

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Ingegneria dell'Informazione

Corso di Sistemi Embedded (Ing. L. Pomante)

A.A. 2012/2013

HomeLab #4: FPGA

(Esercizi di VHDL e introduzione a ISE Design Suite)

a cura dell'Ing. Fabio Federici (fabio.federici@univaq.it)

HomeLab vincolato: restituzione kit entro 5 giorni dalla consegna

Versione: 1.0 del 06/12/2012

Nota: tutti gli esempi mostrati sono stati testati utilizzando *ISE Design Suite 2012 (14.2)* sul sistema operativo Windows 7. ISE Design Suite è compatibile anche con versioni precedenti di Windows (Vista e XP, benché su quest'ultimo il tool *PlanAhead* sembra non funzionare) o su sistemi GNU/Linux. Per tali sistemi il pieno funzionamento del software e il corretto riconoscimento dell'hardware non è stato verificato.

HomeLab #4 Kit

- Spartan 3AN Starter Kit;
- Alimentatore;
- Cavo USB A to B;
- Convertitore USB – Seriale;

Software necessario

- ISE Design Suite 2012.3 (14.3) [[download](#)], demo da 30 giorni.
- *puTTY* [[download](#)], o un qualunque emulatore di terminale con funzionalità di serial console (*HyperTerminal*, *minicom*)

Preparazione

Installazione di ISE Design Suite

Scaricare dal sito di Xilinx il software e richiedere una licenza per lo stesso. Si può richiedere una licenza demo da 30 giorni che permette di sfruttare pienamente le funzionalità del programma per tale periodo di tempo. La licenza *Web Edition* permette invece di utilizzare il software con funzionalità limitate.

Installare ISE selezionando la versione *System Edition* quando richiesto.

Verifica dei tool installati

Verificare che tutti i tool siano stati installati correttamente. In particolare, provare ad avviare il *Project Navigator* e poi *Planhead*. Se quest'ultimo non dovesse avviarsi correttamente, navigare fino a <ISE Design Suite Install directory>\ISE_DS\PlanAhead\tps\win32 ed eseguire *vcrcedit_x86.exe* per riparare l'installazione di Visual C++, quindi riavviare il sistema.

Spartan 3AN Starter Kit

Leggere la guida del kit di prova [\[link\]](#) e verificare prima di cominciare che la scheda sia correttamente configurata. Fare attenzione a **non spostare**/smarrire i *jumper* e magari controllare preliminarmente la correttezza della loro posizione.

Verifica della comunicazione tra PC e board

Collegare la scheda al PC tramite il cavo USB, alimentare e accendere la scheda portando l'interruttore *power* sulla posizione ON.

Il sistema operativo dovrebbe rilevare un nuovo dispositivo e chiedere l'installazione dei driver. Evitare la ricerca dei driver su windows update e far cercare automaticamente il driver in locale. Il driver dovrebbe venire rilevato automaticamente se installato correttamente durante il l'installazione di ISE.

Per verificare che la comunicazione possa avvenire correttamente avviare il tool *iMPACT* (nella cartella ISE del menu Start, cercare ISE Design Tools – 32 bit tools).

Facciamo creare al programma un progetto in modo automatico e facciamogli configurare il dispositivo via JTAG (lasciamo svolgere a iMPACT l'identificazione in modo automatico).

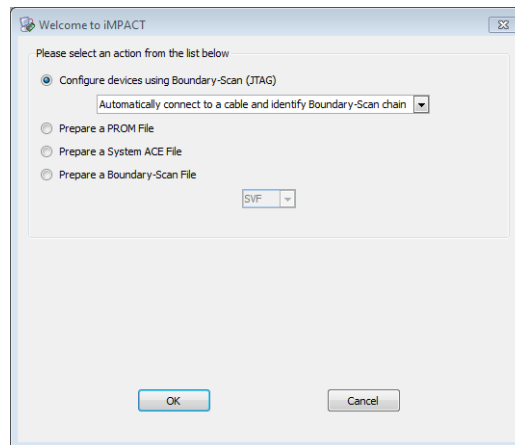


Figura 1 - Schermata di benvenuto di iMPACT

Se la comunicazione avviene correttamente, dovrebbero essere mostrati due dispositivi così come riportato in figura. Nella parte bassa dello schermo dovrebbe inoltre essere notificata la corretta identificazione del dispositivo.

Si può a questo punto chiudere iMPACT, evitando di svolgere le operazioni successive proposte in automatico dal programma.

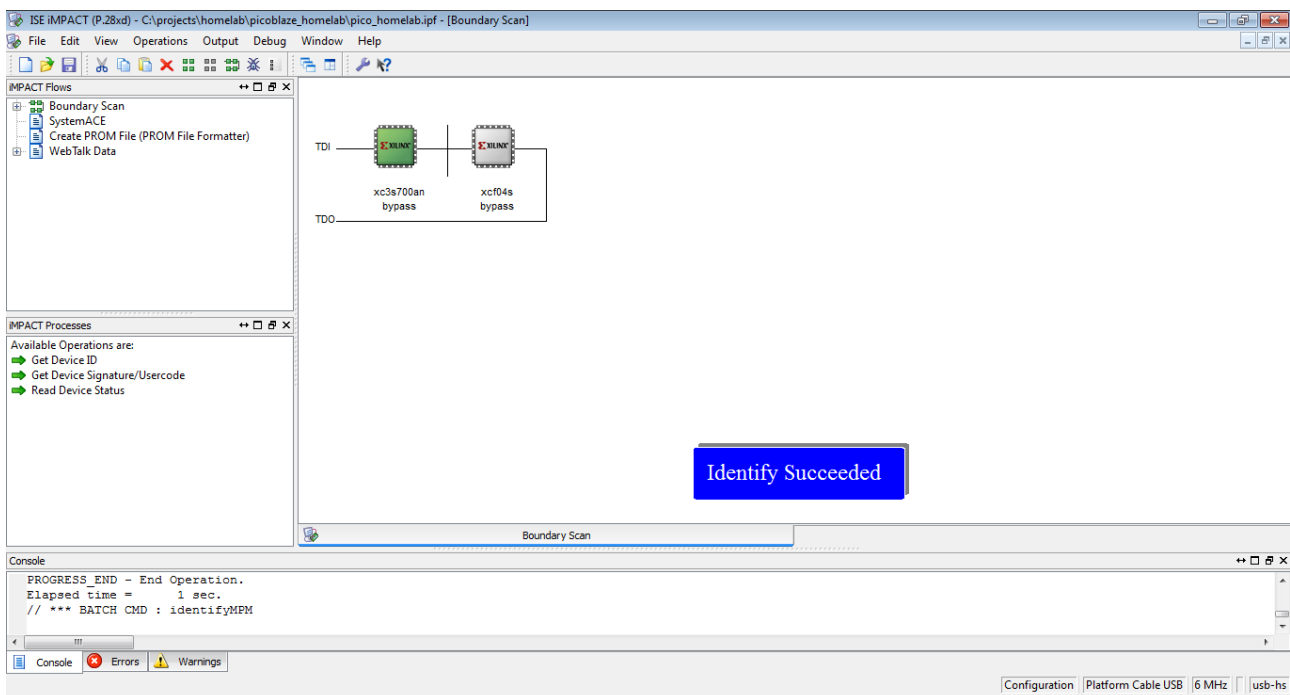


Figura 2 - Schermata principale di iMPACT

Per approfondire

Standard IEEE 1149.1: Standard Test Access Port and Boundary-Scan Architecture (JTAG)

Spartan 3AN, documentazione [\[link\]](#)

Parte 1

Simulazione e sintesi di semplici circuiti combinatori

Questa prima sezione riprende e approfondisce quanto mostrato nella parte finale del seminario introduttivo al VHDL (Corso di Sistemi Embedded, 31 ottobre 2012). Verrà illustrato in particolare come utilizzare ISE Design Suite per la simulazione e la sintesi di un semplice circuito digitale, l'half adder utilizzato come esempio durante la presentazione.

1.1 – Descrizione VHDL di un Half Adder

Avviare il Project Navigator e creare un nuovo progetto (File > New Project). Verrà avviato un wizard per la configurazione del nuovo progetto.

