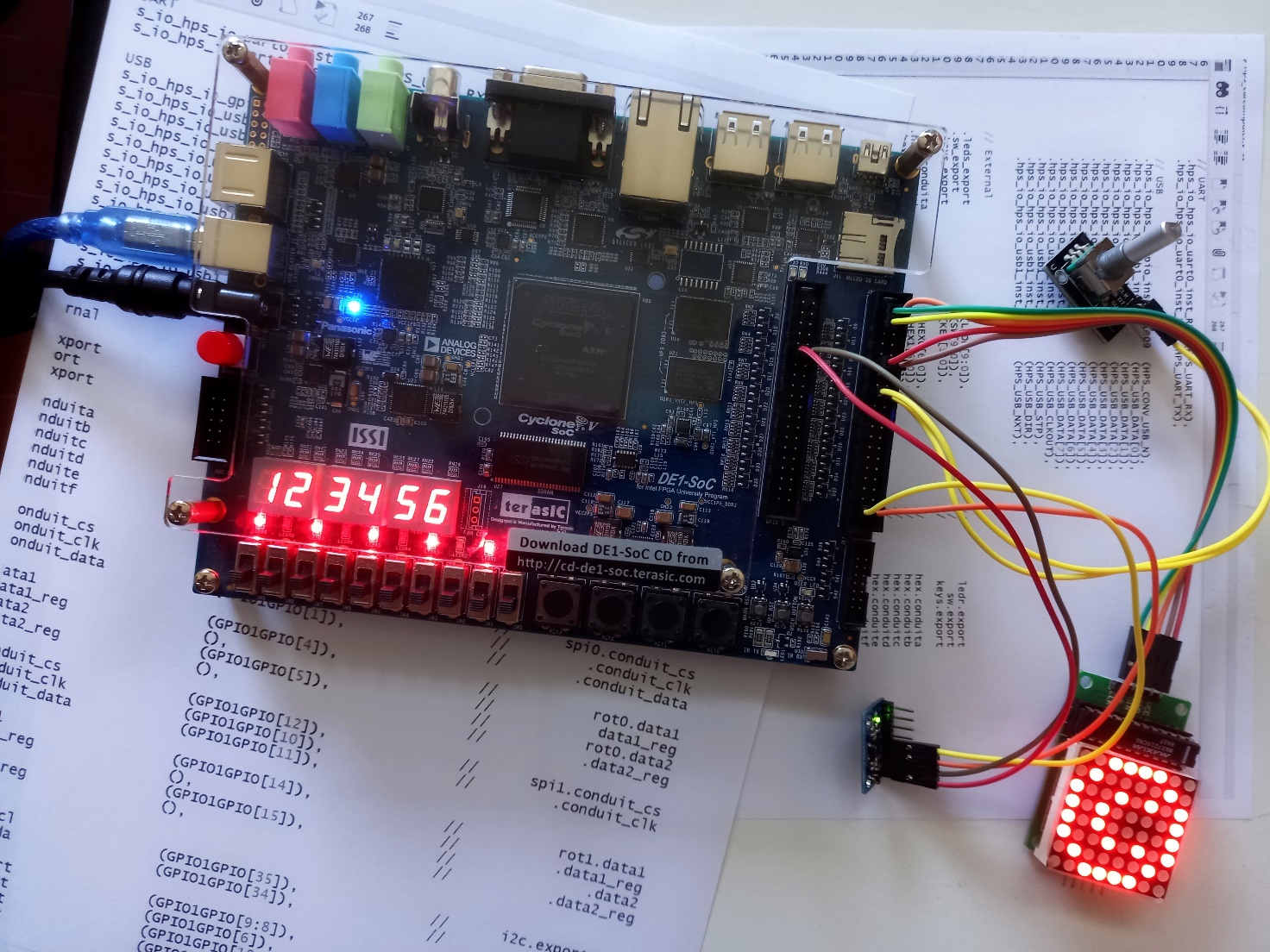
Progettazione di Sistemi Elettronici

# Prof. Marsi Stefano - Università di Trieste Anno Accademico 2023/24

Tutorial 5



**Sviluppo di un semplice processore con alcune periferiche basato su NIOS II.**  
**Hardware** impiegato: Terasic DE1-SoC Board  
**Software** Impiegato: Quartus 22.1

Tutorial 5

Realizzazione di un processore NIOS II, dotato di periferiche.

Descrizione: In questo tutorial si andrà a realizzare un semplice sistema composto da un processore NIOS2, una memoria ed alcune periferiche.

Scopo: Lo scopo è quello di vedere come si possa realizzare un intero sistema di calcolo all’interno di una singola FPGA e come questo sistema possa essere impiegato per interagire col mondo esterno attraverso periferiche già sviluppate da terze parti oppure anche sviluppando in proprio la periferica più idonea per realizzare una certa interfaccia verso l’esterno.

Apprendimento previsto:

* Apprendimento del tool di Quartus “Platform Designer” per realizzare un processore insieme al suo sistema di interfaccie
* Configurazione del processore e delle periferiche sviluppate da terze parti
* Realizzazione di periferiche “ad hoc”
* Programmazione del processore ed esecuzione del codice in un processo “Hardware-Software” Codesign

# Premessa

Le risorse interne di una FPGA sono sufficienti per realizzarvi un processore che potrà quindi successivamente essere interfacciato verso il mondo esterno e programmato per svolgere varie operazioni.

Questa pratica è talmente sviluppata che la maggior parte delle FPGA moderne contengono già al loro interno sia processori embedded realizzati in Hardware (Hard-Processors), sia alcune tra le più comuni interfacce (I2C, SPI, IrDA, UART, …) . La CycloneV non è da meno ed anche essa contiene già al suo interno un ARM biprocessore e svariate periferiche, ma per impratichirsi su come un processore possa essere realizzato, configurato, interfacciato e programmato in questo primo tutorial si verrà a realizzare quello che comunemente viene denominato un “Soft-Processor”, ovvero un processore realizzato utilizzando le risorse “Generla Purpose” interne all’FPGA quali blocchi logici e memorie. Le potenzialità del processore che verrà realizzato saranno estremamente limitate, ma lo scopo del Tutorial è quello di familizarizzare col tool di sviluppo. Per l’uso del processore Embedded (Hard-Processor) si rimanda il lettore ai successivi Tutorials.

La realizzazione di un intero processore partendo dai blocchi logici primitivi sarebbe un compito troppo vasto ed arduo per essere contenuto in un singolo tutorial, pertanto si farà uso del sistema sviluppato da Quartus (Platform Designer) che consente di definire e configurare una piattaforma di elaborazione con tutta la sua rete di interfacce in maniera piuttosto semplice.

# Realizzazione dell’ architettura

Si inizi realizzando lo “scheletro” di un sistema utilizzando il tool “System Builder” che preveda l’impiego di

* LEDs
* Display a 7 segmenti
* Swithes
* Tasti
* Clock
* GPIO1 (in modalità default)

Non tutte queste periferiche verranno impiegate all’inizio, ma la loro presenza sarà utile più tardi senza dover riconsiderare il progetto dall’inizio.

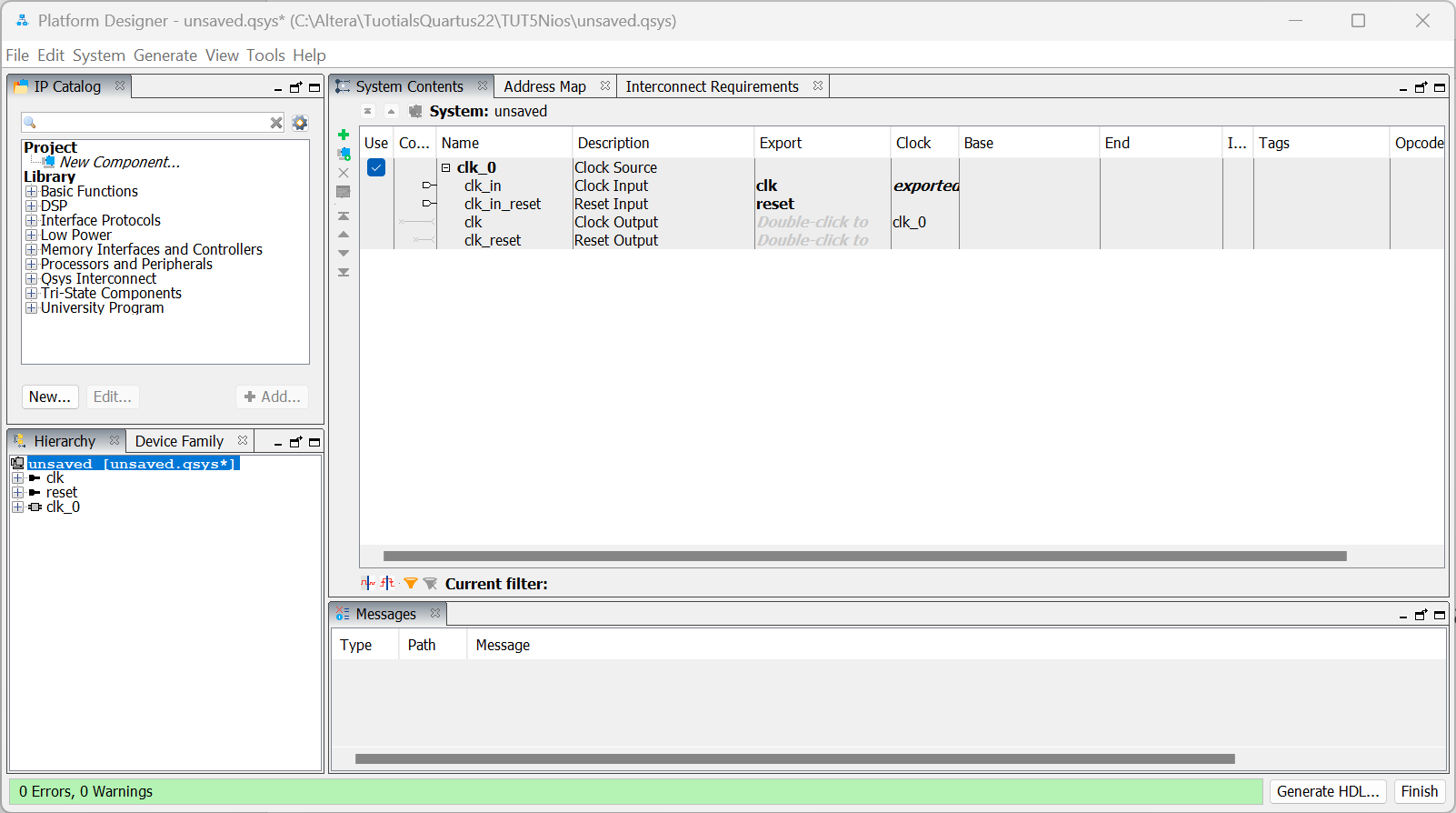
Si Apra il progetto così generato all’interno di Quartus.

All’interno di Quartus:

Tools > Platform Design

Si apre un sistema che consente di configurare l’architettura in base a blocchi già sviluppati da terze parti.

Il primo blocco è già presente ed è un blocco utile alla gestione del clock e del reset

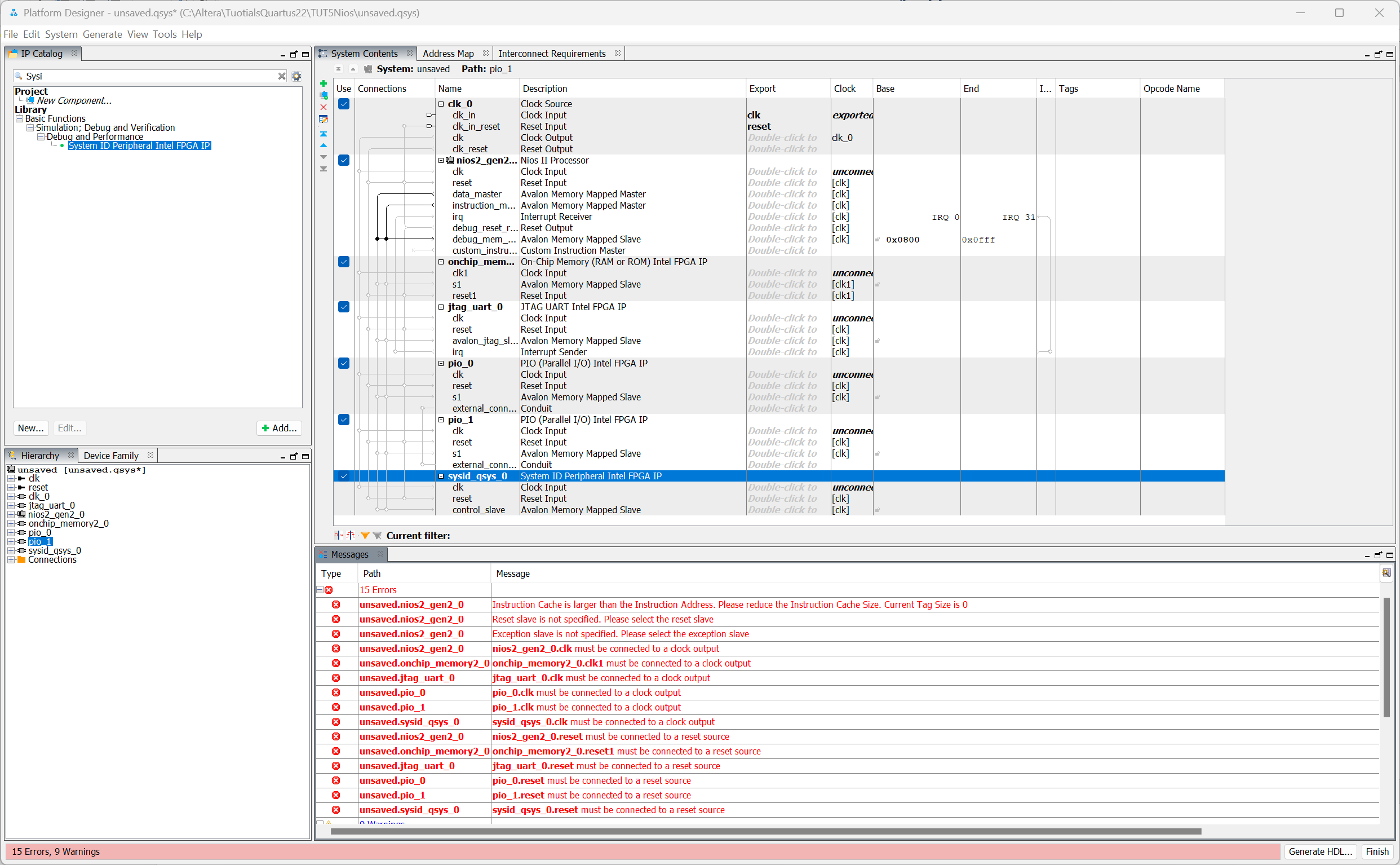


Nella Finestra di sinistra organizzati in sotto menù sono presenti diversi blocchi già sviluppati da terze parti.

