificate direttamente agendo su quest’ultimo.

Premendo “Ok” il file verrà infine generato.

Aggiungere il contenuto

Il file generato dal wizard dovrebbe contiene uno scheletro di base (“--" identifica un commento) Tra “is” e “begin” aggiungiamo alcuni segnali che dovremo pilotare:

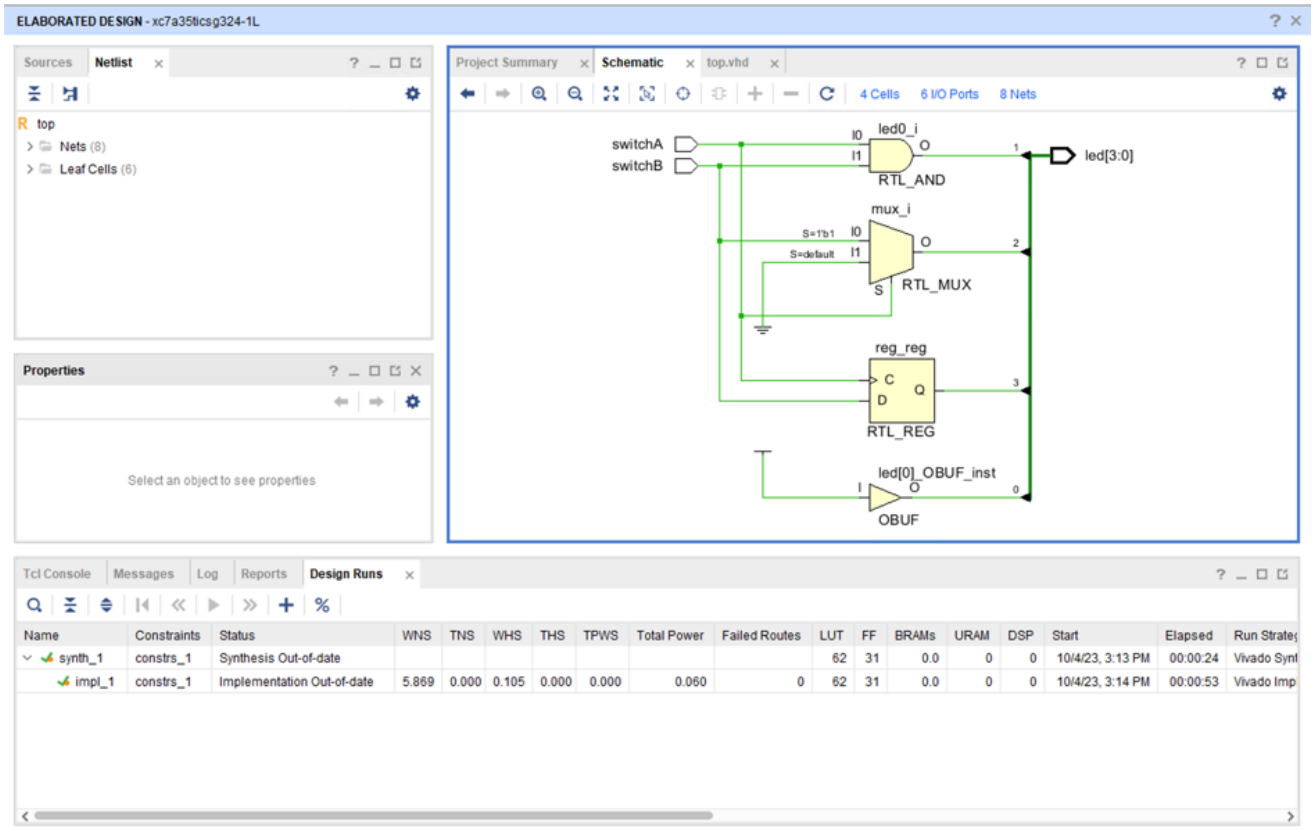
```
signal mux: std_logic;
signal reg: std_logic;
```

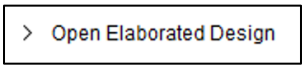
tra “begin” ed “end” aggiungiamo le operazioni che vogliamo compiere:

```
led(0) <= '1';
led(1) <= switchA and switchB;

mux <= switchA when switchB = '1' else '0';
led(2) <= mux;

process (switchA)
begin
     if(rising_edge(switchA)) then
       reg <= switchB;
     end if;
end process;
led(3) <= reg;
```

Questo codice accende il LED 0, assegna al LED 1 l’AND tra i due switch, LED2 è un multiplexer controllato da switchB e LED 3 è un registro dove switchA funziona da clock.