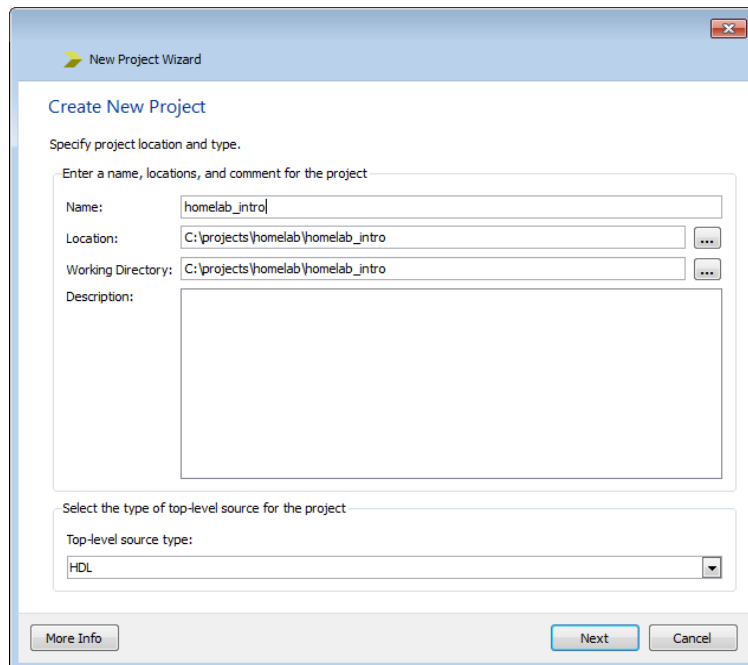
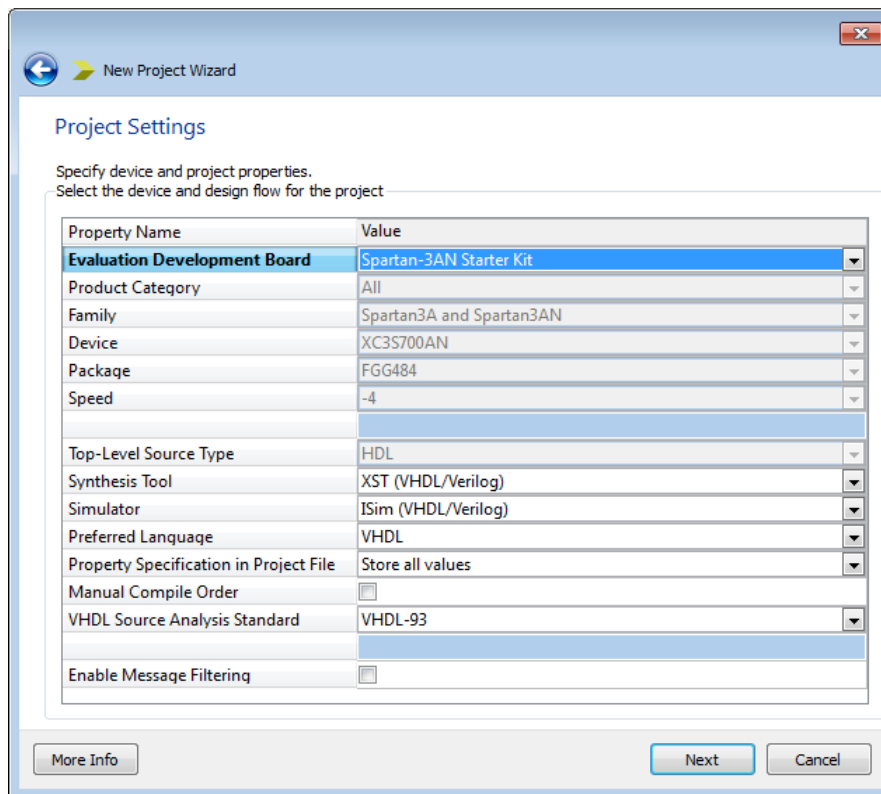


Figura 3 - Wizard di configurazione di un nuovo progetto all'interno del Project Navigator

Compilare i vari campi, inserendo il nome del progetto e la cartella in cui salvarlo (è eventualmente possibile, ma in questo caso non necessario, specificare una directory di lavoro differente). Specificare HDL come tipologia di sorgente top level da utilizzare.

La form che segue (figura 4) permette di specificare una board o un dispositivo target e le principali impostazioni del progetto.



The screenshot shows the 'New Project Wizard' dialog box, specifically the 'Project Settings' tab. The dialog is titled 'New Project Wizard' and has a close button in the top right corner. Below the title bar, there is a section titled 'Project Settings' with the instruction 'Specify device and project properties. Select the device and design flow for the project.' The main area contains a table with two columns: 'Property Name' and 'Value'. The table lists various project settings, including device selection, synthesis tool, simulator, and language. At the bottom of the dialog, there are three buttons: 'More Info', 'Next', and 'Cancel'.

Property Name	Value
Evaluation Development Board	Spartan-3AN Starter Kit
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S700AN
Package	FGG484
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

Figura 4 - Impostazione della piattaforma target

Selezionare lo *Spartan-3AN Starter Kit* tra le Evaluation Development Board supportate nativamente dall'ambiente di sviluppo. In questo modo verranno automaticamente configurate in maniera corretta le impostazioni riguardanti l'FPGA target.

Scegliere i tool nativi XST e ISim come strumenti di sintesi e simulazione e specificare VHDL (VHDL-93) come linguaggio di descrizione hardware da utilizzare.

Verrà proposto un riassunto della configurazione impostata. Se tutte le impostazioni sono corrette, cliccare su Finish. Verrà a questo punto presentata la seguente schermata:

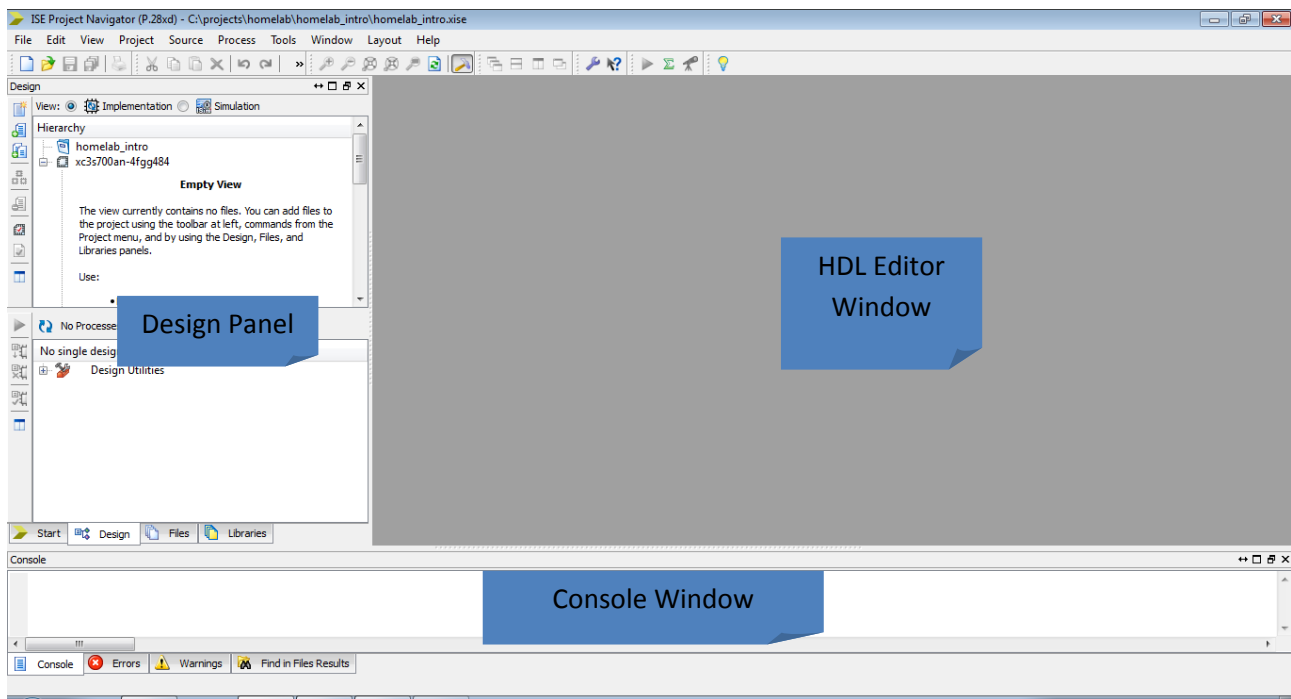


Figura 5 - Schermata principale del Project Navigator

La schermata è suddivisa in tre aree principali:

- *Design Panel*: situato nell'area a sinistra dello schermo, racchiude due finestre principali, la *Source window* in cui vengono mostrati tutti i file sorgente associati al progetto corrente e la *Process window* in cui vengono elencate tutte le operazioni che è possibile compiere sul file sorgente selezionato.
- *Console Panel*: situata nella parte inferiore dello schermo, serve a visualizzare i messaggi di stato (inclusi errori e warning, visualizzabili individualmente in due sotto-tab selezionabili).
- *HDL Editor Window*: l'area più grande a destra, permette la visualizzazione del codice sorgente del file selezionato nella *Source window*.

Il progetto creato non contiene per ora nessun file sorgente. Aggiungiamone dunque uno facendo click con il tasto destro del mouse sul nome del nostro progetto all'interno della *Source window* e selezionando la voce *New Source*.

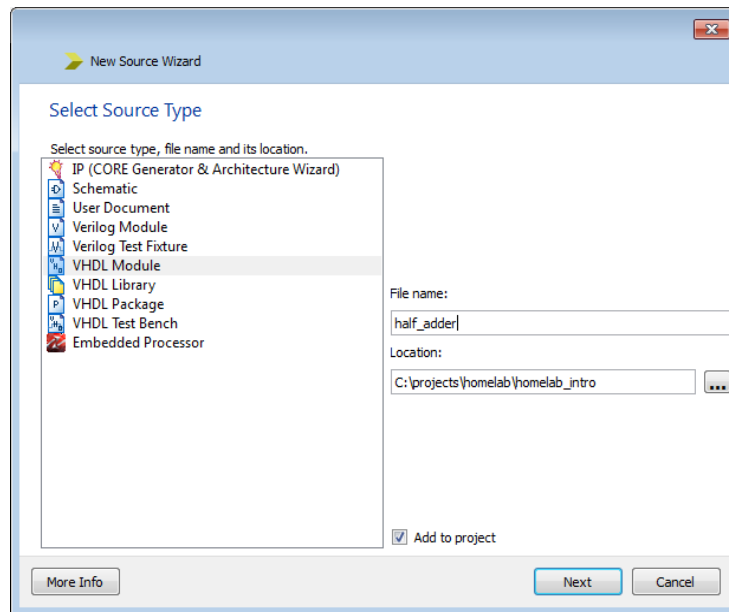


Figura 6 - Inserimento di un nuovo modulo VHDL

Selezioniamo *VHDL module* come tipologia di file sorgente e specificare un nome per lo stesso (ad esempio `half_adder`). La form successiva consente di specificare la entity (nome e porte) e la architecture (nome) del dispositivo