Tra questi andremo a ricercare (magari sfruttando la funzione filtro) e ad aggiungere i seguenti Blocchi.  
Per ciascuno di essi si aprirà una finestra di configurazione, ma inizialmente per ciascuno di essi si accetti la configurazione di default attraverso il tasto “Finish”.

* **NIOS II Processor** (il soft processor)
* **On-chip Memory** **(RAM or ROM) Intel FPGA IP** (la memoria dove risiederanno dati e codice)
* **JTAG UART Intel FPGA IP** (L’interfaccia seriale STDIO per dialogare col processore)
* Due interfacce di tipo **PIO (Parallel I/O) Intel FPGA IP**
* **System ID Peripheral Intel FPGA IP** (identificatore del sistema – SysID)

La finestra con l’architettura si dovrebbe presentare più o meno così:



Dove in rosso sono indicati errori dell’architettura che andremo via via sistemando.

Le varie colonne nella scheda “System Contents” servono a definire:

1. I Collegamenti tra i blocchi
2. Il nome dei blocchi e dei segnali di I/O
3. La descrizione dei precedenti
4. I segnali che i blocchi esportano verso l’esterno dell’architettura (I/O)
5. Il dominio di clock
6. L’indirizzo di Base
7. L’indirizzo finale
8. Il livello di interupt

…

Cliccando sui vari blocchi attraverso il comando “CTRL-R” si può definire il nome del blocco più idoneo, mentre attraverso “CTRL-E” si possono definire le specifiche. Cliccando sui “puntini” bianchi o neri della varie connessioni esse possono essere attivate o disattivate.

Si rinomino opportunamente i vari blocchi e si configurino le connessioni come segue:

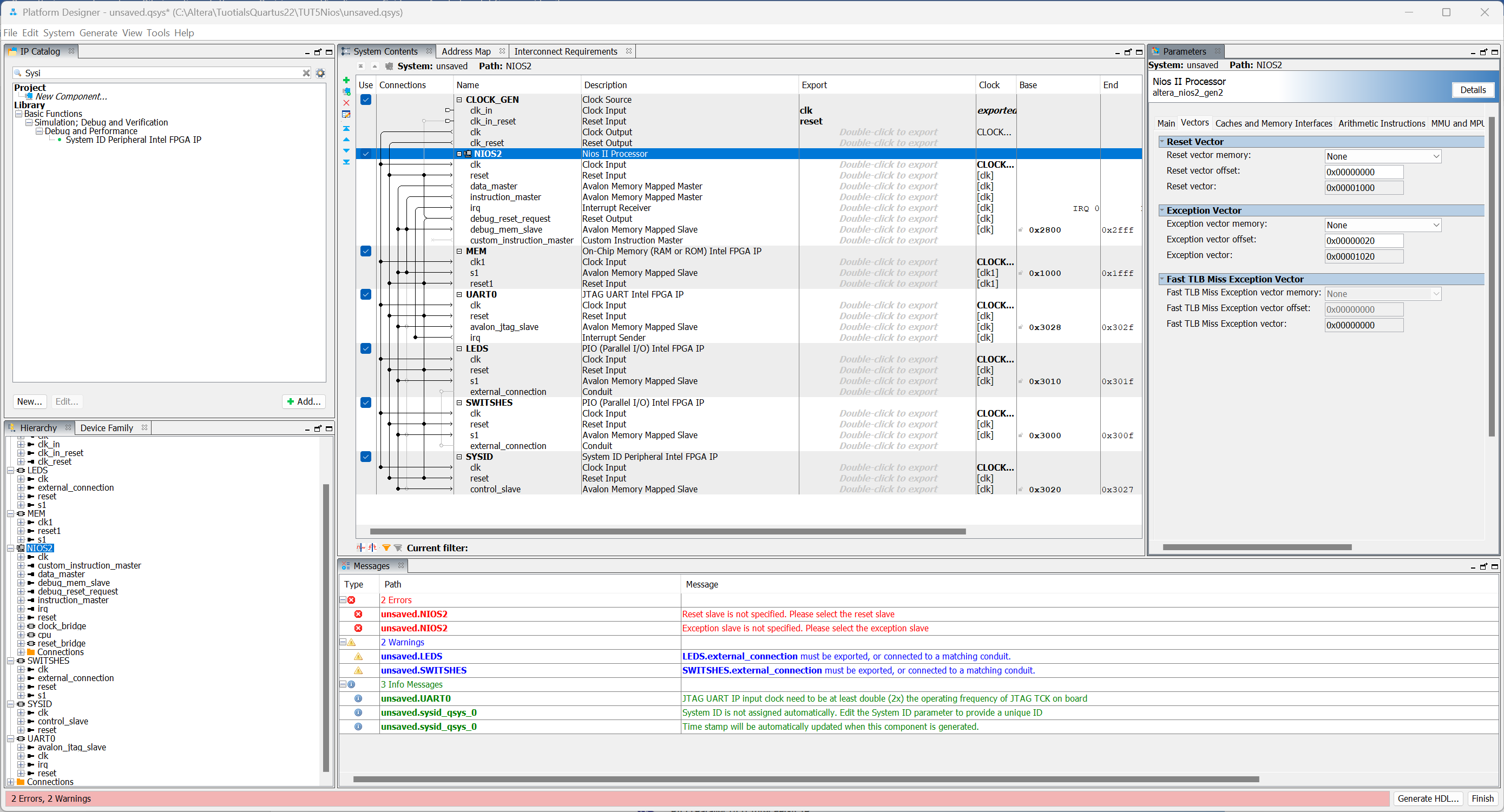
* Il segnale di clock in uscita dal “Clock generator” sia collegato a tutti i blocchi
* Il segnale di Reset in uscita dal clock generator sia collegato a tutti i blocchi

System > Create Global Reset Network

* La linea **data\_master** del NIOS2 raggiunga tutti i blocchi   
  (ovvero il nios legge/scrive dati da tutti i blocchi)
* la Linea **instruction\_master** raggiunga il blocco di memoria   
  (ovvero le istruzioni per in Nios risiedono solo in memoria)
* L’interrupt di **UART0** sia connesso al processore

Si vadano ora a definire opportuni indirizzi di memoria per i vari blocchi. Utile in tal senso può essere la procedura automatica

System > Assign Base Address

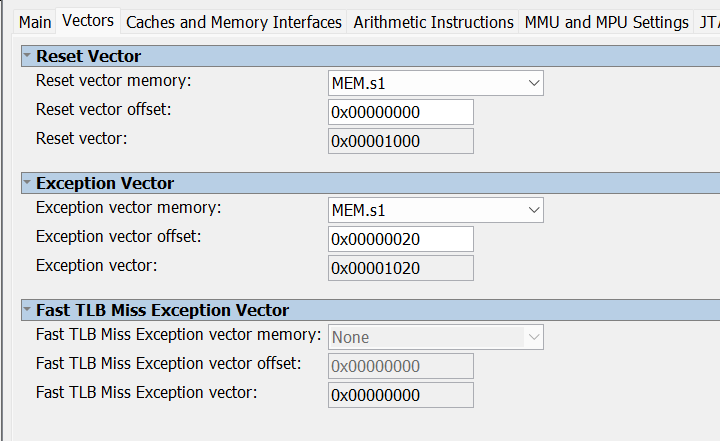


L’architettura potrebbe apparire più o meno così con ancora qualche errore.

Si vada quindi a configurare (CTRL-E) sui vari blocchi

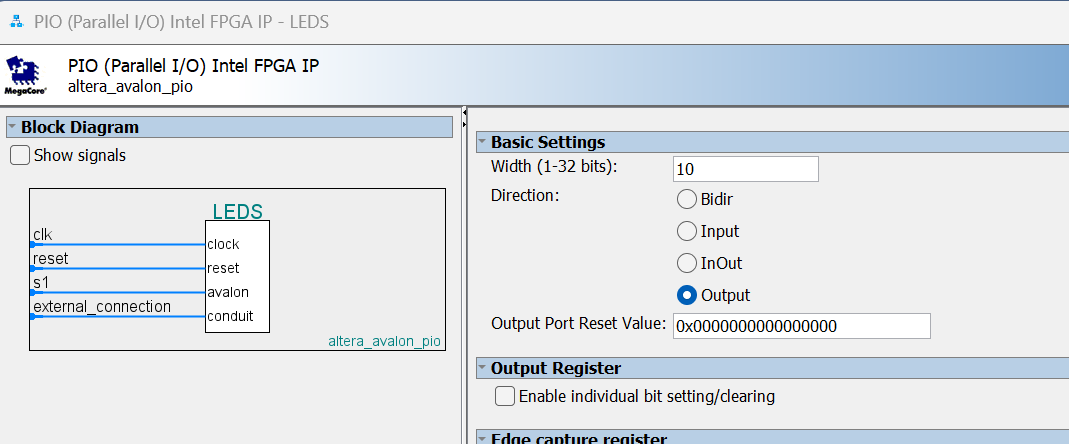
**Nios:**

* Tipo Nios II/e
* Reset Vector Memory deve far riferimento alla “On-chip Memory”
* Exception Vector Memory deve far riferimento alla “On-chip Memory”



LEDs

* Abbiano una lunghezza di 10 bit e siano configurati come uscita

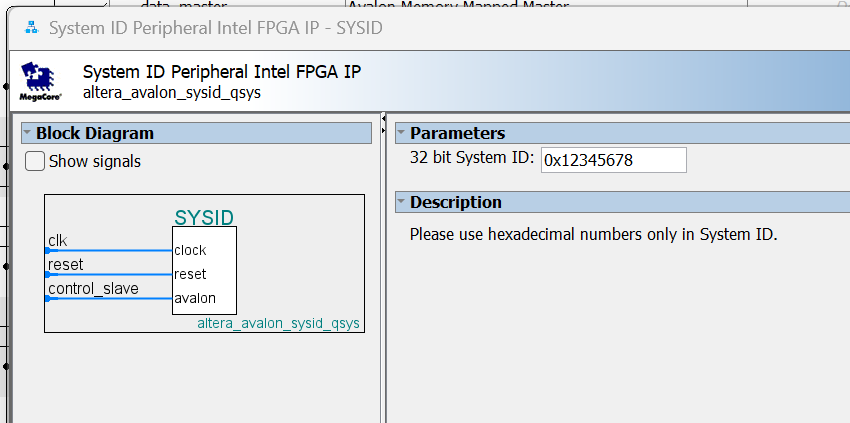


SWITCHES

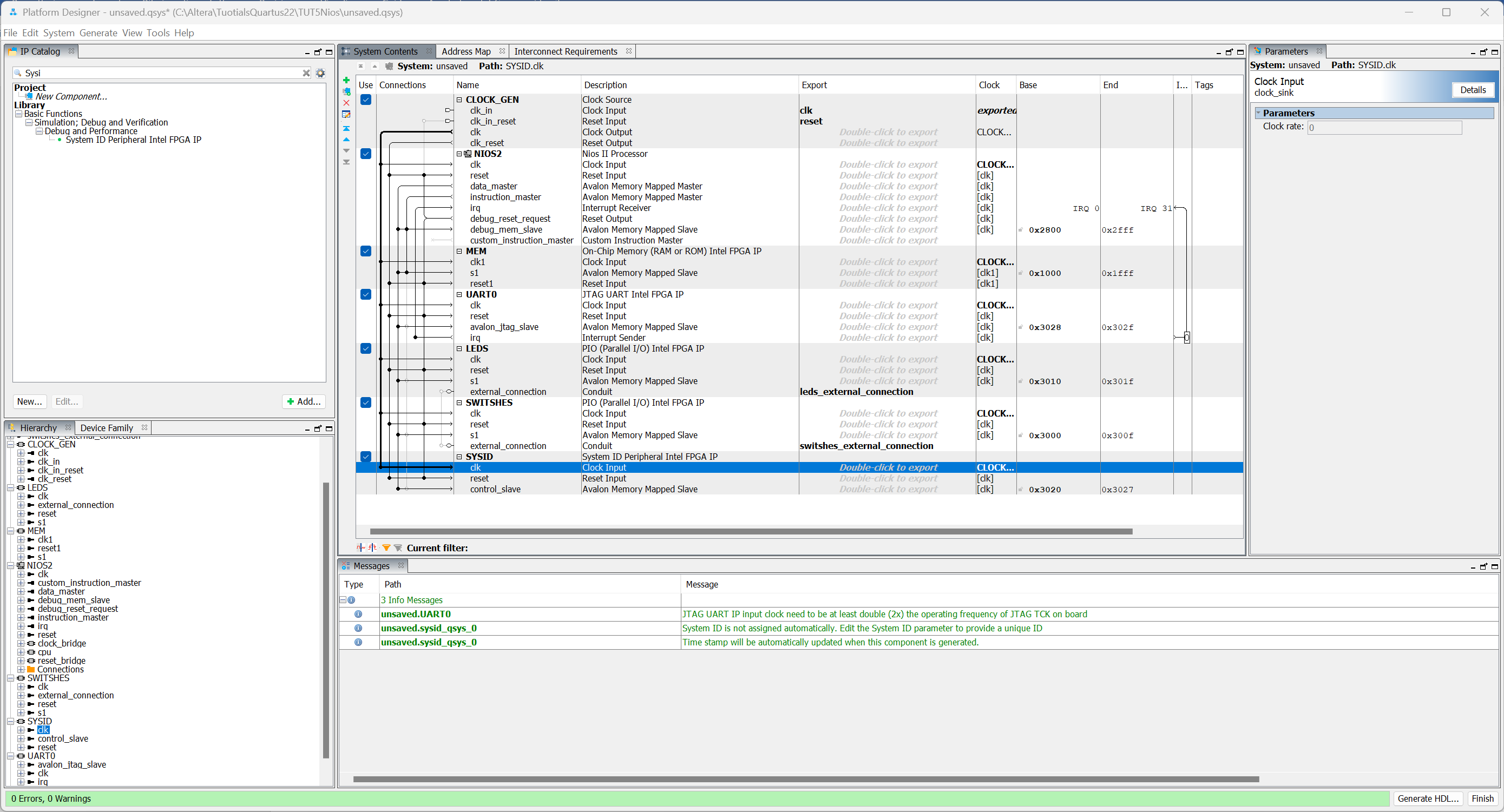
* Abbiano una lunghezza di 10 bit e siano configurati come ingresso

SYSID

* Fornire un valore identificativo univoco



Da ultimo, con un doppio click sul segnale “conduit” di “led” e “switches” si vada a definire il nome con cui tali blocchi vanno ad interfacciarsi verso l’esterno dell’architettura.



