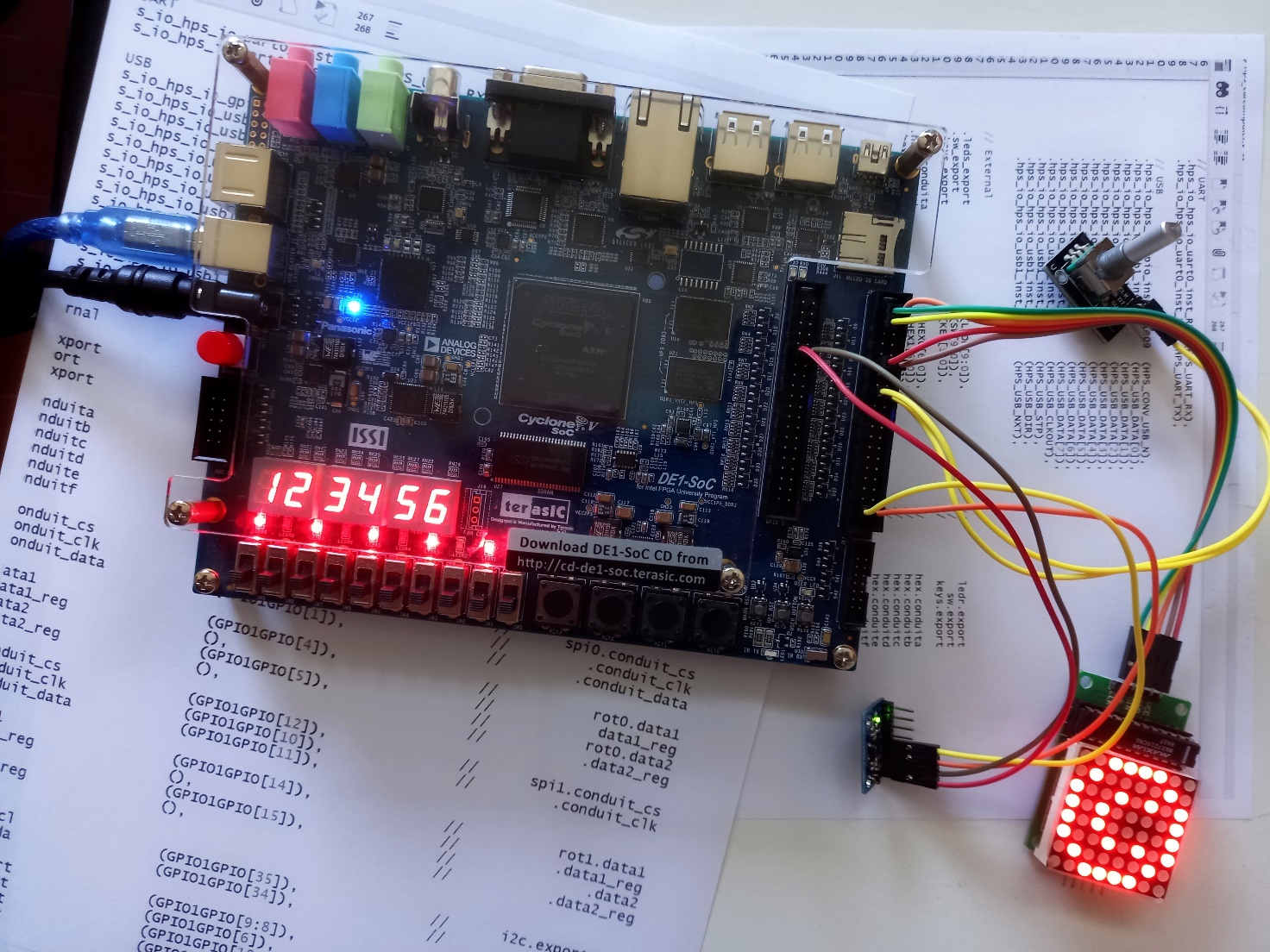
Progettazione di Sistemi Elettronici

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

Tutorial 4



**Progetto di un sistema di intefaccia SPI.**  
**Hardware** impiegato: Terasic DE1-SoC Board  
**Software** Impiegato: Quartus 22.1, System Builder

Tutorial 4

Realizzazione di un sistema in grado di generare un segnale SPI

Descrizione: In questo tutorial si andrà a generare su FPGA un sistema modulare in grado di generare un segnale SPI in grado di pilotare eventuali dispositivi che usino detto protocollo. Nel dettaglio si andrà a utilizzare il protocollo SPI per scrivere nei registri di un sistema per la gestione di una matrice di LED basato sull’integrato MAX7219

Scopo: lo scopo di questo tutorial è quello di familiarizzare con le interfacce esterne e con la generazione di segnali che sottostiano a particolari specifiche attraverso la definizione di un sistema atto allo scopo. Sarà inoltre approfondito l’impiego di un tool, sviluppato all’interno di Quartus utile per monitorare l’evoluzione temporale di segnali.

Apprendimento previsto:

* Definizione di un sistema completo di interfaccia SPI attraverso uno schema gerarchico.
* Approfondimento del linguaggio Verilog e del simulatore Questa/ ModelSim
* Familiarizzare col tool “Logic Analyzer” per monitorare i segnali interni all’FPGA
* Familiarizzare con In-System Memory Editor per interagire con le memorie

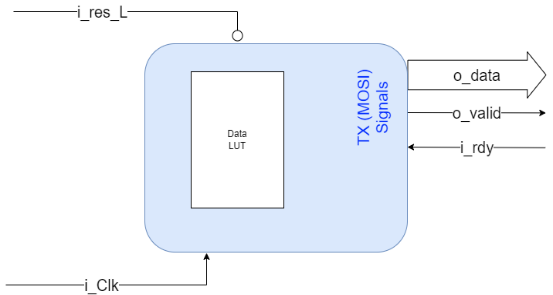
# Premessa

A differenza dei Tutorials precedenti, questa volta si partirà definendo e simulando il sistema che si vuole sviluppare all’esterno del sistema integrato Quartus, per poi importarlo, una volta che le simulazioni abbiano dato esito positivo all’interno del tool di sintesi.