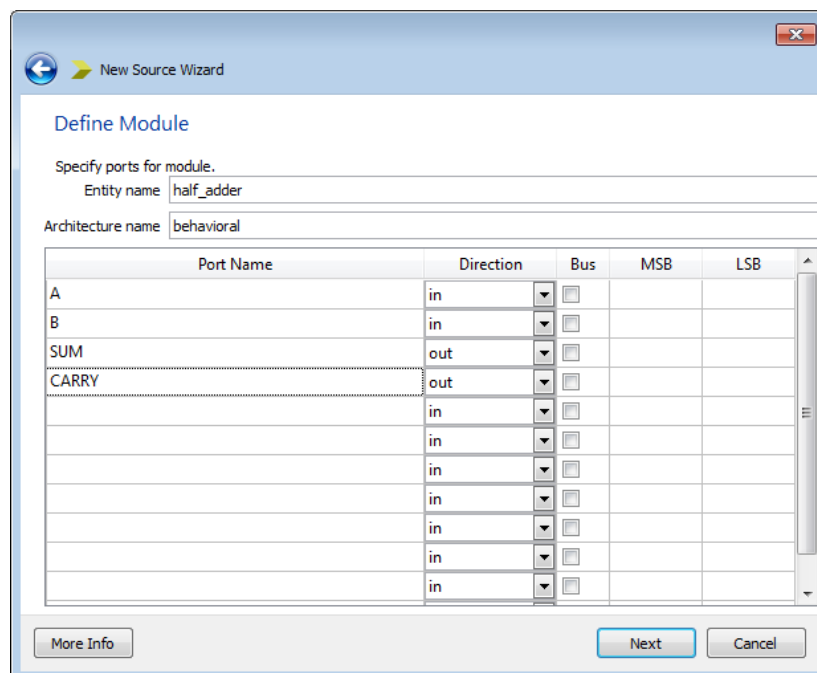


Figura 7 - Wizard per l'impostazione del modulo creato

Compilati i vari campi secondo la struttura dell'half adder, cliccare su next e verificare la correttezza delle impostazioni specificate; quindi terminare la creazione del file sorgente cliccando su finish.

Aprire il nuovo file sorgente generato facendo doppio click su di esso nella source window. Al suo interno sarà stato automaticamente impostato lo scheletro del codice VHDL necessario. Si può notare come per completare la descrizione sia necessario solo specificare la architecture. Completiamo quindi la descrizione così come visto nel corso del seminario:

```

ARCHITECTURE behav OF half_adder IS

BEGIN

    SUM <= A xor B;
    CARRY <= A and B;

END behav;

```

A questo punto la descrizione del modulo half adder risulta completa. Salvare il file sorgente modificato.

1.2 - Simulazione

Verifichiamo innanzitutto che il comportamento del blocco sia quello atteso. A tal scopo, è necessario creare un test bench tramite il quale sollecitare in maniera opportuna il circuito appena descritto. Aggiungiamo al progetto un ulteriore file sorgente, ma stavolta selezioniamo come tipologia *VHDL Testbench*:

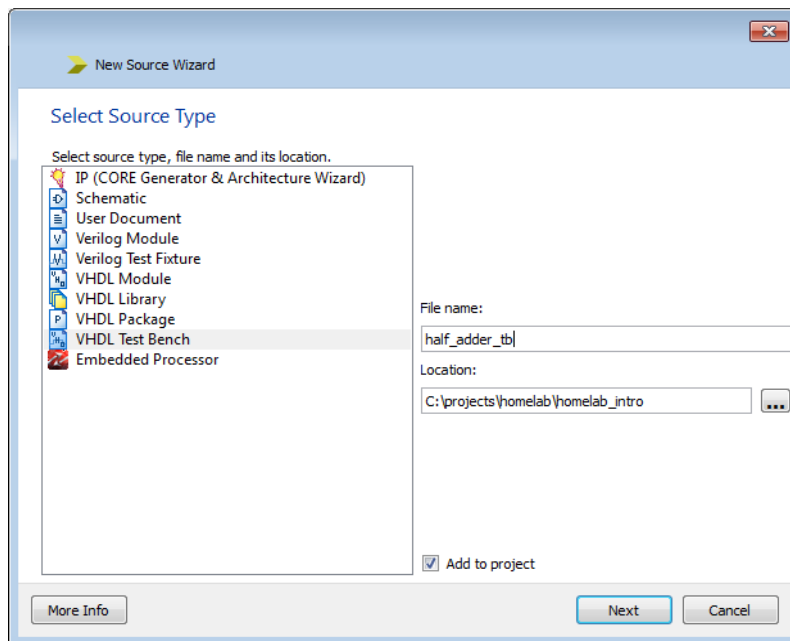


Figura 8 - Creazione di un testbench

Il wizard di creazione ci proporrà direttamente l'inclusione di un file sorgente (ovvero di istanziare all'interno del testbench uno dei blocchi già descritti nel progetto). Selezioniamo l'unico blocco presente nel nostro progetto, ovvero l'half adder e andiamo avanti fino alla conclusione del wizard. Modifichiamo la vista nella Source window in Simulation, in modo da vedere anche il sorgente del testbench generato.

Notiamo come lo scheletro creato:

- Comprenda una entity senza alcuna porta (un testbench non necessita di segnali in ingresso o in uscita);
- Preveda già la dichiarazione del componente half adder e la sua la creazione di una sua istanza nel corpo della architecture;
- Preveda la dichiarazione di due segnali corrispondenti agli ingressi dell'half adder e due segnali corrispondenti alle uscite. Questi segnali sono interni al testbench e servono per poter applicare degli stimoli e osservare cosa accade in uscita.
- Per default venga creata una parte di codice relativa alla gestione di un eventuale segnale di clock;

Il circuito in esame è di tipo combinatorio, per cui si può eliminare dalla architecture il processo riferito al clock. Si ottiene in tal modo il seguente scheletro di codice:

```

LIBRARY ieee;

```



```

USE ieee.std_logic_1164.ALL;

ENTITY ha_testbench IS
END ha_testbench;

ARCHITECTURE behavior OF ha_testbench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT half_adder
    PORT(
        A : IN  std_logic;
        B : IN  std_logic;
        SUM : OUT std_logic;
        CARRY : OUT std_logic
    );
    END COMPONENT;

    --Inputs
    signal SIG_A : std_logic := '0';
    signal SIG_B : std_logic := '0';

    --Outputs
    signal SIG_SUM : std_logic;
    signal SIG_CARRY : std_logic;

    constant period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: half_adder PORT MAP (
        A => SIG_A,
        B => SIG_B,
        SUM => SIG_SUM,
        CARRY => SIG_CARRY
    );

    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        wait for 100 ns;

        -- insert stimulus here

    end process;

END;

```

Nel codice è stata mantenuta la costante period (eliminando il riferimento al clock): tale costante potrà essere utilizzata per definire l'intervallo temporale di variazione dei segnali di ingresso. Attraverso il costrutto PROCESS e l'istruzione wait è possibile modellare l'evoluzione dei segnali di ingresso. Esaminare ad esempio il listato seguente e provare a disegnare l'andamento temporale atteso dei segnali.

```

A_stim_proc: process
begin

    SIG_A <='0';
    wait for period;
    SIG_A <='1';
    wait for period;

end process;

B_stim_proc: process

```

```

begin

    SIG_B <='0';
    wait for 2*period;
    SIG_B <='1';
    wait for 2*period;

end process;

```

Definita l'evoluzione degli ingressi è possibile procedere con la simulazione: nella Process window completare i due step disponibili, ovvero verificare la correttezza sintattica del codice VHDL e avviare la simulazione. Quest'ultimo passo avvierà il simulatore ISim, che mostrerà automaticamente le forme d'onda risultanti (cliccare sul pulsante Zoom to Full View per visualizzare le forme d'onda con il corretto livello di dettaglio).

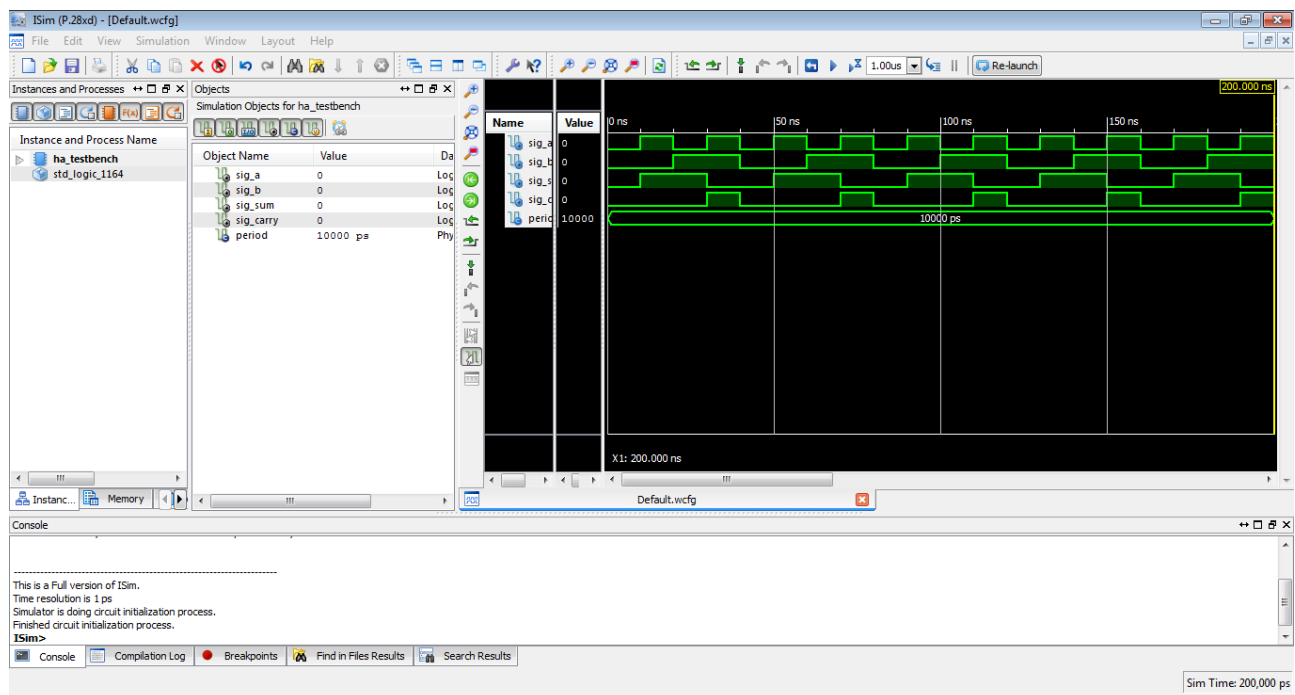


Figura 9 - Schermata principale del simulatore ISIM

Verificare che il comportamento delle uscite sia effettivamente quello atteso sulla base della tabella di verità dell'half adder.