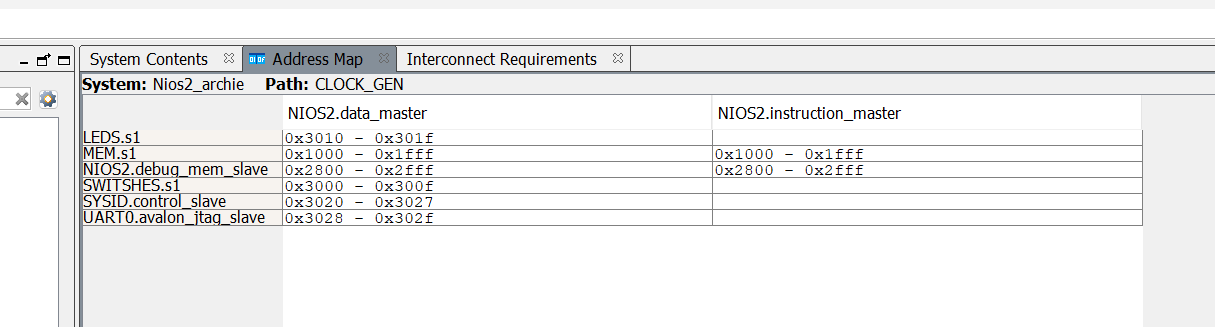
L’architettura è completamente definita si salvi con un nome opportuno (ad esempio “Nios2\_archie”)

è può essere generato il suo rispettivo codice HDL, usando in questo caso la scelta “Verilog HDL”

Quando la generazione è ultimata

< Close >

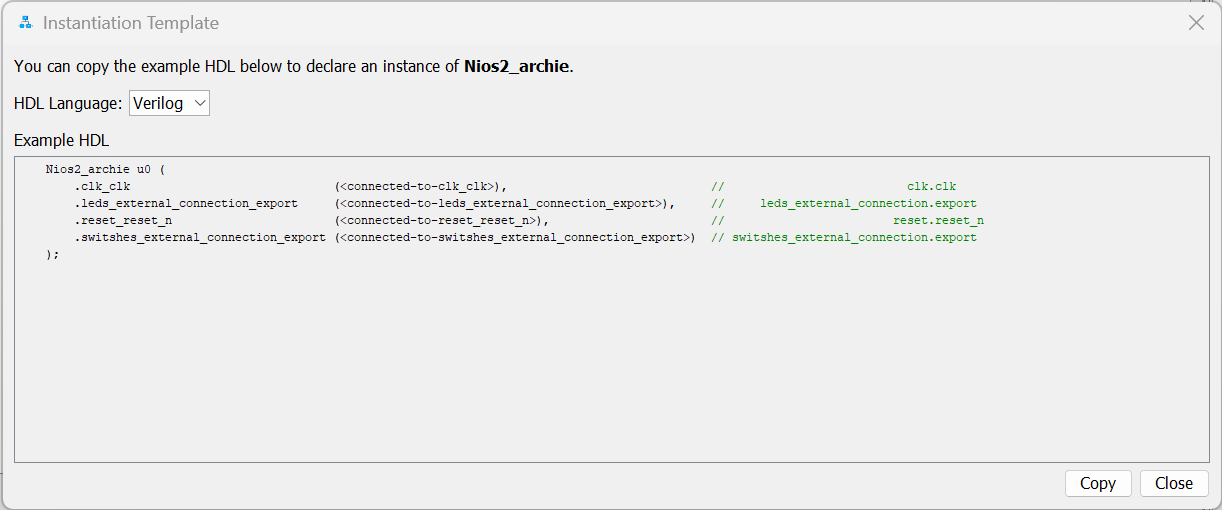
Prima di chiudere il procedimento prendere nota delle locazioni di memoria delle varie periferiche



E potrebbe essere utile vedere un esempio di come instanziare il componente:

Generate > Show Instanstation Template

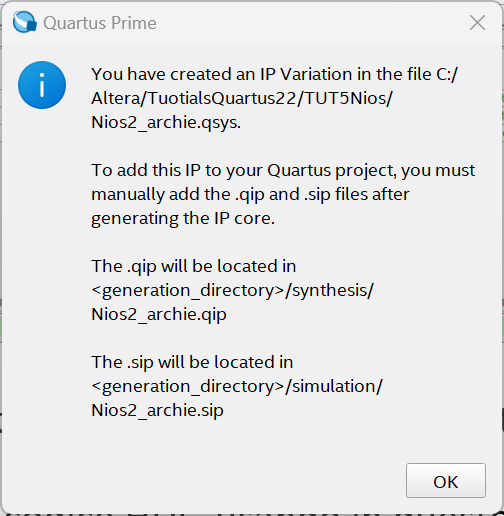
Copiare il codice



< Copy >

< Finish >

Probabilmente comparirà una finestra simile che ricorda dove sono stati archiviati i files da includere manualmente al progetto.

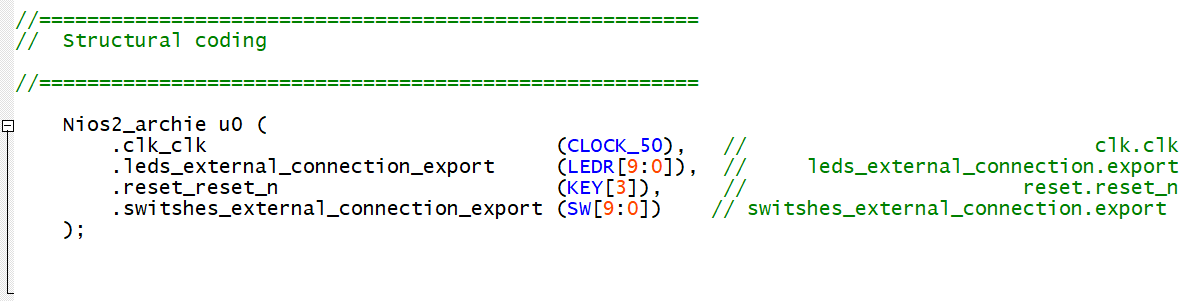


# Istanziazione dell’ architettura all’interno del progetto

L’architettura generata al passo precedente deve essere adesso importata a livello di sistema globale ed interfacciata con eventuali altri blocchi e segnali.

Ad esempio bisogna far si che il clock ed un opportuno segnale di reset arrivi all’architettura ed i suoi segnali di I/O relativi a led e switshs si colleghino ai corrispettivi piedini all’interno della scheda DE0-SoC

A livello della “Top Level Entity” generata col tool “system builder” all’inizio del tutorial, prendendo spunto dal modello di istanziazione appena generato si può integrare il codice con:



Mentre a livello di sistema bisogna ancora includere il file “.qip” generato nel progetto.

Project > Add/Remove files in project

Cliccare sui tre puntini

Andare a trovare il file laddove ci era stato indicato precedentemente dalla finestra di dialogo ed importarlo.

<OK>

A Questo punto l’intero sistema può essere compilato e caricato sulla scheda DE1-SoC.

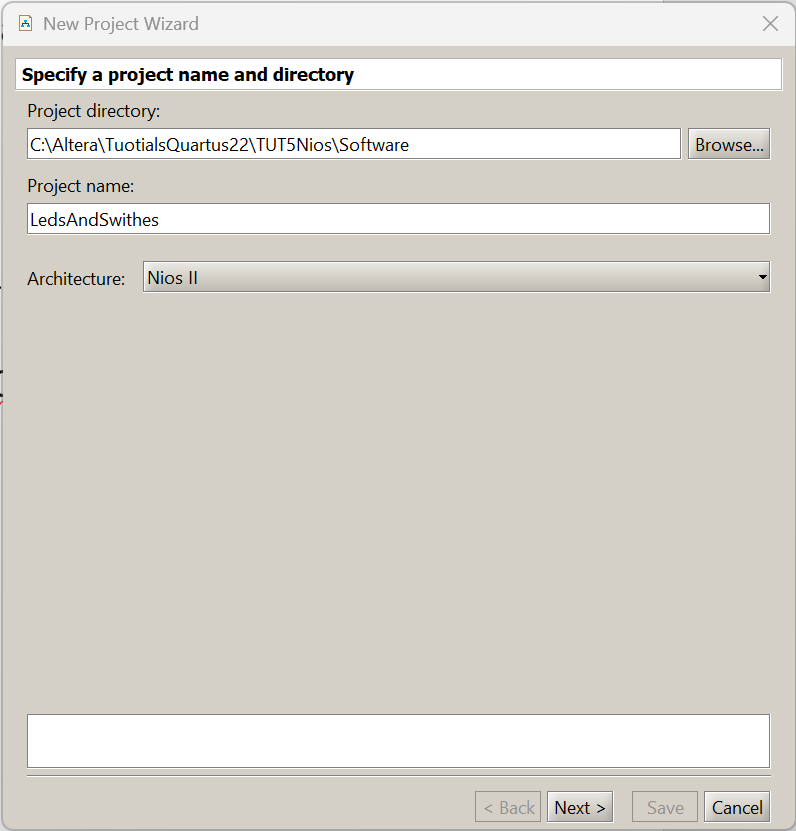
La Parte Hardware inerente alla realizzazione del sistema è finita e si può chidere Quartus.

# Intel FPGA Monitor Program

Lanciare “Intel FPGA Monitor Program”

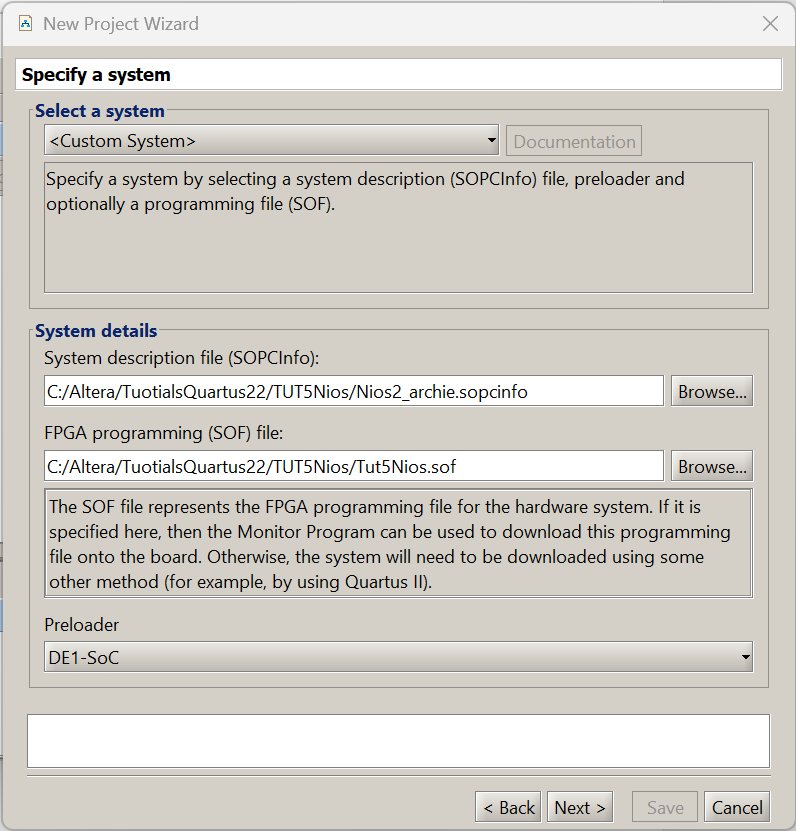
File > New Project

Definire una directory opportuna, un nome per il progetto e scegliere quale processore NiosII