Elaborated design



A questo punto premendo su  nel “Flow Navigator”, Vivado mostrerà il circuito corrispondente alla vostra logica.

Verificate che sia compatibile con quanto avevate in mente, dopodiché chiudete l’*Elaborated Design* premendo sulla croce in alto a destra ***(non quella di Vivado!)***

Aggiunta di un file di constraints (XDC)

Prima di procedere alla compilazione, dobbiamo assicurarci che i pin di input/output siano effettivamente corrispondenti a quelli connessi a dispositivi sulla scheda.

Per aggiungere questo “vincolo” il produttore della scheda fornisce solitamente un Costraints File ma può succedere di doverlo scrivere da zero partendo dallo schematico della scheda.

Create un nuovo file come precedentemente ma, alla prima finestra, selezionate “Add or create costraints” e questa volta scegliete “Add Files” e scegliete il file “simple.xdc” che si trova sul Desktop.

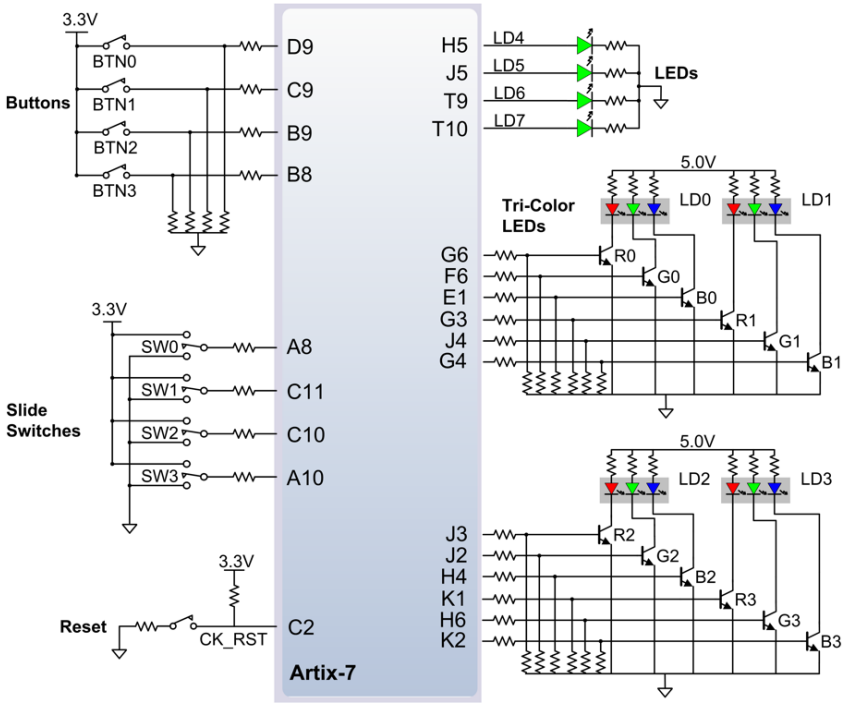
Nota: *Assicuratevi che la spunta “copy costraints files into project” sia presente!!*

Come vedete il file contiene righe simili tra loro:

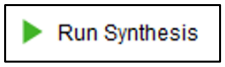
```
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports
{ switchA }];
```

Questo identifica univocamente il pin a cui associare la porta `switchA` e lo standard da utilizzare

Oooooops, ci siamo dimenticati di inserire il constraints per `switchB` … potete aggiungerlo voi??
Questa parte dello schematico della scheda vi sarà molto utile!