1.2 -Sintesi

Chiudere ISim, tornare al Project Navigator e ripristinare la vista Implementation nella Source Window. Tra le azioni disponibili, cliccare su *Synthesize* e attendere che il processo di sintesi dell'half adder venga portato a termine. Espandendo il menu Synthesize sarà possibile visualizzare lo schematico RTL generato dalla sintesi e lo schematico vero e proprio riferito all'implementazione del circuito sul dispositivo target.

1.3 – Approfondimenti

Provare a scrivere la descrizione VHDL di altri semplici circuiti combinatori (es. multiplexer, demultiplexer, decoder ecc.). Verificare che il comportamento dei circuiti descritti sia quello atteso tramite simulazione.

Circuiti sequenziali, I/O e vincoli

Per poter sfruttare l'hardware sulla board abbiamo bisogno di far comunicare i blocchi implementati all'interno dell'FPGA con l'esterno. A tal proposito, vediamo come realizzare all'interno dell'FPGA un primo circuito sequenziale (contatore) e come visualizzarne le uscite sui LED presenti sulla board [5].

2.1 – Contatore

Creiamo un nuovo progetto nel Project Navigator (chiamato ad esempio homelab_counter) e aggiungiamo un nuovo file sorgente contenente la entity hl_counter. Di seguito si riporta la possibile struttura del contatore:

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

ENTITY hl_counter IS
    PORT ( clk      : in    STD_LOGIC;
          reset     : in    STD_LOGIC;
          count_out : out   STD_LOGIC_VECTOR (3 downto 0));
END hl_counter;

ARCHITECTURE behav OF hl_counter IS

    SIGNAL temp_count  : std_logic_vector(3 downto 0) := "0000";
    SIGNAL slow_clk    : std_logic;
    SIGNAL clk_divider : std_logic_vector(23 downto 0) := x"000000";

BEGIN

    clk_division : PROCESS (clk, clk_divider)
    BEGIN

        IF clk'event AND clk = '1' THEN
            clk_divider <= clk_divider + 1;
        END IF;

        slow_clk <= clk_divider(23);

    END PROCESS;

    counting : PROCESS(reset,slow_clk, temp_count)
    BEGIN

        IF reset = '1' THEN
            temp_count <= "0000";
        ELSIF slow_clk'event AND slow_clk= '1' THEN
            IF temp_count < 9 then
                temp_count <= temp_count + 1;
            ELSE
                temp_count <= "0000";
            END IF;
        END IF;

        count_out <= temp_count;
    END PROCESS;
END;
```

NOTA: è necessario generare uno “slow clock” perché altrimenti il ritmo del conteggio sarebbe troppo elevato e risulterebbe invisibile all’occhio umano

```
END PROCESS;
```

```
END behav;
```

Definita l'architettura del contatore, si può passare all'implementazione. Come passo intermedio, si può in ogni caso verificare la correttezza funzionale e delle specifiche temporali simulando il circuito. A tal scopo occorrerà definire un testbench così come fatto nel corso dell'esercizio precedente e istanziare al suo interno il contatore.

Passando invece all'implementazione vera e propria, si vuole che una volta scaricato il bitstream le uscite del contatore pilotino direttamente quattro dei LED presenti sulla board. A tal scopo, andiamo a definire un vincolo per l'implementazione, agendo sugli *User Constraint File*.

Nella Process window espandere il menu User Constraint per visualizzare le varie opzioni disponibili. È possibile definire vincoli pre e post sintesi.

Possiamo in questo caso cliccare direttamente sulla voce I/O Pin Planning (PlanAhead) Post Synthesis. In tal modo verrà eseguita automaticamente la sintesi del circuito descritto e si avvierà il processo di specifica dei vincoli. Lasciamo creare il file UCF necessario in modo automatico quando ci viene richiesto (si può verificare come tale file venga effettivamente aggiunto nella gerarchia del progetto) e attendiamo l'avvio del tool PlanAhead. Terminato l'avvio del programma e l'importazione del progetto da parte del nuovo tool avviato, si può provvedere all'assegnazione dei pin.

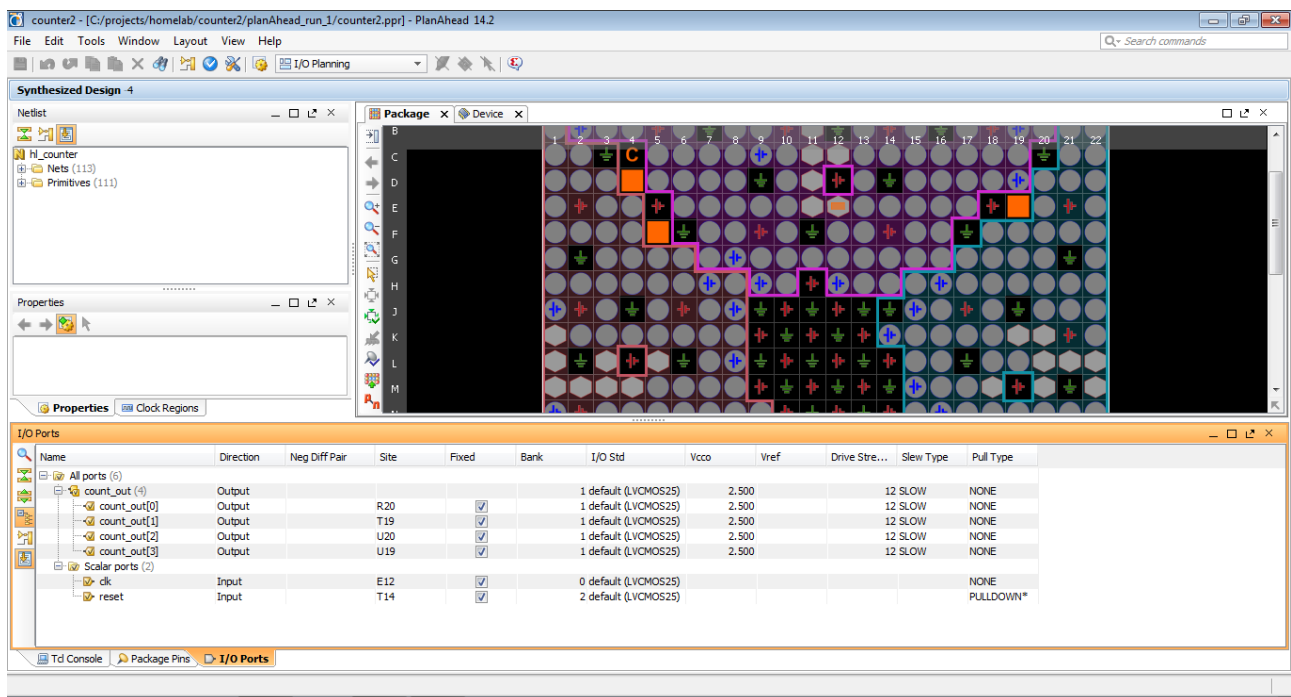


Figura 10 - Assegnazione dei vincoli tramite PlanAhead

Lo Spartan 3AN Starter Kit dispone di 8 user LED, ciascuno collegato a un pin dell'FPGA:

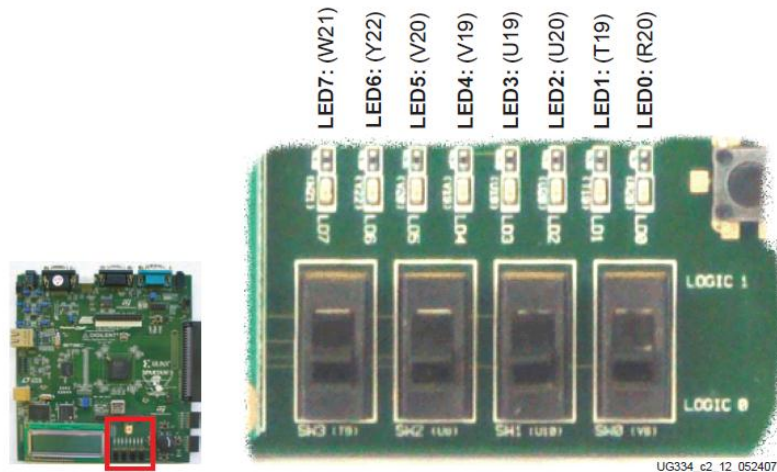


Figura 11 - LED sullo Spartan 3AN Starter Kit

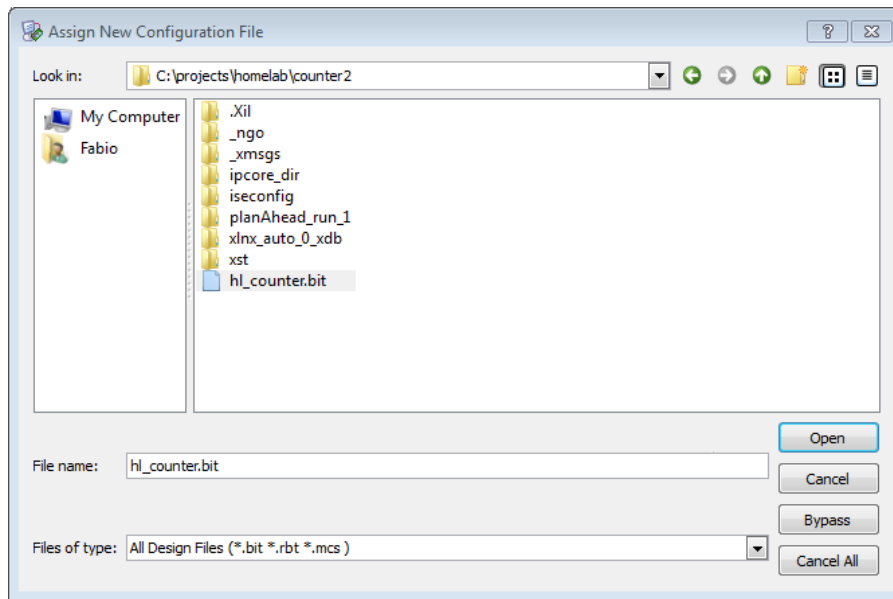
Facciamo click sulle sotto-voci corrispondenti alle singole linee di uscita e assegniamo a ciascuna il corretto pin, (campo *Site*, così come riportato in figura). Impostiamo anche le linee di reset, collegata al pushbutton north (facendo attenzione a specificare la presenza di un resistore di pull-down) e di clock (collegando al segnale di clock del nostro contatore la linea del clock a 50 MHz proveniente dall'oscillatore esterno). Salviamo e usciamo da PlanAhead: il tool dovrebbe aver aggiornato automaticamente il file UCF creato prima con i vincoli specificati (doppio click sul file ucf per verificare).