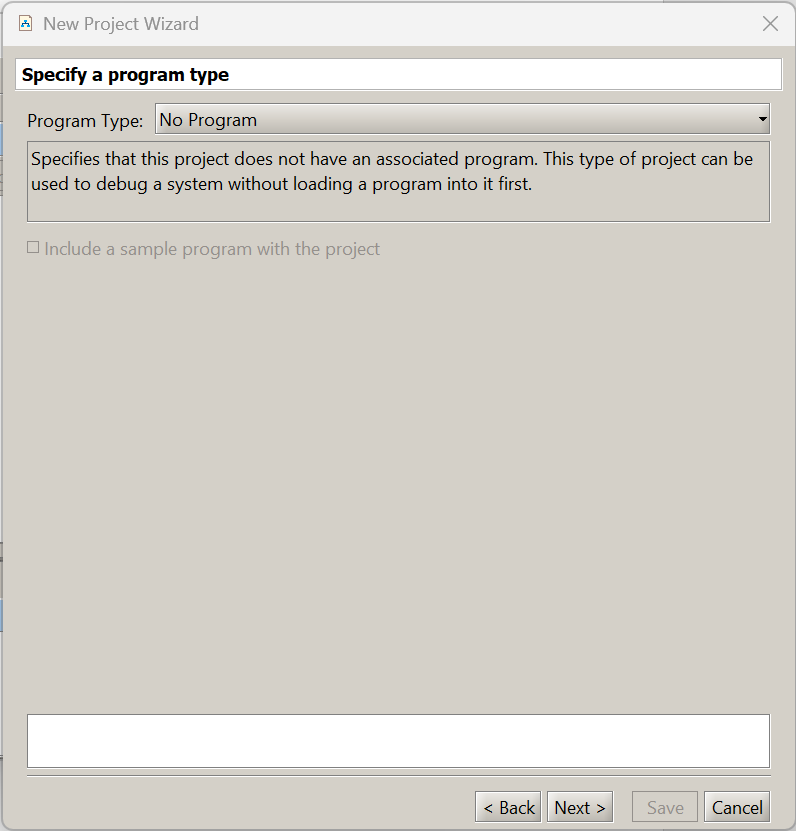
> Next

Come tipologia di architettura scegliere <Custom>, indicare poi i file che caratterizzano il sistema hardware .sopcinfo e .sof ubicati nel direttorio dove risiede l’hardware del progetto come pre-loader quello relativo alla scheda DE1-Soc



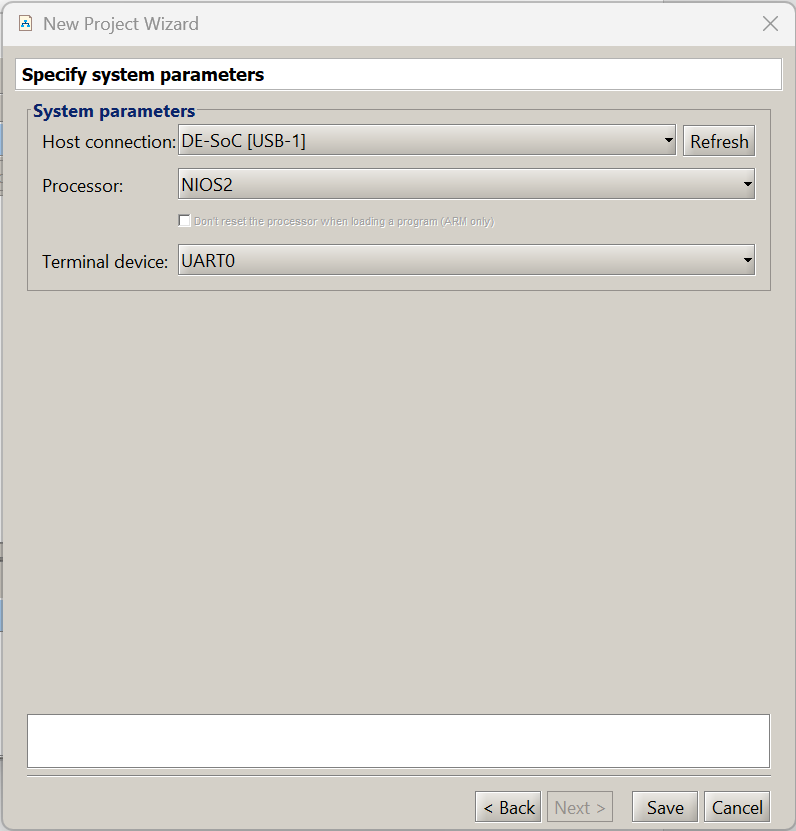
> Next >

Come tipologia di programmazione al momento scegliere <No Program>



> Next >

Definire quindi Processore ed interfaccia di comunicazione



> Save

Il sistema chiederà se si vuole caricare il file di configurazione sulla scheda, ma presentando un piccolo bago del sistema è meglio soprassedere su questo punto ed impiegare Quartus per programmare la scheda (come gia fatto).

Action > Connect to System

Accedere alla scheda Memory

Goto Address andare agli indirizzi in cui sono stati mappati LEDs, Swithes e SysID (ad es. 0x3000), mettere la spunta su “Query Device”

> GO

A questo punto si possono vedere le varie periferiche attraverso i loro indirizzi di memoria:

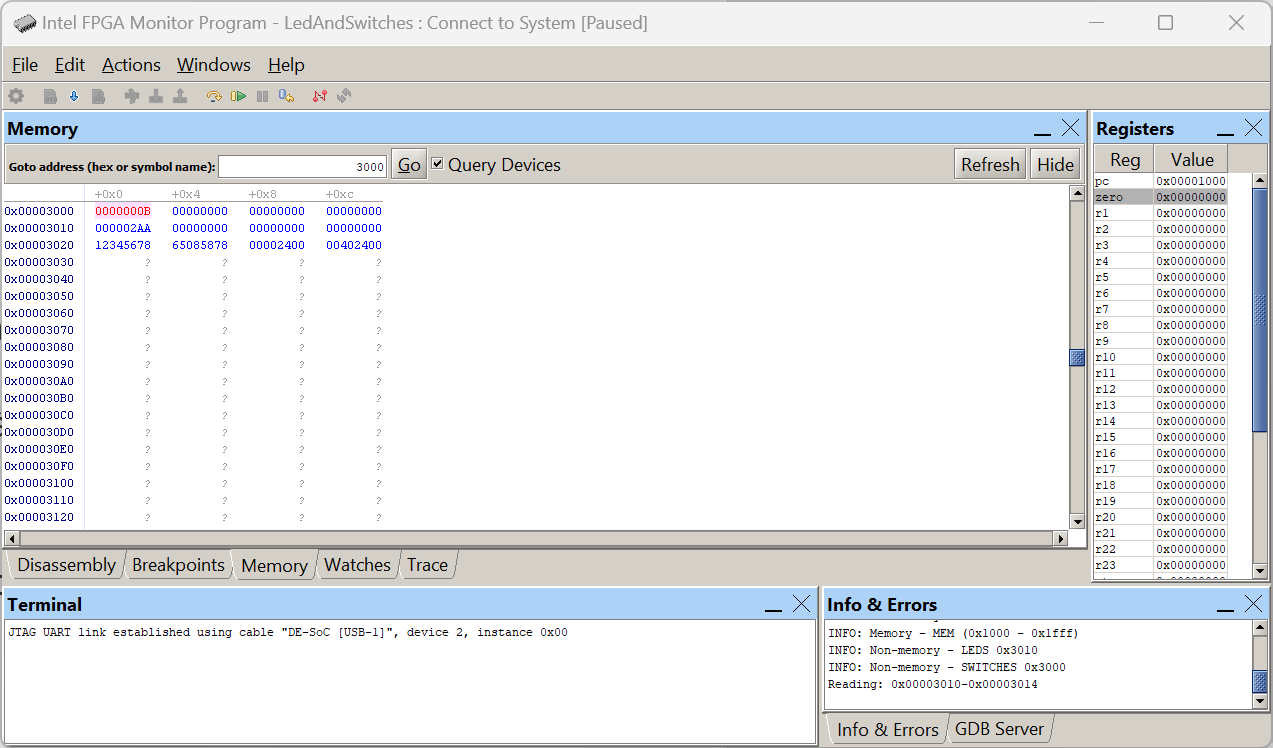
- scrivendo un qualche dato (es 0x2AA alla locazione relativa dei LED) si vedranno i led accendersi alternati)

- attivando una configurazione sugli switches e cliccando <Refresh> si vedrà tale configurazione mappata in memoria

- alla locazione del SysID si vedrà mappato il valore fissato via Hardware in fase di definizione dell’architettura

- Pigiando il tasto preposto al Reset (Key[3]) i led si spegneranno

In rosso vengono riportati i valori modificati dalla precedente lettura.

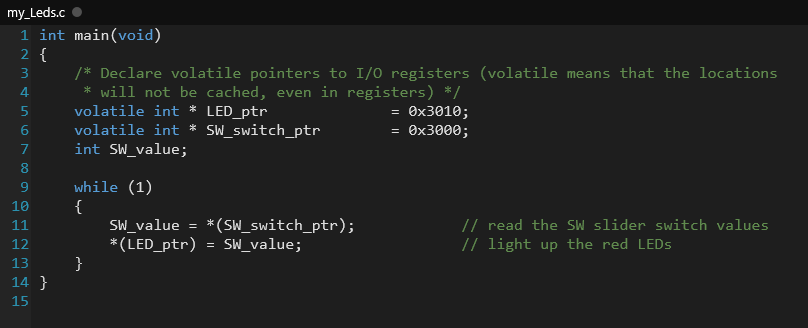


Finora le operazioni di lettura e scrittura verso le periferiche sono state fatte manualmente attraverso l’interfaccia JTAG ed il processore non vi ha preso parte.

# Software

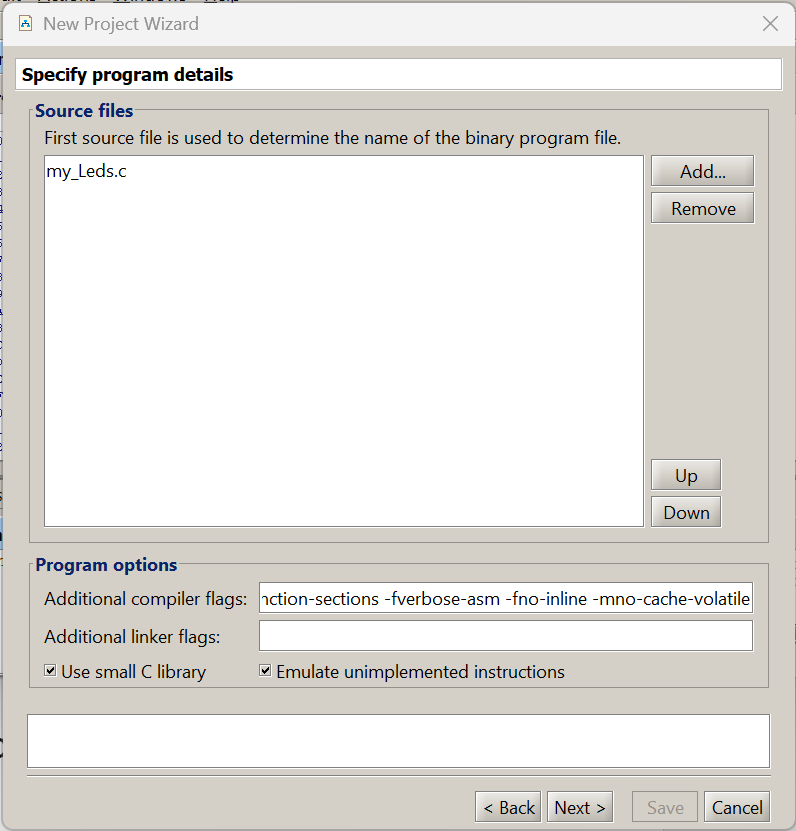
Si voglia adesso scrivere un programma in C che eseguendo un loop infinito legga i dati dagli switches e li copi sui LED.

Un codice opportuno potrebbe essere il seguente



Ovviamente modificando le locazioni dei puntatori opportunamente

Si crei un nuovo progetto all’interno di Intel FPGA Monitor Program, ma questa volta alla scheda inerente la tipologia di programmazione scegliere “C program” ed alla scheda successiva importare il programma precedentemente scritto attraverso un text editor e selezionare “Use Small C Libraries”



Le altre schede che caratterizzano la configurazione software del progetto possono rimanere immutate.

Ultimata la configurazione

Action > Compile & Load (F5)

Il software viene compilato e caricato nella memoria del processore

Action > Continue (F3)

Fa partire il programma che continuamente legge i dati dagli swithes e li copia sui leds.

Se si pigia il tasto di Reset (KEY[3]) sulla scheda il ciclo si ferma ed i led si spengono

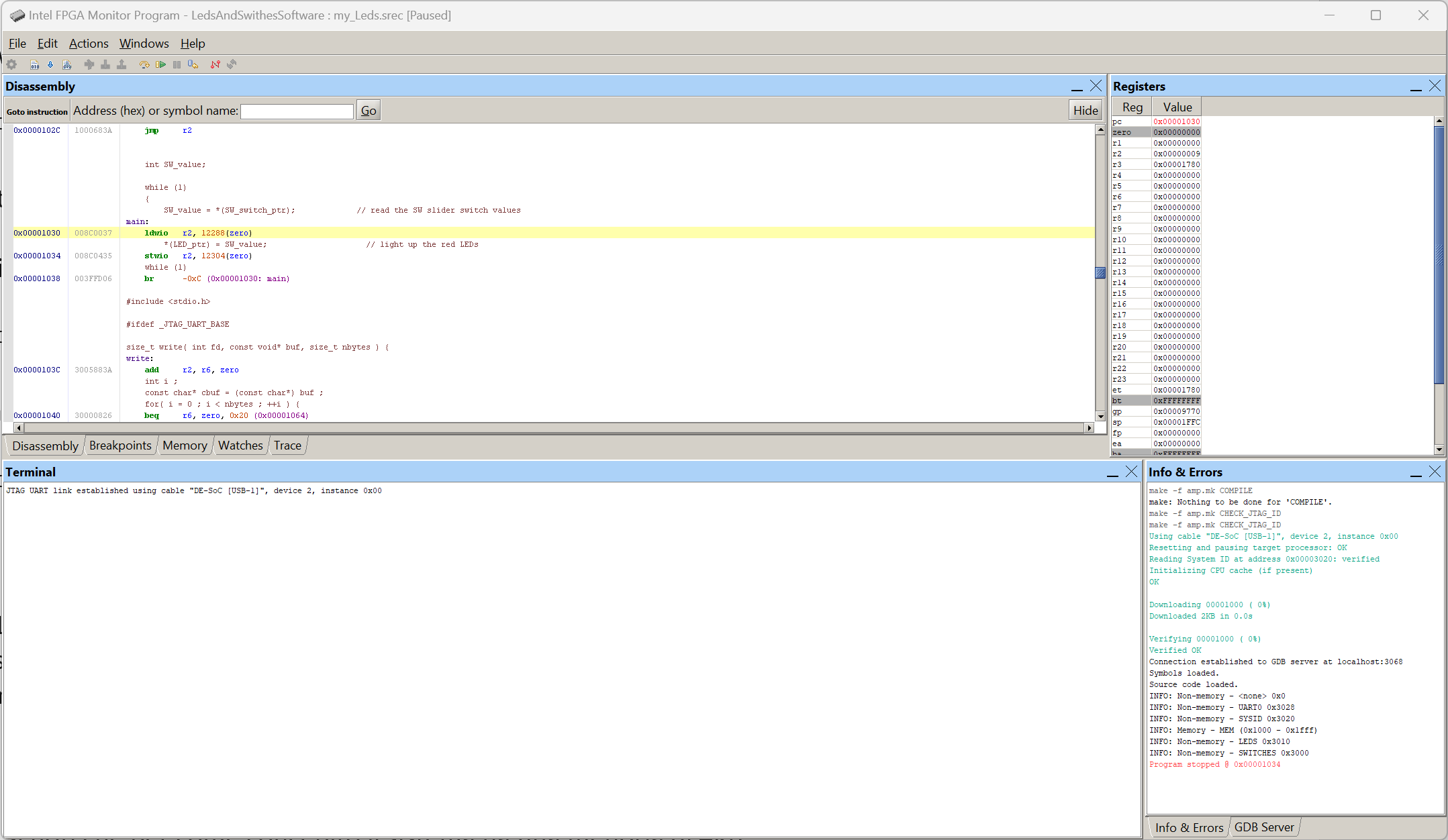
Action > Continue (F3)

Fa ripartire il sistema

Action > Stop (F4)

Ferma il processore che non esegue più il ciclo prescritto ma NON resetta i leds.