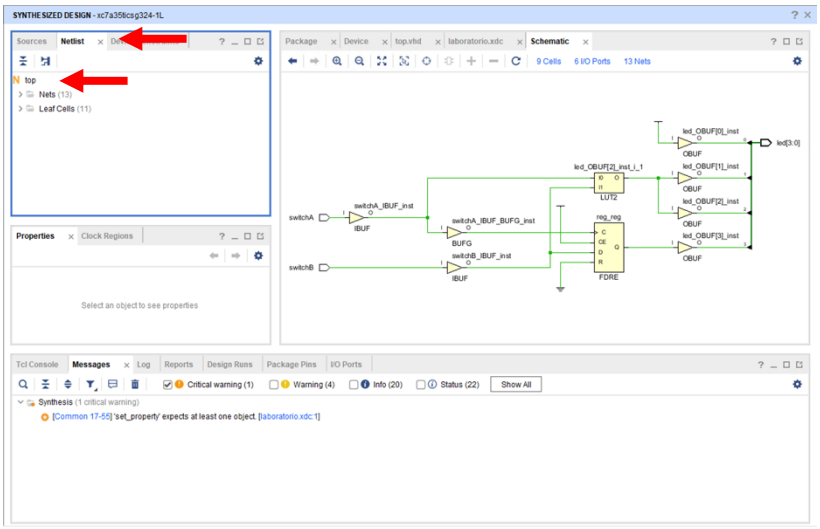


Sintesi del Progetto

Quando siete convinti, premete 

sul Flow Navigator o il corrispondente pulsante  sulla barra degli strumenti e selezionando “Run Sysnthesis”.
Dopo una conferma, questo avvierà il primo step di compilazione. L’operazione avviene in background ed è rimarcata dall’icona in alto a destra.
Potete seguire lo stato della compilazione utilizzando le finestre in basso, in particolare “Design Runs” mostra una barra di caricamento, “Messages” mostrerà eventuali errori e/o warnings e “Log” mostrerà svariate informazioni sulle operazioni in corso.

Una volta sintetizzato il progetto scegliete l’opzione “Open Synthetized design”. Accanto a “Sources” si aprirà un nuovo pannello **Netlist**. All’interno di questo selezionate il vostro modulo e premete sul pulsante Schematic  .

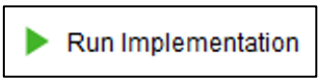


Esplorate le differenze tra questo e il precedente “Elaborated design”:

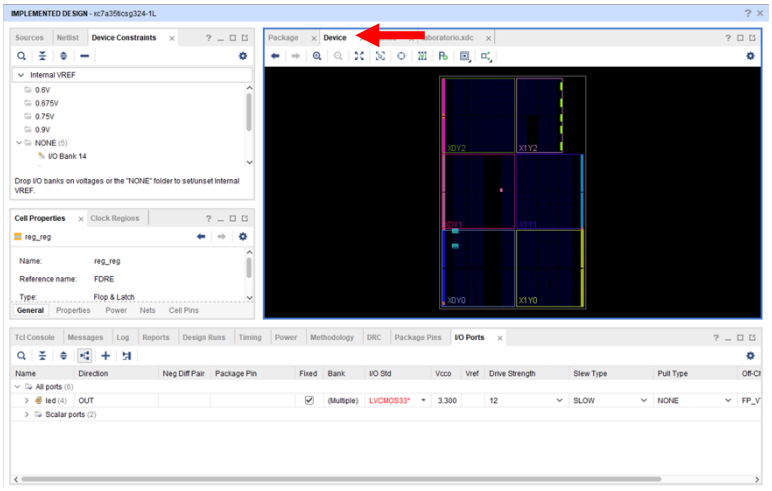
- Cosa sono le LUT e questi *BUF?
- dove è finito il mio multiplexer????

Fatto questo chiudete il design.

Implementazione del Progetto

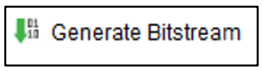
A questo punto siamo pronti all’implementazione finale del progetto!
Se tutto è corretto è sufficiente premere 

sul “Flow Manager” o il corrispettivo pulsante  nella barra degli strumenti, scegliendo poi “Run Implementation”.



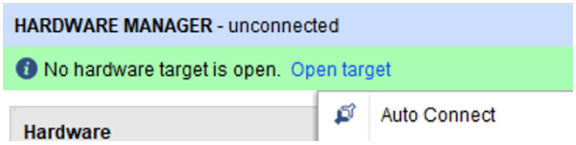
Una volta terminato potete scegliere “Open Implemented Design” e cliccare su **Device** per ammirare il vostro codice implementato in hardware (sono i blocchi celeste).

Siamo adesso pronti a generare il file bitstream (con estensione “.bit”) corrispondente a questo design.

Come prima è sufficiente premere 

sul Flow Navigator o il corrispettivo bottone  nella Toolbar.

Programmare l'FPGA



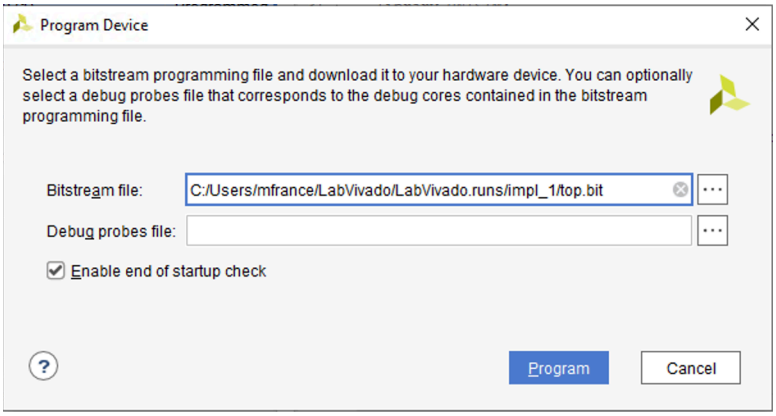
Una volta generato il bitstream si può procedere a caricarlo sul dispositivo. Il componente **Hardware Manager** di Vivado si occupa proprio di questo, lo trovate come ultimo pulsante nel Flow manager o sceglierlo come opzione al termine della generazione del bitstream.