```
# PlanAhead Generated physical constraints

NET "count_out[0]" LOC = R20;
NET "count_out[1]" LOC = T19;
NET "count_out[2]" LOC = U20;
NET "count_out[3]" LOC = U19;
NET "reset" LOC = T14 | PULLDOWN;
NET "clk" LOC = E12;
```

Completiamo quindi il processo di sintesi e generiamo il bitstream cliccando sulla voce *Generate Programming File* della process window. Collegata la scheda al PC, possiamo configurare l'FPGA cliccando sulla voce *Configure Target Device* della process window.

Un popup ci avvertirà a questo punto che non esiste ancora alcun progetto di iMPACT e ci inviterà a crearne uno. Cliccando ok si aprirà la GUI di iMPACT: creiamo un nuovo progetto scegliendo la procedura manuale e salvando il file nella directory principale del progetto in corso; nel wizard scegliamo ancora la voce *Configure Device using boundary scan (JTAG)* e nel menu collegato l'identificazione automatica del programming cable. Se tutto è andato bene nella prova precedente, anche in questo caso pc e board dovrebbero comunicare correttamente e l'inizializzazione dovrebbe andare a buon fine. Stavolta clicchiamo Yes alla richiesta di assegnare i file di configurazione e indichiamo ad iMPACT il file dal medesimo nome del top level e dall'estensione .bit che si trova all'interno della directory principale del progetto.



Clicchiamo Bypass e Ok nelle finestre che seguono e salviamo il progetto. A questo punto possiamo impostare il dispositivo e cliccando con il tasto destro sullo stesso, scegliere la voce *Program FPGA only*.

Al termine dell'operazione i LED 0-3 sulla board dovrebbero illuminarsi secondo la sequenza di un conteggio binario.

2.2 - Rilevatore di sequenza (Macchine a Stati Finiti)

Vediamo come realizzare un circuito in grado di rilevare una precisa sequenza di bit su una linea di ingresso. Si vuole realizzare un rilevatore sincrono, ovvero avente in ingresso una linea dati (x) e un segnale di clock. I dati verranno letti in corrispondenza dei fronti positivi del segnale di clock. L'uscita (y) sarà pari a 1 solo nel momento in cui viene rilevata la sequenza. È possibile testare questo circuito utilizzando due switch per "simulare" i segnali di clock e di ingresso.

Il seguente diagramma degli stati descrive un rilevatore di sequenza in cui la parola da rilevare sull'ingresso seriale (101) è "hardcoded".

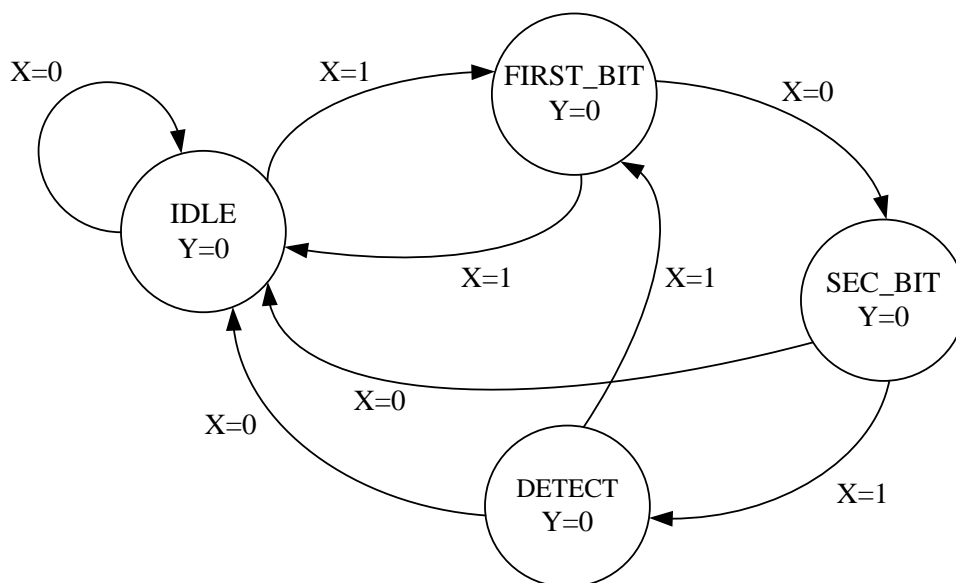


Figura 12 - Macchina a stati in grado di rilevare la sequenza "101" sull'ingresso x

Di seguito una bozza di codice VHDL per una macchina a stati (Moore) che implementa il comportamento descritto dal diagramma precedente.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL;

-----

ENTITY seq_detector IS
PORT( x:          IN std_logic;
      clock: IN std_logic;
      reset: IN std_logic;
      y:          OUT std_logic
);
END seq_detector;

-----

ARCHITECTURE behav OF seq_detector IS

    -- define the states of FSM model

    TYPE state_type IS (IDLE, FIRST_BIT, SEC_BIT, DETECT);
    SIGNAL next_state, current_state: state_type;

BEGIN

    -- state register
    state_reg: PROCESS(clock, reset)
    BEGIN

        IF (reset='1') THEN
            current_state <= IDLE;
        ELSIF (clock'event and clock='1') THEN
            current_state <= next_state;
        END IF;

    END PROCESS;

    -- next state logic
    comb_logic: PROCESS(current_state, x)
    BEGIN

        CASE current_state IS

            WHEN IDLE =>
                IF x='0' THEN
                    next_state <= IDLE;
                ELSIF x='1' THEN
                    next_state <= FIRST_BIT;
                END IF;

            WHEN FIRST_BIT =>
                IF x='0' then
                    next_state <= SEC_BIT;
                ELSIF x='1' THEN
                    next_state <= IDLE;
                END IF;

            WHEN SEC_BIT =>
                IF x='0' then
                    next_state <= IDLE;
                ELSIF x='1' THEN
                    next_state <= DETECT;
                END IF;

            WHEN DETECT =>
                IF x='0' THEN
                    next_state <= IDLE;
```

La definizione di un tipo dato enumerato è estremamente comoda per la rappresentazione degli stati

```

        ELSIF x='1' THEN
            next_state <= FIRST_BIT;
        END IF;

    WHEN OTHERS =>
        next_state <= IDLE;

    END CASE;

END PROCESS;

-- Output Logic
y <= '1' WHEN current_state = DETECT ELSE '0';

END behav;

```

Terminare l'esercizio verificando il funzionamento del rilevatore (costruire lo user constraint file per specificare il corretto collegamento tra il rilevatore di sequenze, i LED e gli switch sulla board e procedere quindi alla programmazione dell'FPGA così come fatto nell'esercizio precedente).

2.3 – Approfondimenti

GCD (Opzionale)

Provare ad implementare la macchina a stati dell'esempio sull'algoritmo GCD visto a lezione.

[\[link\]](#)

PicoBlaze

Il PicoBlaze è un soft processor a 8 bit di tipo RISC distribuito da Xilinx. Gli esempi che seguono riguarderanno l'utilizzo di tale processore: si raccomanda pertanto la lettura della guida [\[link\]](#) o del capitolo sulla comunicazione seriale in [2].