**NOTA1**: Interessante è l’uso della finestra “Disassembly” che propone il codice eseguito dal processore in forma “Assembly” e che può essere eseguito anche uno step alla volta visualizzando la posizione all’interno del codice nonché il contenuto di tutti i registri.



**Suggerimento**: si modifichi la definizione dei puntatori da “volatile int” ad “int”, cosa cambia a livello si esecuzione ? ed a livello di codice Assembly ?

**NOTA2**: si noti che per come concepito il sistema esso ha una memoria di 4Kb, estremamente risicata per qualsiasi programma, se si provasse a scrivere un codice anche solo di poco più complesso del presente oppure che richiedesse l’utilizzo di libreie il sistema NON sarebbe in grado di supportarlo. Si potrebbe aumentare la dimensione della ON-Chip Memory, ma i blocchi messi a disposizione internamente all’FPGA sono comunque piuttosto limitati, l’unica soluzione è pertanto quella di interfacciarsi a memorie esterne l’FPGA stessa sfruttando opportune interfacce.

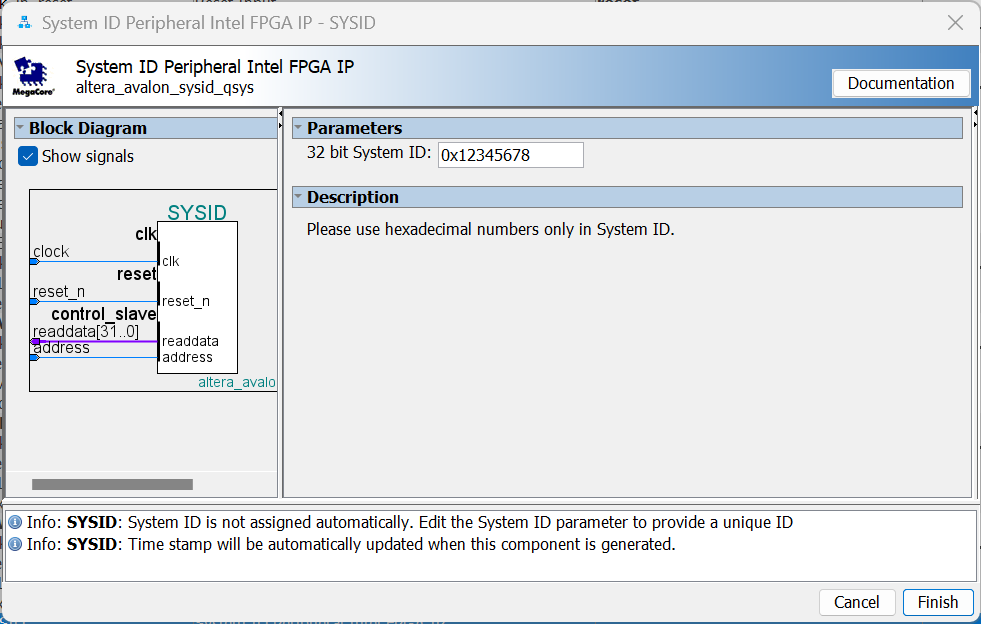
# Analisi delle interfacce utilizzate finora

Tutti i blocchi finora utilizzati sono stati sviluppati da terze parti e semplicemente inclusi nell’architettura nella forma in cui sono stati proposti. Per comprendere meglio come il processore possa agire in lettura e scrittura su questi moduli una strategia utile è quella di analizzarne il loro codice:

Si riapra il progetto Quartus chiuso in precedenza e si analizzino alcuni dei blocchi semplici che sono stati importati attraverso il tool “platform design”:

**SysID:**

All’interno del tool “platform design” attivando sulla scheda “Block diagram” l’opzione “show signals” si ha una visualizzazione come segue



Il che indica come l’accesso da parte del processore alla risorsa attraverso il bus “avalon” avviene con pochi semplici segnali:

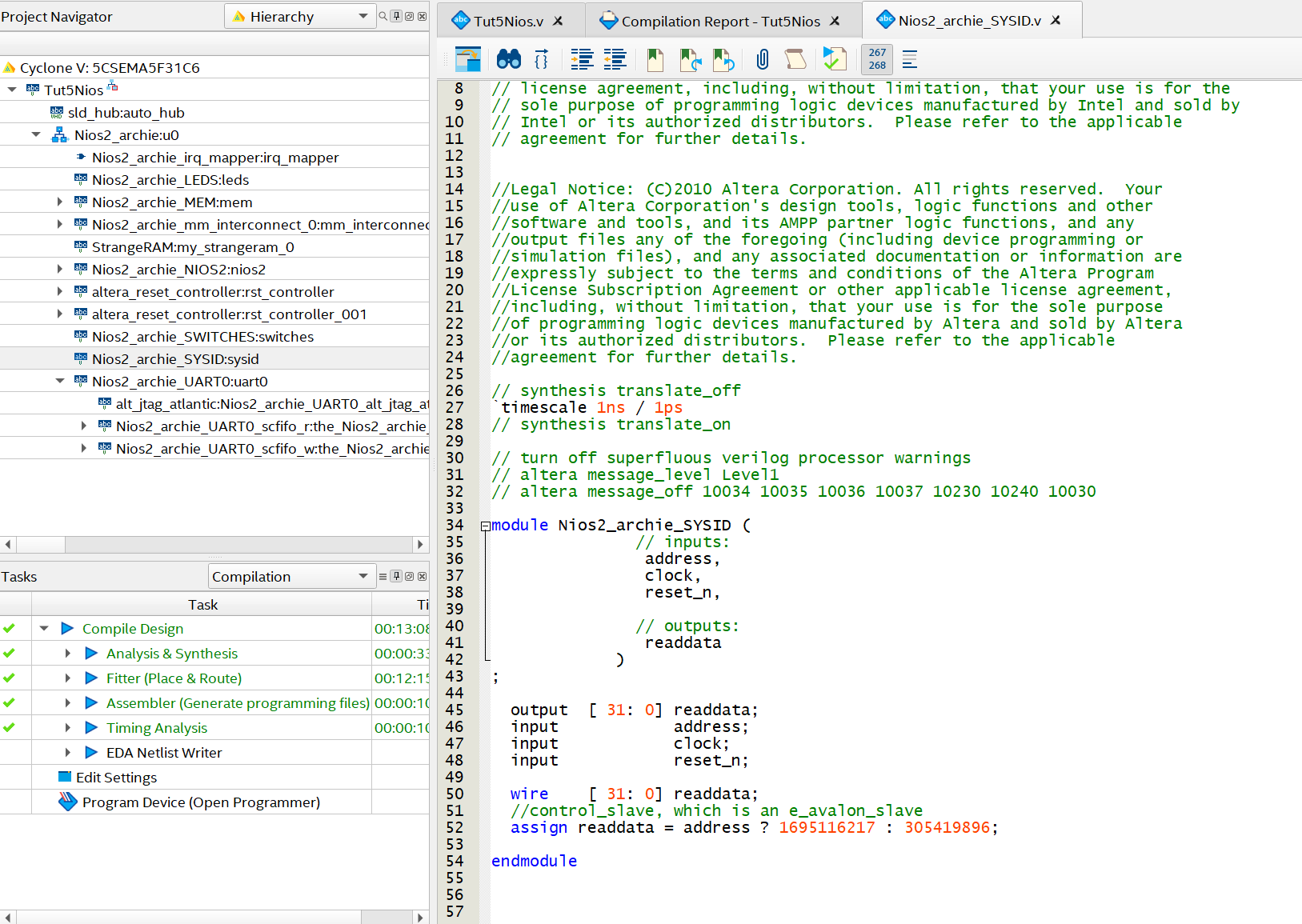
- un clock di sistema (doveroso in tutti i sistemi, ma qui come vedremo non utilizzato)

- un reset di sistema (doveroso in tutti i sistemi, ma qui come vedremo non utilizzato)

- un'unica linea di indirizzo (essendo l’elemento composto in pratica da una sola cella di memoria)

- un bus dati in lettura a 32 bit (solo in lettura perché di fatto non vi è nessuna necessità di scrivere)

Per analizzare il codice ad esso legato si apra invece all’interno del “Project Navigator” di Quartus, messo in modalità “Hierarchy” attraverso un doppio click l’elemento: Nios2\_archie\_SYSID

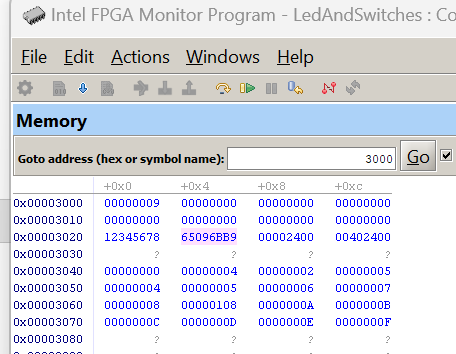


Nell’architettura in VerilogHDL si troveranno gli stessi segnali di I/0 già analizzati ed un’unica assegnazione combinatoria alla riga 52:

assign readdata = address ? 1695116217 : 305419896;

Che potrebbe essere riassunta nella seguente forma verbosa: se “address==1” il dato sarà 1695116217=0x65096BB9 altrimenti se “address==0” 305419896=0x12345678

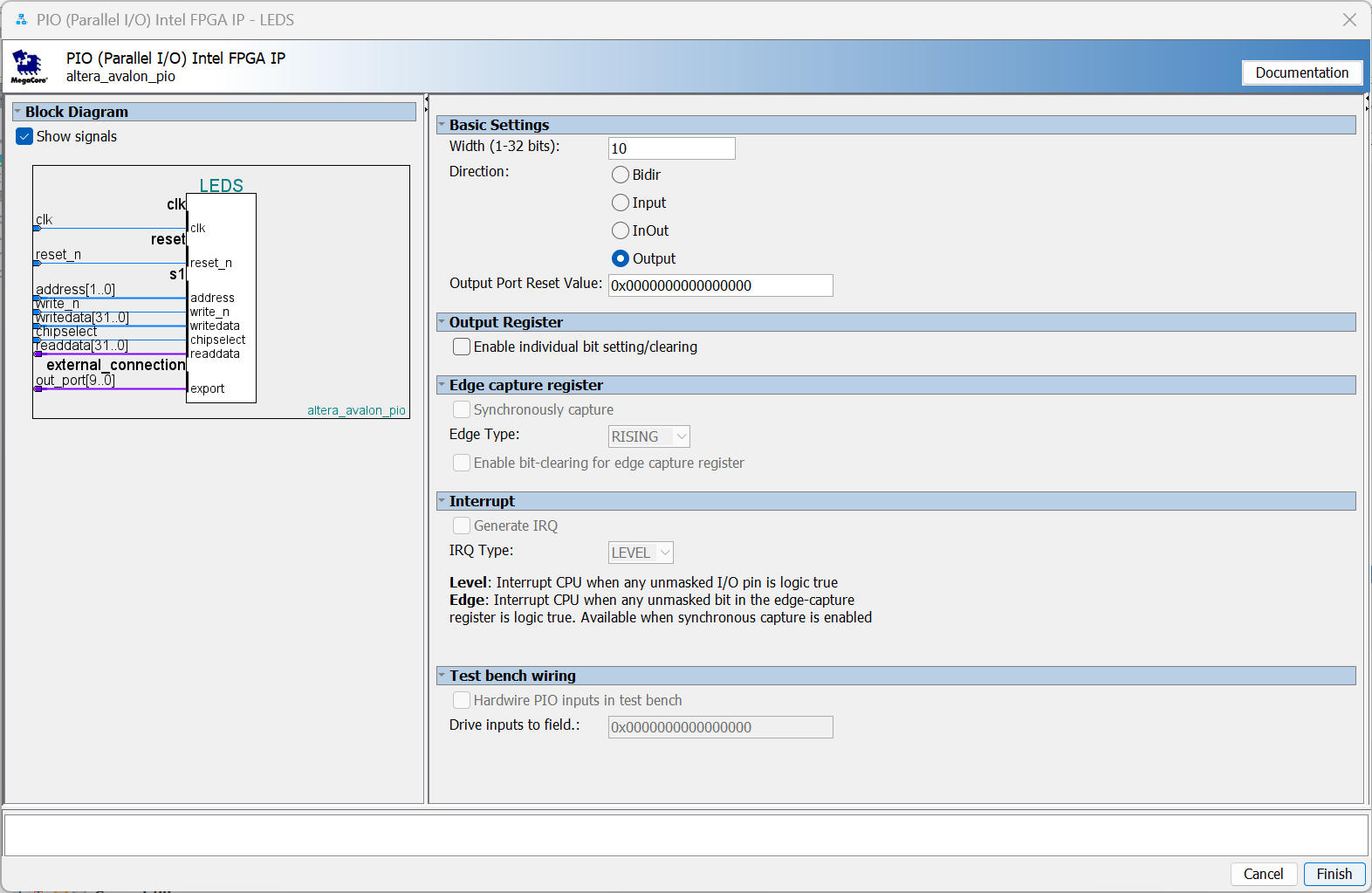
Ovvero quando si effettua una lettura della cella all’indirizzo 0 si troverà 0x12345678 mentre all’indirizzo 1 0x65096BB9.