Una volta aperto premete sul link **Open Target** all'interno del banner verde e infine su "auto Connect". Questo farà apparire la vostra FPGA nella finestra delle connessioni.

Per programmare il dispositivo cliccate su 

nel banner verde sul bottone "Program" all'interno della sezione "Hardware Manager" del "Flow Navigator".

Si aprirà questa la finestra:



Il "Bitstream file" dovrebbe essere auto-popolato dalla compilazione precedente.
In caso contrario utilizzate il pulsante  per navigare fino a *<Project Directory>/<Project Name>.runs/impl_1/* e selezionare il file corretto.
Ora premete **Program** e verificate il funzionamento del dispositivo.

Notate come il valore dei LED siano aggiornati simultaneamente (se non ci credete prendete l'oscilloscopio!): le quattro istruzioni che avete inserito sono eseguite realmente in parallelo.

Variazioni sul tema: HDL sequenziale

Se si vuole realizzare una logica sequenziale, sincronizzata da un segnale di **clock** ci sono alcune cose da considerare.

- Va aggiunta una porta di input su cui ricevere il segnale dall'oscillatore a 100 MHz presente sulla scheda (diciamo col nome di `clk100`)
- Vanno aggiunte queste due righe al file dei constraint:

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk100 }];
create_clock -add -name sys_clk -period 10.00 -waveform {0 5} [get_ports { clk100 }];
```

Notate come, oltre al PIN viene definito un clock con un periodo di 10 ns con transizione al tempo 0 ns e 5 ns.

A questo punto si possono usare costrutti VHDL che descrivono la logica sequenziale:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
Port (
      clk100 : in STD_LOGIC;
      switchA : in STD_LOGIC;
      switchB : in STD_LOGIC;
      led : out STD_LOGIC_VECTOR(3 downto 0)
);
end top;

architecture behavioral of top is
      signal counter : STD_LOGIC_VECTOR(23 downto 0) := X"000000";
begin
      process (clk100)
      begin
            if(rising_edge(clk100)) then
                  if(switchA = '1') then
                        counter <= X"000000";
                  else
                        counter <= counter + '1';
                  end if;
            end if;
      end process;

      led(3 downto 0) <= counter(23 downto 20);
end behavioral;
```

Questo codice definisce un registro a 24 bit che è mantenuto a "0" finche switchA è acceso e che conta liberamente quanto è spento. I quattro bit più significativi sono mostrati sui led.

Esercizio finale: il semaforo

Si vuole realizzare un semaforo che stia 1.5 s sul rosso, 0.9 s sul verde e 0.6 s sul giallo.

Rispettivamente questi corrispondono ai valori 150000000 clk ticks ~= X''A000000''
90000000 clk ticks ~= X''6000000''
60000000 clk ticks ~= X''4000000''
Si modellizza questo circuito come una macchina a stati a ciascuno dei quali è associato un colore. Sarà anche necessario un contatore a 28 bit per tener traccia dello scorrimento del tempo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity semaforo is
Port (
          clk100 : in STD_LOGIC;
      reset_btn : in STD_LOGIC;
          green_led : out STD_LOGIC;
          yellow_led : out STD_LOGIC;
      red_led : out STD_LOGIC
);
end semaforo;

architecture behavioral of semaforo is
      type state_type is (red_state, green_state, yellow_state);
      signal state : state_type;
      signal counter : STD_LOGIC_VECTOR(27 downto 0) := X"0000000";
begin
      process (clk100)
      begin
            if(rising_edge(clk100)) then
                  if(reset_btn = '1') then
                        counter <= X"A000000";
                        state <= red_state;
                  else
                        case state is
                        when red_state =>
                            -- inserite il codice qui !!!!
                        when green_state =>
                            -- inserite il codice qui !!!!
                            end if;
                        when yellow_state =>
                            -- inserite il codice qui !!!!
                            end if;
                        end case;
             end if;
            end if;
      end process;
      red_led <= '1' when state = red_state else '0';
      yellow_led <= '1' when state = yellow_state else '0';
     green_led <= '1' when state = green_state else '0';
end behavioral;
```