3.1 – Input/Output e programmazione di base

È possibile scaricare il codice del PicoBlaze dalla seguente [pagina](#), cliccando sul link download e indicando come tipologia *PicoBlaze for Spartan-3 & Virtex II/Pro FPGAs*.

Il file scaricato conterrà:

- Una cartella VHDL contenente i file sorgente del processore (e di altri elementi che esamineremo in seguito)
- La documentazione del PicoBlaze;
- I tool necessari per il suo utilizzo, tra cui l'assemblatore;

Avviare il Project Navigator e creare un nuovo progetto. Scompattare per comodità l'archivio scaricato all'interno della directory del progetto.

Nella source window facciamo click con il tasto destro del mouse sul progetto creato e scegliamo la voce Add Source dal menu contestuale. Aggiungiamo quindi al progetto il file KCPSM3.vhd dalla cartella VHDL prima creata. In questo modo avremo importato nel nostro progetto il PicoBlaze.

Dovremo a questo punto generare la rom contenente le istruzioni. Creiamo quindi (ad esempio direttamente all'interno della cartella Assembler) un file chiamato ad esempio testprog.psm (è importante che il nome del file abbia lunghezza inferiore agli otto caratteri) contenente il seguente programma scritto nell'assembly del Picoblaze:

```
; read from switches and display to the LEDs
start: INPUT  s0,00
OUTPUT s0,01
JUMP start
```

Provare a scrivere l'equivalente pseudo-codice del programma riportato e dedurre la funzionalità implementata.

Avviamo una command window, portiamoci nella directory Assembler e eseguiamo il tool con la seguente sintassi:

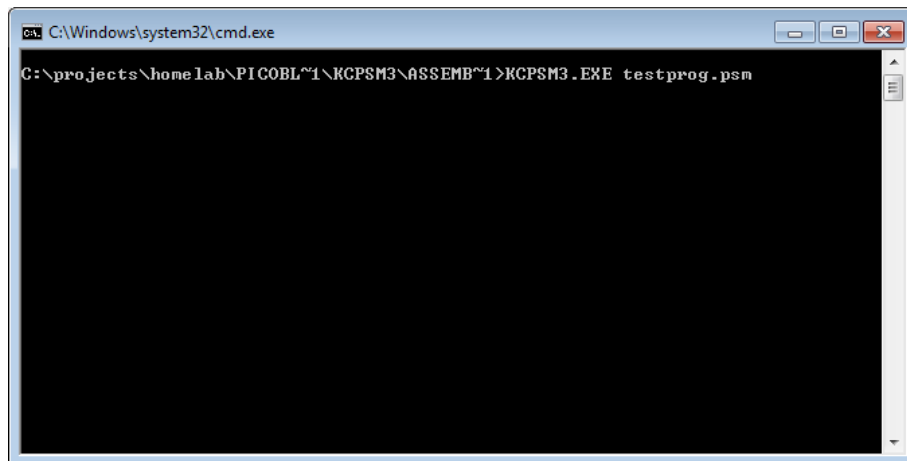


Figura 13 – Utilizzare il comando: KCPSM3.exe testprog.psm

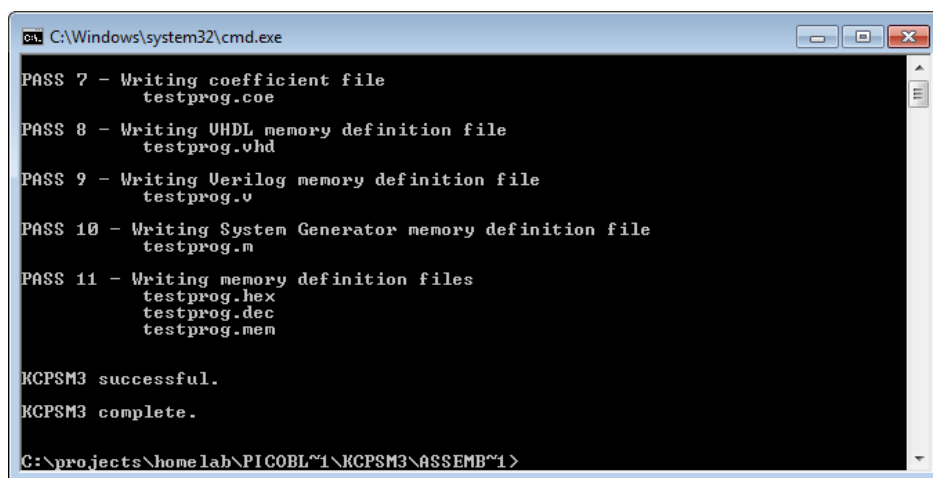


Figura 14 - La generazione della ROM istruzioni

L'utility genera vari file: tra essi il file testprog.vhd corrisponde alla descrizione in VHDL della ROM istruzioni contenente il programma precedentemente scritto in assembly.

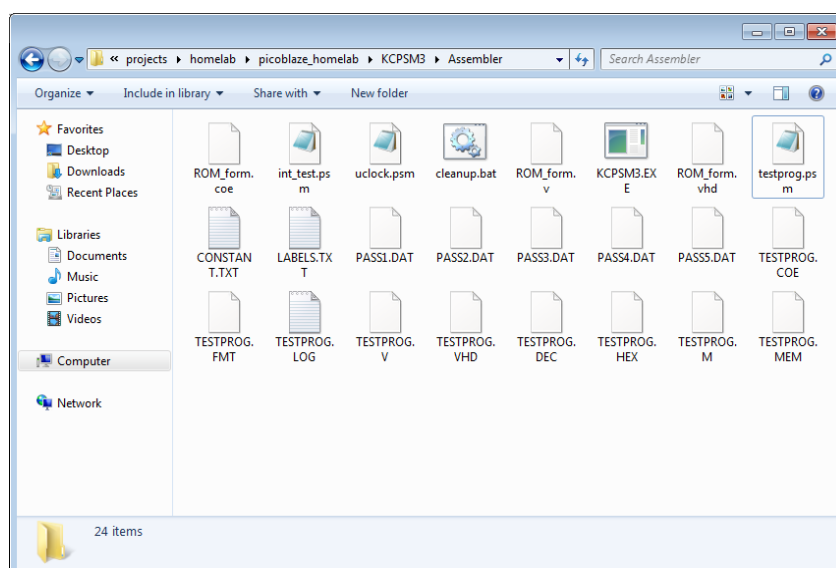


Figura 15 - File generati dall'utility KCPSM.exe

Importiamo all'interno del progetto il file TESTPROG.VHD. A questo punto si disporrà all'interno del progetto di due blocchi distinti, il PicoBlaze e la sua ROM istruzioni.

Il breve programma in assembly riportato in precedenza non fa altro che leggere i dati in ingresso (indirizzo 0x00) e copiarli in uscita (indirizzo 0x01), indefinitamente. Le istruzioni di input e output usano il registro s0 come momentaneo buffer dati.

Possiamo verificare la funzionalità effettiva del programma collegando quattro linee di ingresso e quattro linee di uscita rispettivamente a switch e LED sulla board. In tal modo utilizziamo il PicoBlaze semplicemente per rilevare che uno switch è passato in posizione *on* e accendere un led di segnalazione.

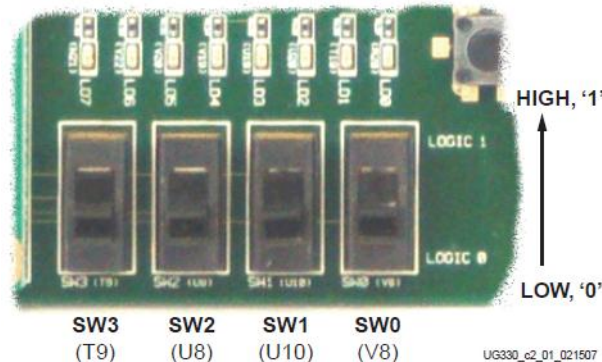


Figura 16 - Gli switch sullo Spartan 3AN Starter Kit

Per collegare PicoBlaze e ROM avremo bisogno di creare un top level in cui istanziare i due moduli e i relativi segnali di collegamento. Avremo inoltre bisogno di memorizzare i dati in ingresso e in uscita in due registri temporanei (accessibili in modo sincrono sulla base delle linee di indirizzo e degli strobe di lettura e scrittura del processore).