Si noti che sebbene la periferica poi si troverà “mappata” alla locazione 0x3020 gli indirizzi di cui sopra sono da intendersi come riferimenti “relativi” ai quali poi verrà aggiunto poi, in fase di sintesi del circuito la logica atta a traslarlo alla posizione corretta.

**LEDS:**

Con analoga procedura relativa al blocco di interfaccia “Nios2\_archie\_LEDS”

****

Si nota innanzitutto la presenza di più segnali:

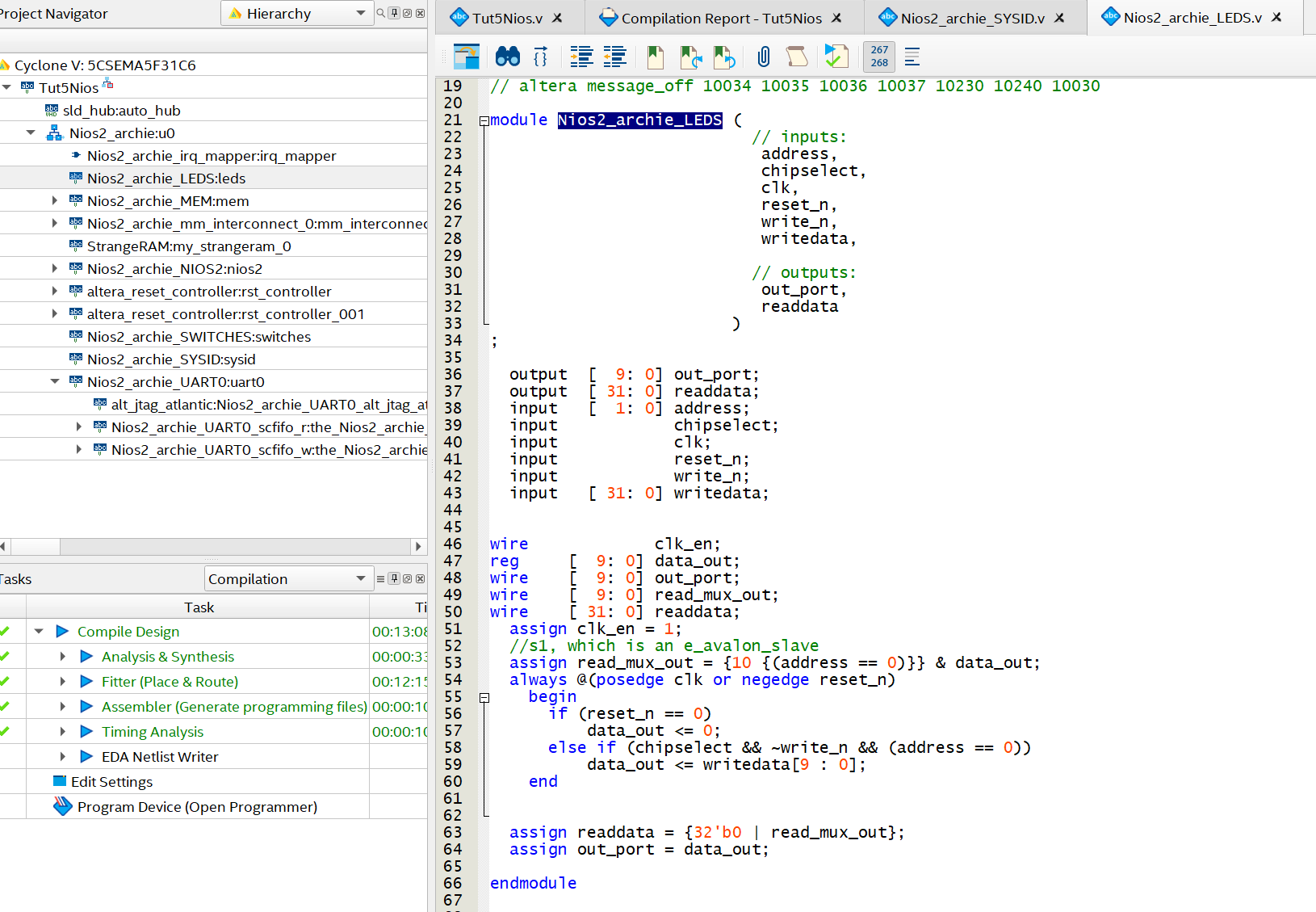
- clock e reset (sempre utili)  
- address a 2 bit (per 4 posizioni)  
- segnale write\_n per attivare, quando LOW la scrittura e quando HIGH la scrittura  
- writedata (in ingresso) e readdata (in uscita), entrambi a 32 bits  
- chipselect ad indicare che la comunicazione è attiva per l’interfaccia in questione.  
- out\_port in uscita che andrà a pilotare i 10 led esterni

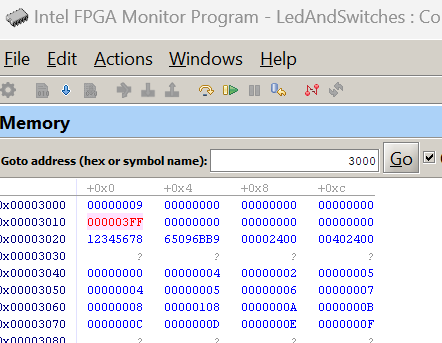
Ed analizzando il codice VerilogHDL corrispondente si può notare la possibilità di 3 operazioni principali:

- alle righe 58/59 : Quando chipselect è attivo, write\_n è basso (scrittura) e l’indirizzo è quello della prima posizione di memoria (address==0) nel registro data\_out vengono caricati i 10 bit meno significativi del dato in ingresso writedata.

- al rigo 64 i 10 valori presenti nel registro data\_out vengono portati nella porta d’uscita (collegata ai leds) out\_port

- alle righe 63( e 53) i dati letti dalla cella readdata (a 32 bits) sono composti da 32 bit posti di cui i primi posti sempre a 0 e gli ultimi 10 che assumono il valore del registro data\_out. Infattise nella cella di memoria corrispondente ai LED si provasse a scrivere un valore che avesse più di 10 bit significativi, in fase di ri-lettura della cella i primi 22 bit sarebbero azzerati





**SWITCHES**

Analoga analisi si consiglia di svolgere anche per l’interfaccia di I/O relativa agli switches.

# Generazione ed importazione di interfacce “custom”

Si provi adesso a generare ex-novo una nuova interfaccia per il processore che condivida con esso la comunicazione attraverso il bus AvalonMM. Si voglia realizzare, a scopo puramente didattico una sorta di memoria di memoria composta da diverse celle a 32 bit accessibili in scrittura ed in lettura, dove però in fase di lettura restituisca la somma del contenuto della cella e l’indirizzo (relativo) della stessa.

Il Codice proposto è il seguente:

module StrangeRAM (clk, addr, rdata, wdata, read, write, reset\_n);

parameter ADDRSIZE = 1;

parameter WORDSIZE = 32;

input clk;

input [ADDRSIZE-1:0] addr;

output [WORDSIZE-1:0] rdata;

input [WORDSIZE-1:0] wdata;

input read,write;

input reset\_n;

reg [WORDSIZE-1:0] Mem [0:(1<<ADDRSIZE)-1];

assign rdata = (read) ? (Mem[addr]+addr) : {WORDSIZE{1'bz}};

always @(posedge clk)

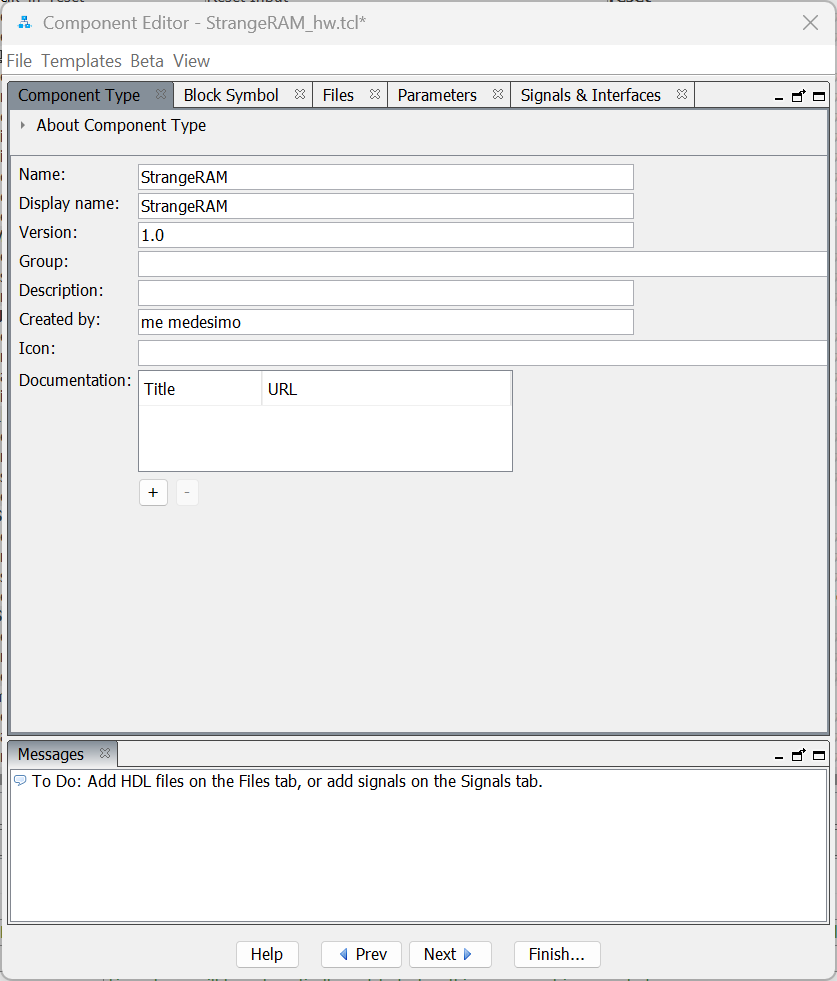
if (write)

Mem[addr] = wdata;

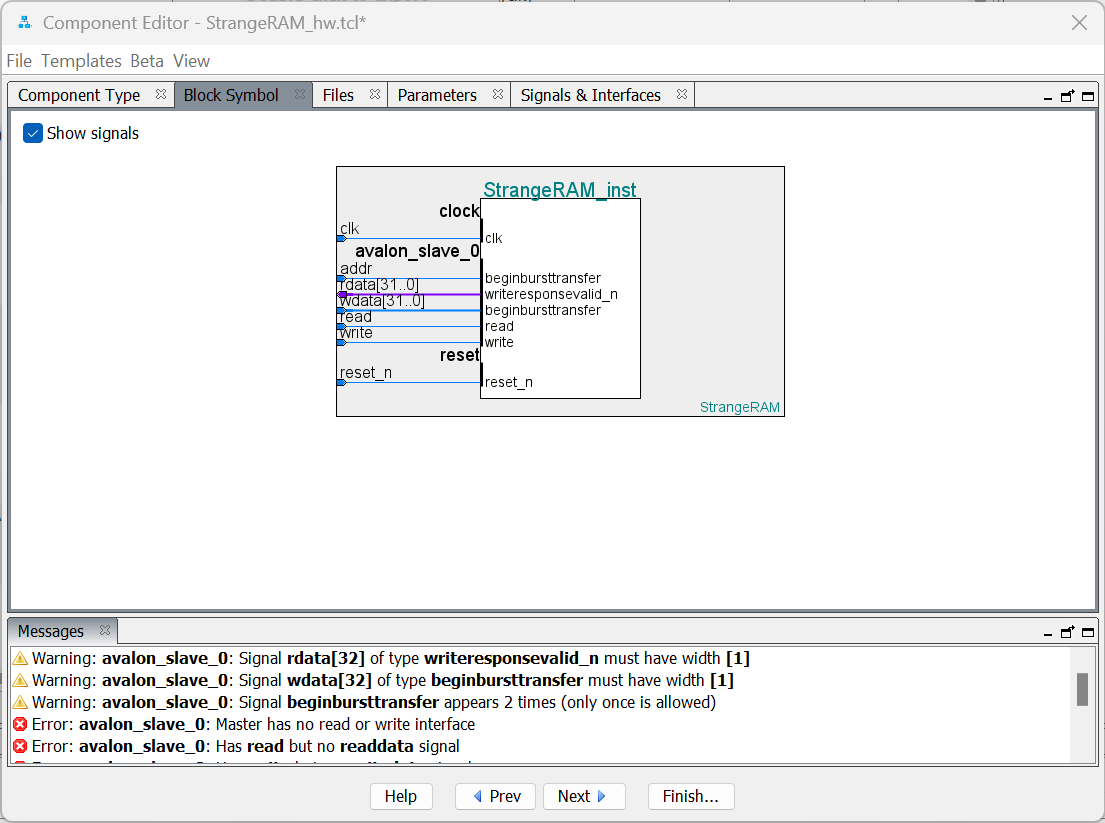
endmodule

Si ricopi il codice attraverso un text editor in un opportuno file \*.v ed eventualmente si simuli attraverso Questa/Modelsim

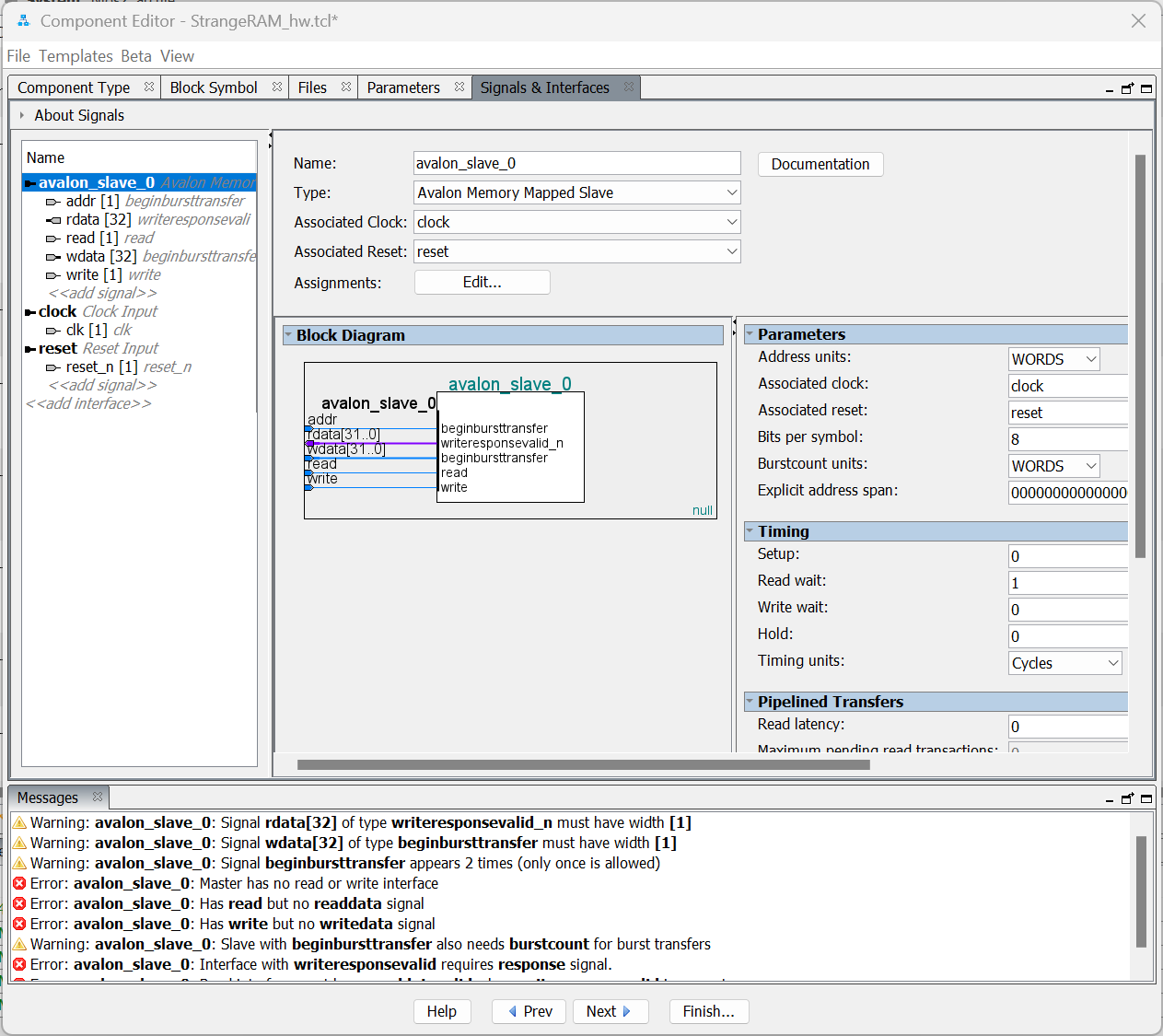
Per importare l’elemento all’interno di “platform designer” si effettui un doppio click su *“New Component”*



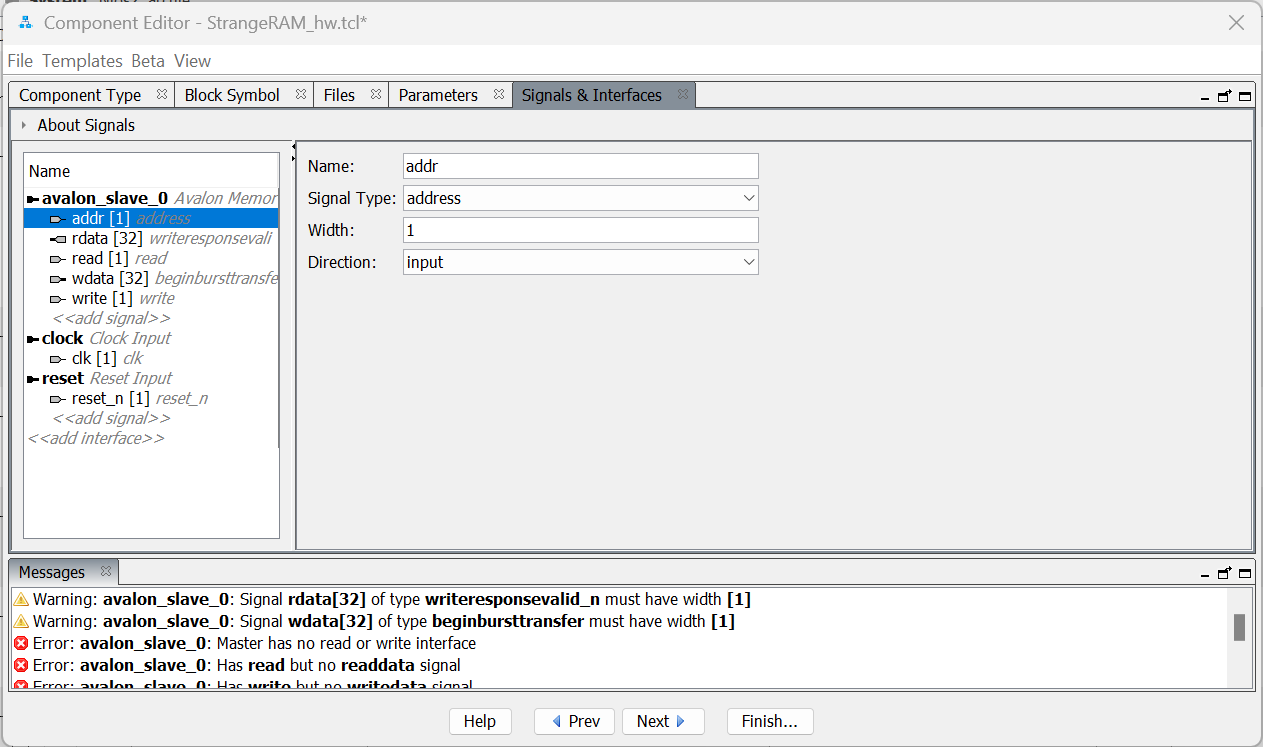
* Nella scheda “Component Type” si fornisca un Nome al componente e si riempiano opportunamente i vari campi (opzionale).
* Nella scheda Files cliccando sul pulsante “Add File…” si importi il file precedentemente salvato.
* Successivamente “Analyze Syntesis File”
* Nella scheda “Block Symbol” si ha una vista dei segnali di interfaccia e come essi sono stati interpretati. Il sistema in base al nome del segnale cerca di interpretarne la funzione, ma per alcuni è necessario definire tale funzione manualmente, infatti sono presenti diversi warning ed errori.



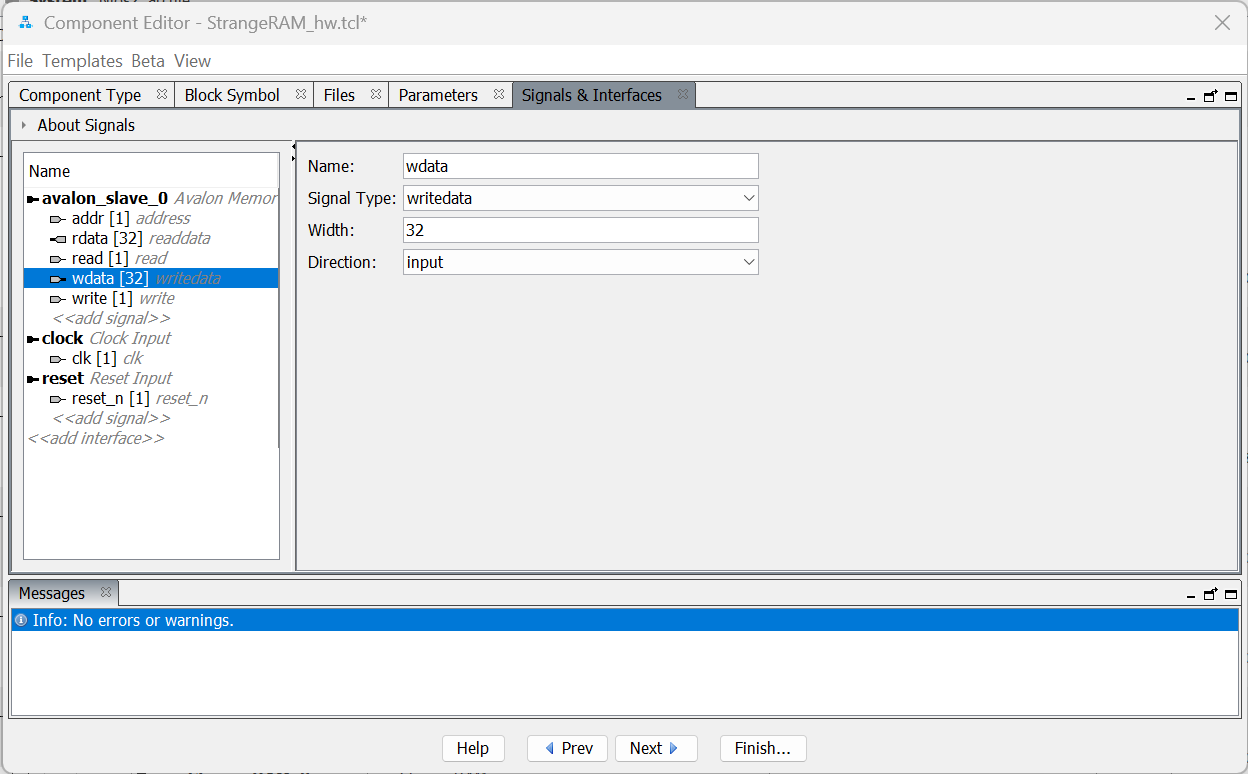
* Nella scheda “Signals & Interfaces” sono presenti tre interfaccie (clock, reset, avalon slave) ciascuna con i propri segnali. L’interfaccia di clock e di reset dovrebbero essere già correttamente realizzate. Ma l’interfaccia “avalon slave” deve essere configurata.
  + Si evidenzi (cliccandoci sopra) l’interfaccia in questione e si definisca attraverso il menù a tendina l’interfaccia di reset associata



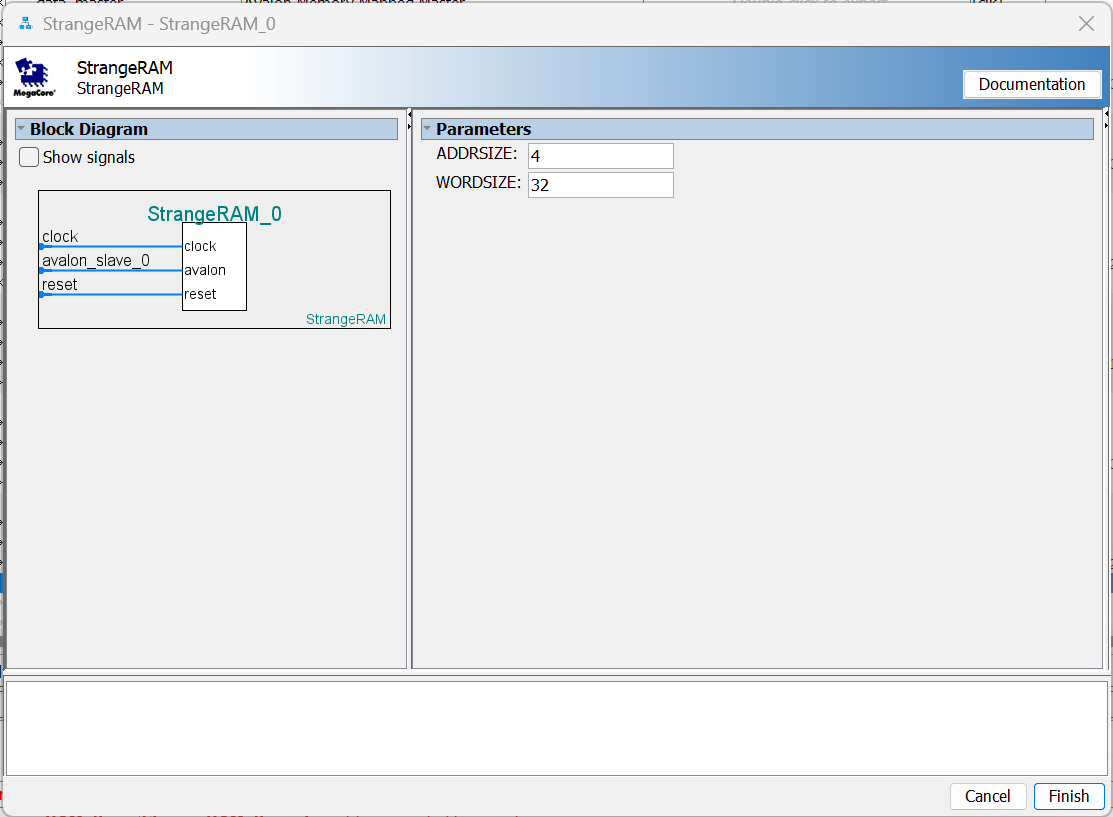
* + Successivamente si evidenzi il segnale che identifica l’indirizzo (addr) e si definisca quale “Signal Type” l’opzione “address”



* + Analogamente per il segnale rdata si scelga come tipo readdata
  + E ancora per wdata il tipo writedata
  + Se non vi sono più errori si può cliccare su <Finish>