Creiamo dunque un nuovo file (vhdl module) ed istanziamo al suo interno il PicoBlaze, la memoria di programma e i due registri da utilizzare rispettivamente per memorizzare i dati acquisiti dagli switch e relativi alla configurazione dei LED. Un esempio di come fare è riportato nel listato seguente.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY pico_homelab is
PORT ( switches : in std_logic_vector(3 downto 0);
      LEADS      : out std_logic_vector(3 downto 0);
      clk        : in std_logic);
END pico_homelab;

ARCHITECTURE behav OF pico_homelab IS

    COMPONENT kcpsm3
    PORT( address      : out std_logic_vector(9 downto 0);
          instruction   : in std_logic_vector(17 downto 0);
          port_id       : out std_logic_vector(7 downto 0);
          write_strobe  : out std_logic;
          out_port      : out std_logic_vector(7 downto 0);
          read_strobe   : out std_logic;
          in_port       : in std_logic_vector(7 downto 0);
          interrupt     : in std_logic;
          interrupt_ack : out std_logic;
          reset         : in std_logic;
          clk           : in std_logic);
    END COMPONENT;

    COMPONENT testprog
```

```

PORT( address      : in std_logic_vector(9 downto 0);
      instruction  : out std_logic_vector(17 downto 0);
      clk          : in std_logic);
END COMPONENT;

SIGNAL address_sig      : std_logic_vector(9 downto 0);
SIGNAL instruction_sig  : std_logic_vector(17 downto 0);
SIGNAL port_id_sig      : std_logic_vector(7 downto 0);
SIGNAL out_port_sig     : std_logic_vector(3 downto 0);
SIGNAL in_port_sig      : std_logic_vector(3 downto 0);
SIGNAL write_strobe_sig : std_logic;
SIGNAL read_strobe_sig  : std_logic;
SIGNAL interrupt_ack_sig : std_logic;

SIGNAL reset_sig        : std_logic := '0';
SIGNAL interrupt_sig     : std_logic := '0';

BEGIN

processor: kcpsm3
PORT MAP( address => address_sig,
          instruction => instruction_sig,
          port_id => port_id_sig,
          write_strobe => write_strobe_sig,
          out_port (3 downto 0) => out_port_sig,
          out_port (7 downto 4) => OPEN,
          read_strobe => read_strobe_sig,
          in_port (3 downto 0) => in_port_sig,
          in_port (7 downto 4) => "0000",
          interrupt => interrupt_sig,
          interrupt_ack => interrupt_ack_sig,
          reset => reset_sig,
          clk => clk);

program: testprog
PORT MAP( address => address_sig,
          instruction => instruction_sig,
          clk => clk);

process(clk)
begin
    if clk'event and clk='1' then
        if (read_strobe_sig='1' and port_id_sig="00000000") then
            in_port_sig <= switches;
        end if;
    end if;
end process;

-- Connect Output from PicoBlaze
process(clk)
begin
    if clk'event and clk='1' then
        if (write_strobe_sig='1' and port_id_sig="00000001") then
            LEDs <= out_port_sig;
        end if;
    end if;
end process;

end behav;

```

Prima di procedere con la sintesi, specifichiamo tra i vincoli le corrette connessioni dei pin in ingresso e in uscita. Utilizziamo ancora PlanAhead per collegare otto switch e otto LED ai due bus di ingresso e uscita rispettivamente.

Si può a questo punto procedere con la sintesi, il download del bitstream e la verifica del corretto funzionamento del programma scritto.

3.2 – Approfondimenti

Modificare il programma scritto in precedenza includendo anche istruzioni diverse da quelle di input/output (provare ad esempio ad utilizzare le istruzioni di controllo di flusso). Verificare il corretto funzionamento implementando il sistema su FPGA.

Riferimento: PicoBlaze 8 bit Embedded Microcontroller User Guide [\[link\]](#)

Comunicazione Seriale

Xilinx mette a disposizione insieme al sorgente del PicoBlaze anche quello di un controller UART: i relativi file sono contenuti all'interno della cartella VHDL dell'archivio scaricato all'inizio. Possiamo utilizzare questo core per realizzare una comunicazione seriale tra il nostro PC e il PicoBlaze. Si consiglia di consultare preliminarmente la relativa documentazione contenuta tra il materiale del PicoBlaze scaricato, oppure [2].

4.1 – PicoBlaze & UART

Al fine di dimostrare le funzionalità del core UART verrà mostrato come aggiungere al PicoBlaze la sezione di trasmissione [6]. Il completamento dell'esercizio con l'aggiunta della sezione di ricezione viene lasciato al lettore.

Creiamo un nuovo progetto e, per comodità, scompattiamo nuovamente l'archivio contenente i file del PicoBlaze all'interno della directory principale del progetto. Importiamo nel progetto i seguenti file:

- KCPSM3.vhd: il PicoBlaze
- uart_tx.vhd: wrapper che racchiude i componenti della sezione di trasmissione dell'UART
- kcuart_tx.vhd: Constant (K) Compact UART Transmitter, il trasmettitore vero e proprio
- bbfifo_16x8.vhd: descrizione della FIFO utilizzata dal trasmettitore

Il seguente programma in assembly fa in modo che il PicoBlaze trasmetta (in loop, indefinitamente), una stringa di caratteri sulla porta seriale.

```
; UART Transmit Example
;
start: LOAD s1, 48 ;ASCII "H".
CALL xmit          ;Send character.
LOAD s1, 65        ;ASCII "e".
CALL xmit          ;Send character.
LOAD s1, 6c        ;ASCII "l".
CALL xmit          ;Send character.
LOAD s1, 6c        ;ASCII "l".
CALL xmit          ;Send character.
LOAD s1, 6f        ;ASCII "o".
CALL xmit          ;Send character.
LOAD s1, 2c        ;ASCII ", ".
CALL xmit          ;Send character.
LOAD s1, 20        ;ASCII " ".
CALL xmit          ;Send character.
LOAD s1, 49        ;ASCII "I".
CALL xmit          ;Send character.
LOAD s1, 27        ;ASCII "'".
CALL xmit          ;Send character.
LOAD s1, 6d        ;ASCII "n".
CALL xmit          ;Send character.
LOAD s1, 20        ;ASCII " ".
CALL xmit          ;Send character.
LOAD s1, 50        ;ASCII "P".
```

```

CALL xmit          ;Send character.
LOAD s1, 69        ;ASCII "i".
CALL xmit          ;Send character.
LOAD s1, 63        ;ASCII "c".
CALL xmit          ;Send character.
LOAD s1, 6f        ;ASCII "o".
CALL xmit          ;Send character.
LOAD s1, 42        ;ASCII "B".
CALL xmit          ;Send character.
LOAD s1, 6c        ;ASCII "l".
CALL xmit          ;Send character.
LOAD s1, 61        ;ASCII "a".
CALL xmit          ;Send character.
LOAD s1, 7a        ;ASCII "z".
CALL xmit          ;Send character.
LOAD s1, 65        ;ASCII "e".
CALL xmit          ;Send character.
LOAD s1, 20        ;ASCII " ".
CALL xmit          ;Send character.
LOAD s1, 3b        ;ASCII ";".
CALL xmit          ;Send character.
LOAD s1, 29        ;ASCII ")".
CALL xmit          ;Send character.
LOAD s1, 0d        ;ASCII "\r".
CALL xmit          ;Send character.
JUMP start

; Routine to transmit data via RS-232
xmit:INPUT s0, 00   ;read uart status.
AND s0, 80          ;Test buffer full condition
JUMP NZ, xmit       ;Poll buffer full condition
OUTPUT s1, 00       ;If buffer is not full, send byte
RETURN

```

Come già fatto nell'esempio n.3, creiamo un file di programma all'interno della cartella Assembler (ad es. uartprog.psm) e ripetiamo la procedura per la generazione della ROM. Ottenuto il file uartprog.vhd, importiamolo all'interno del progetto.

A questo punto abbiamo tutti i componenti necessari per verificare la comunicazione seriale: un processore, un trasmettitore UART e una ROM di programma contenente le istruzioni per far inviare al processore dei dati sulla UART.

Creiamo un top level per il nostro progetto all'interno del quale istanzieremo e collegheremo le varie componenti in modo opportuno. Di seguito viene riportato un codice di esempio (attenzione a modificare il nome della ROM se al passo precedente avete usato un nome differente da uartprog).

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY pb_uart_top IS
    PORT ( clk: IN std_logic;
           txd: OUT std_logic);
END pb_uart_top;

ARCHITECTURE Behavioral OF pb_uart_top IS

    COMPONENT KCPSM3
        PORT (
            address : out std_logic_vector(9 downto 0);
            instruction : in std_logic_vector(17 downto 0);
            port_id: out std_logic_vector(7 downto 0);
            write_strobe: out std_logic;
            out_port: out std_logic_vector(7 downto 0);
            read_strobe: out std_logic;
            in_port: in std_logic_vector(7 downto 0);

```

```

        interrupt : in std_logic;
        interrupt_ack: out std_logic;
        reset : in std_logic;
        clk: in std_logic
    );
END COMPONENT;

--PicoBlazeROM File
COMPONENT uartprog
    PORT (
        address : in std_logic_vector(9 downto 0);
        instruction : out std_logic_vector(17 downto 0);
        clk: in std_logic
    );
END COMPONENT;

--UART Module
COMPONENT uart_tx
    PORT (
        data_in: in std_logic_vector(7 downto 0);
        write_buffer: in std_logic;
        reset_buffer: in std_logic;
        en_16_x_baud : in std_logic;
        serial_out: out std_logic;
        buffer_full: out std_logic;
        buffer_half_full: out std_logic;
        clk: in std_logic
    );
END COMPONENT;

--signals for processor
SIGNAL address_signal: std_logic_vector(9 downto 0);
SIGNAL instruction_signal: std_logic_vector(17 downto 0);
SIGNAL pb_inp_data: std_logic_vector(7 downto 0);
SIGNAL pb_out_data: std_logic_vector(7 downto 0);
--signals for interfacing uart
SIGNAL baud_count: integer range 0 to 511 := 0;
SIGNAL en_16_x_baud : std_logic;
SIGNAL uart_write: std_logic;

BEGIN

    processor: kcpsm3
        PORT MAP (
            address => address_signal,
            instruction => instruction_signal,
            port_id=> open,
            write_strobe=> uart_write,
            out_port=> pb_out_data,
            read_strobe=> open,
            in_port=> pb_inp_data,
            interrupt => '0',
            interrupt_ack=> open,
            reset => '0',
            clk=> clk
        );

    program: uartprog
        PORT MAP (
            address => address_signal,
            instruction => instruction_signal,
            clk=> clk
        );

    tx_uart: uart_tx
        PORT MAP (
            data_in=> pb_out_data,
            write_buffer=> uart_write,
            reset_buffer=> '0',
            en_16_x_baud => en_16_x_baud,

```



```

        serial_out=> txd,
        buffer_full=> pb_inp_data(7),
        buffer_half_full=> pb_inp_data(0),
        clk=> clk
    );

-- UART Baudrate timer
    baud_timer: PROCESS (clk)
    BEGIN
        IF clk'event and clk= '1' THEN
            IF baud_count= 325 THEN
                baud_count<= 0;
                en_16_x_baud <= '1';
            ELSE
                baud_count<= baud_count+ 1;
                en_16_x_baud <= '0';
            END IF;
        END IF;
    END PROCESS baud_timer;

END Behavioral;

```

Una volta salvato il file dovrebbe essere riconosciuto come top level e avere i restanti file prima importati al di sotto nella gerarchia visualizzata.

Il top level ha solamente una linea di ingresso (il segnale di clock) e una di uscita (la linea tx dell'UART). Sulla board sono disponibili due connettori DB9 (DE-9P) per comunicazione seriale. Useremo il connettore di tipo DCE per collegare la board al nostro PC.

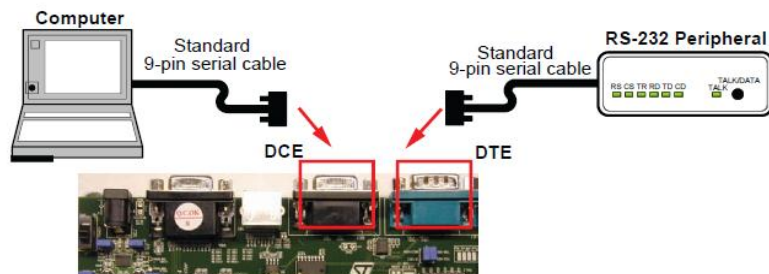


Figura 17 - Connettori DB9 sullo Spartan 3AN Starter Kit

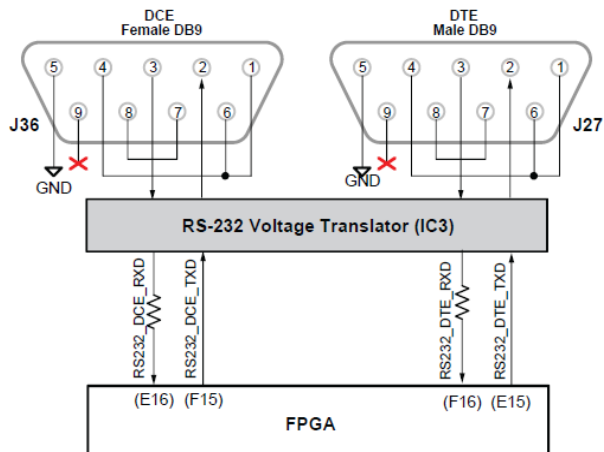


Figura 18 - Connessioni interne dei due connettori DCE e DBE

La figura precedente illustra le connessioni tra FPGA e connettori DB9: notiamo come i collegamenti previsti impediscano l'implementazione del controllo di flusso hardware (i segnali DCD, DTR, DSR sono collegati insieme, così come i segnali RTS e CTS).

Come si nota, la linea RS232_DCE_TXD del connettore è collegata al pin F15 dell'FPGA. Utilizzammo quindi PlanAhead per specificare che la linea txd del nostro top level va collegata a tale pin. Colleghiamo anche il segnale di clock come già visto negli esempi precedenti.

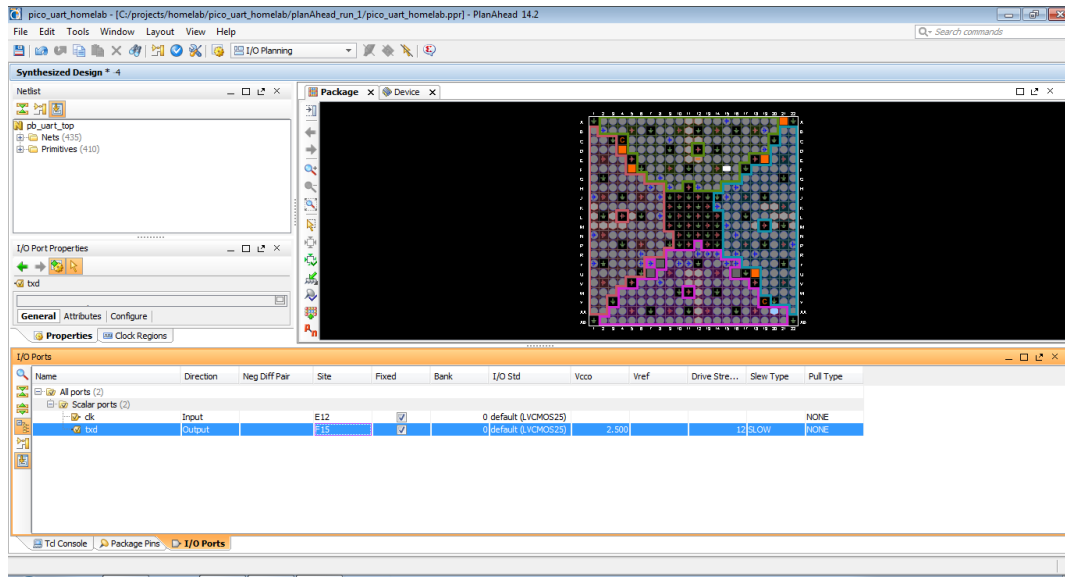


Figura 19- Vincoli per il trasmettitore in PlanAhead

Colleghiamo a questo punto il PC alla board utilizzando il cavo USB – seriale. Inserendo il cavo, windows dovrebbe procedere in modo automatico all'installazione del driver e comunicarci la presenza di una nuova porta seriale (cui assegnerà un identificativo del tipo COMx). Prendiamo nota dell'identificativo assegnato (è sempre possibile controllarlo accedendo alla Gestione periferiche dalle proprietà del computer).

Utilizziamo quindi puTTY per metterci preliminarmente in ascolto sulla porta seriale. Selezioniamo le proprietà dell'interfaccia seriale e specifichiamo le seguenti impostazioni per la comunicazione:

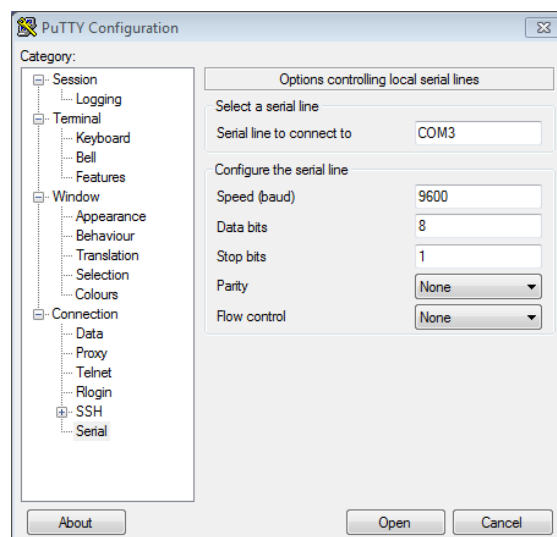


Figura 20 - Impostazioni della comunicazione seriale in puTTY

Torniamo quindi alla schermata contenente i parametri della sessione e avviamo la comunicazione seriale.

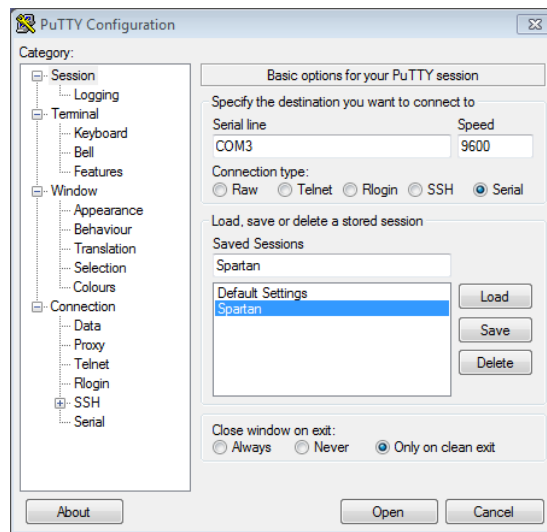


Figura 21 - Gestione della sessione di comunicazione in puTTY

Eseguita la sintesi e programmato il dispositivo, sul terminale dovrebbe essere visualizzato il messaggio inviato dal PicoBlaze.

4.1 – Approfondimenti

Provare a implementare la sezione di ricezione dell'UART

Scopriamo il DNA del nostro dispositivo (Opzionale)

Possiamo utilizzare il PicoBlaze per leggere il DNA dell'FPGA presente sullo starter kit e farcelo comunicare attraverso l'interfaccia seriale o (come illustrato nel tutorial) per visualizzarlo sul display LCD della board

Riferimento: tutorial Xilinx [\[link\]](#)

Paint su PicoBlaze (Opzionale)

Un semplice programma per disegno implementato su PicoBlaze. Viene illustrata anche la gestione di un display VGA e di un mouse PS/2. Si può provare a scaricare i file del progetto e programmare l'FPGA.

Riferimento: tutorial Xilinx [\[link\]](#)

Riferimenti e letture utili

- [1] Spartan 3AN Starter Kit User Guide [\[link\]](#)
- [2] Pong P. Chu – FPGA Prototyping by VHDL Examples - Xilinx Spartan-3 Version, Wiley
- [3] VHDL Tutorial – Learn by examples [\[link\]](#)
- [4] PicoBlaze User Guide [\[link\]](#)
- [5] 4- Bit Counter with Xilinx ISE 9.2 and Spartan 3E [\[link\]](#) – tutorial che ha ispirato l'esempio 3.1
- [6] Using Soft Core Processors: The PicoBlaze RISC Microcontroller [\[link\]](#) - riferimento per l'esempio 4.1