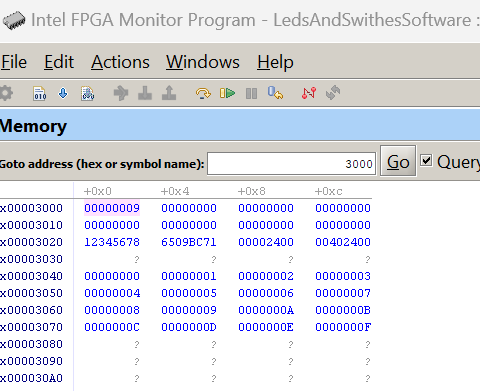
* Si salvi opportunamente il componente che adesso può essere importato nell’architettura attraverso il tool “Platform Design”
* In fase di importazione si definisca opportunamente il parametro inerente alla profondità di memoria desiderata (ADDRSIZE) ad esempio pari a 4 (per totali 16 celle di memoria)



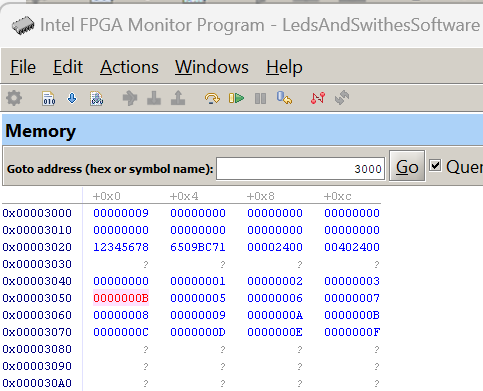
* Si colleghino opportunamente interfacce di clock, reset alla rete globale.
* Si colleghi l’interfaccia “Avalon Slave” con il segnale “data master” del NIOS2
* Si assegni un opportuno indirizzo di memoria al nuovo blocco o manualmente oppure utilizzando la facility :

System > Assign Base Address  
  
Ma poichè in questo caso gli indirizzi delle varie periferiche vengono TUTTI ricalcolati ed aggiornati, per evitare di dover modificare il software già sviluppato si possono vincolare gli indirizzi dei blocchi realizzati precedentemente cliccando sull’icona a “lucchetto” ad essi vicina oppure cliccando col tasto destro su di essi e scegliendo “Lock Base Address”

* Si prosegua quindi rigenerando l’intero sistema.
* Dal punto di vista dell’Istanziazione dell’elemento a livello della top level entity non vi è alcuna modifica da fare in quanto non sono stati modificati i segnali di uscita dell’architettura.
* Si ricompili l’intero progetto in Quartus
* Si operi il download su DE1-SoC
* Ricreando un nuovo progetto all’interno di Intel FPGA Monitor Program oppure aprendo il vecchio progetto che condivide gli elementi essenziali dell’architettura, collegandosi al sistema ed aprendo la scheda di analisi delle locazioni di memoria alla locazione delle periferiche (0x3000) si troverà la mappatura delle locazioni di memoria assegnate al nuovo elemento.



* Si noti in particolare che quando si proverà a scrivere in queste celle un qualsivoglia valore, sebbene esso sarà scritto internamente in modo corretto, in fase di lettura, per le scelte fatte ad esso vi si sommerà l’indirizzo della cella stessa (Es: scrivendo il numero 7 alla locazione 0x3050 ) ciò che vi si leggerà sarà 11 (ovvero in esadecimale 0xB).



Si replichi adesso l’intero processo realizzando un’interfaccia che, interfacciandosi al Bus AvalonMM, sia in grado di scrivere sul display a 7 segmenti, in formato esadecimale, il dato che gli viene fornito attraverso la procedura di “write” e viceversa sia in grado di riportare nel medesimo registro il dato che compare sul display, evidentemente azzerando i primi 8 bit (più significativi) che non possono essere visualizzati su un display a 6 caratteri.

Il codice da impiegare potrebbe essere il seguente:

**module** SeveneSEG (clk, address, readdata, writedata, read, write, reset\_n, hexa,hexb,hexc,hexd,hexe,hexf);

input clk;

input address;

output [31:0] readdata;

input [31:0] writedata;

input read,write;

input reset\_n;

output [6:0] hexa,hexb,hexc,hexd,hexe,hexf;

reg [31:0] reg\_data;

wire [31:0] filtered\_data;

assign readdata = (read) ? filtered\_data : {31{1'bz}};

assign filtered\_data = {32'b0 | reg\_data};

always @(posedge clk)

if (write)

reg\_data = writedata;

SEG7\_LUT u0 (clk,hexa,reg\_data[3:0]);

SEG7\_LUT u1 (clk,hexb,reg\_data[7:4]);

SEG7\_LUT u2 (clk,hexc,reg\_data[11:8]);

SEG7\_LUT u3 (clk,hexd,reg\_data[15:12]);

SEG7\_LUT u4 (clk,hexe,reg\_data[19:16]);

SEG7\_LUT u5 (clk,hexf,reg\_data[23:20]);

**endmodule**

**module** SEG7\_LUT (clk,oSEG,iDIG);

input clk;

input [3:0] iDIG;

output reg [6:0] oSEG;

always @(posedge clk)

begin

case(iDIG)

4'h1: oSEG = 7'b1111001; // ---t----

4'h2: oSEG = 7'b0100100; // | |

4'h3: oSEG = 7'b0110000; // lt rt

4'h4: oSEG = 7'b0011001; // | |

4'h5: oSEG = 7'b0010010; // ---m----

4'h6: oSEG = 7'b0000010; // | |

4'h7: oSEG = 7'b1111000; // lb rb

4'h8: oSEG = 7'b0000000; // | |

4'h9: oSEG = 7'b0011000; // ---b----

4'ha: oSEG = 7'b0001000;

4'hb: oSEG = 7'b0000011;

4'hc: oSEG = 7'b1000110;

4'hd: oSEG = 7'b0100001;

4'he: oSEG = 7'b0000110;

4'hf: oSEG = 7'b0001110;

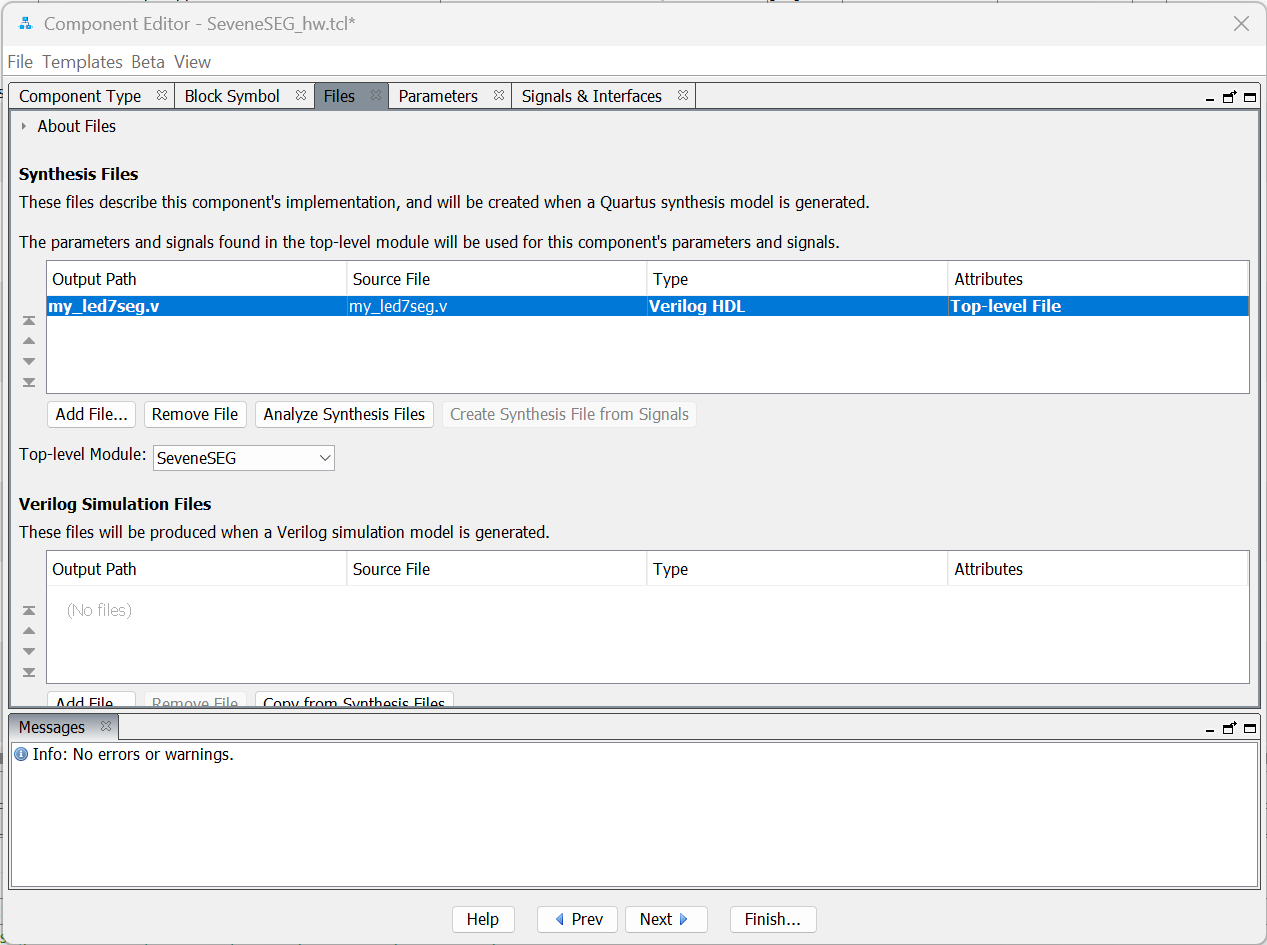
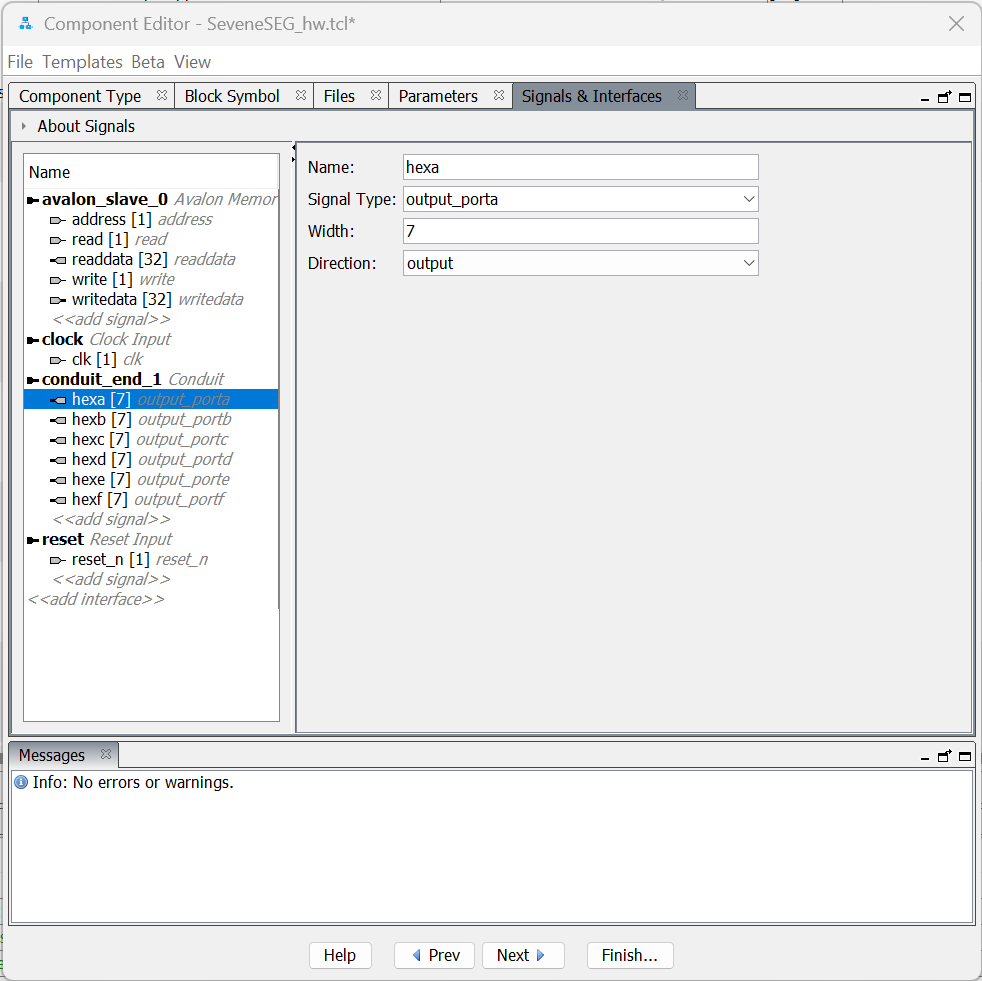
4'h0: oSEG = 7'b1000000;

endcase

end

**endmodule**

Nella fase di creazione del nuovo componete all’interno del tool “Platform Designer” vi sono però alcune procedure aggiuntive:

* Poiché il codice si compone di due moduli bisogna indicare quale sia il modulo che funge da interfaccia (ovvero top level Module)  
  
* Bisogna reare una nuova interfaccia di tipo “Conduit” nella quale includere tutti i segnali d’uscita che vanno a pilotare i 6 display LED a 7 segmenti, fornendo un nome diverso a ciascuna delle 6 diverse porte d’uscita  
  
* Una volta completata l’architettura, quando essa deve venir istanziata nella top level entity che verrà successivamente compilata bisogna ricordarsi di creare i corretti collegamenti tra le porte d’uscita dell’architettura basata sul NIOS e le porte d’uscita dell’FPGA stessa, connesse rispettivamente al Display 7 Seg.



Arrivati a questo punto si può compilare e riprogrammare la scheda DE1-SoC.

Ora accedendo alla memoria dell’architettura attraverso il tool “Monitor Program” e scrivendo nella corretta locazione di memoria (in questo caso 0x3028) il dato exadecimale, esso verrà visualizzato direttamente sul display della scheda ed al contempo letto anche alla locazione successiva (relativa all’indirizzo 1) in quanto in hardware non è stato introdotto alcun controllo sulla linea di indirizzo.