Sebbene il protocollo SPI possa essere impiegato sia in lettura che in scrittura, in questo caso esso verrà impiegato esclusivamente in scrittura, il che porta ad una semplificazione del sistema completo.

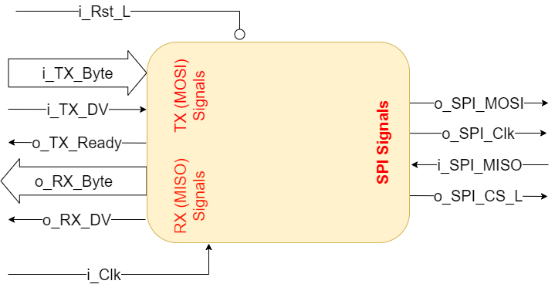
Il sistema che si andrà a realizzare sarà composto essenzialmente da due blocchi:

* Il primo in cui risiedono tutta la serie di dati che si vogliono trasmettere sequenzialmente, memorizzati in una LUT (Look Up Table) che attraverso una modalità di tipo handshaking fornisce i dati (quando richiesti) in parallelo al secondo blocco, aggiungendo un’informazione di “data valid” quando il dato è disponibile e stabile.



* Il secondo blocco che si interfaccia col primo invece genera la richiesta di un dato e quando ricevuto, provvede a generare la sequenza di f.d.o che contraddistingue il protocollo SPI.

Questo secondo blocco è predisposto anche per leggere eventualmente dei dati attraverso il protocollo SPI, ma in questo esercizio questa parte non sarà utilizzata.



# Files Verilog e Simulazioni

L’intero progetto è descritto all’interno di tre files VerilogHDL disponibili nel sito moodle del corso:

* SPI\_Master16b.v: descrive il blocco che genera i segnali SPI
* SPI\_LUT: Descive il blocco che memorizza i dati da inviare
* TOP\_Spi: Unisce i due blocchi nel dispositivo complessivo

Si vuole impiegare lo strumento di simulazione per analizzare e validare il funzionamento.   
(Per un funzionamento dettagliato si rimanda al tutorial Specifico).

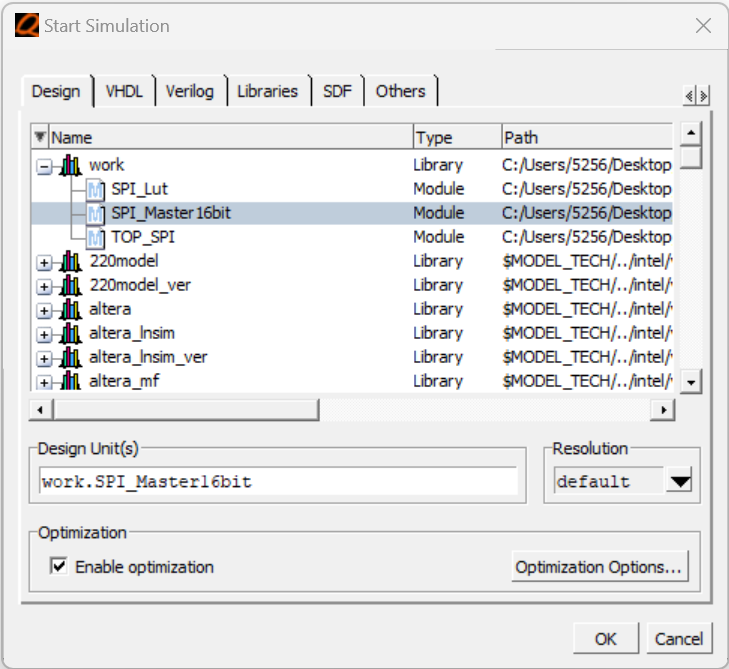
* Si apra il tool di simulazione (ModelSim/Questa)
* Si crei un progettto opportuno
* Si importino i tre files in questione
* Si compilino tutti i Files

### Simulazione di SPI\_Master16b

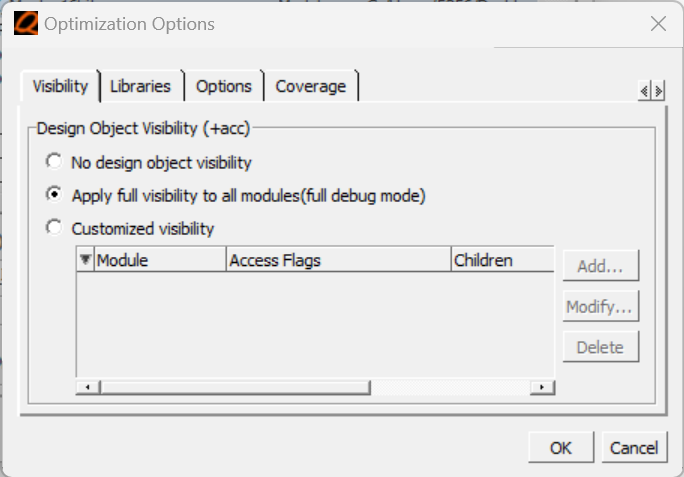
Si passi ora alla simulazione del modulo “SPI\_Master16b”

> Simulate > Start Simulation

Nella finestra si selezioni il modulo prescelto e si clicci su “**Optimization Options**”



Selezionare “Apply Full Visibility” che consente di monitorare tutti i segnali ed i processi interni al modulo



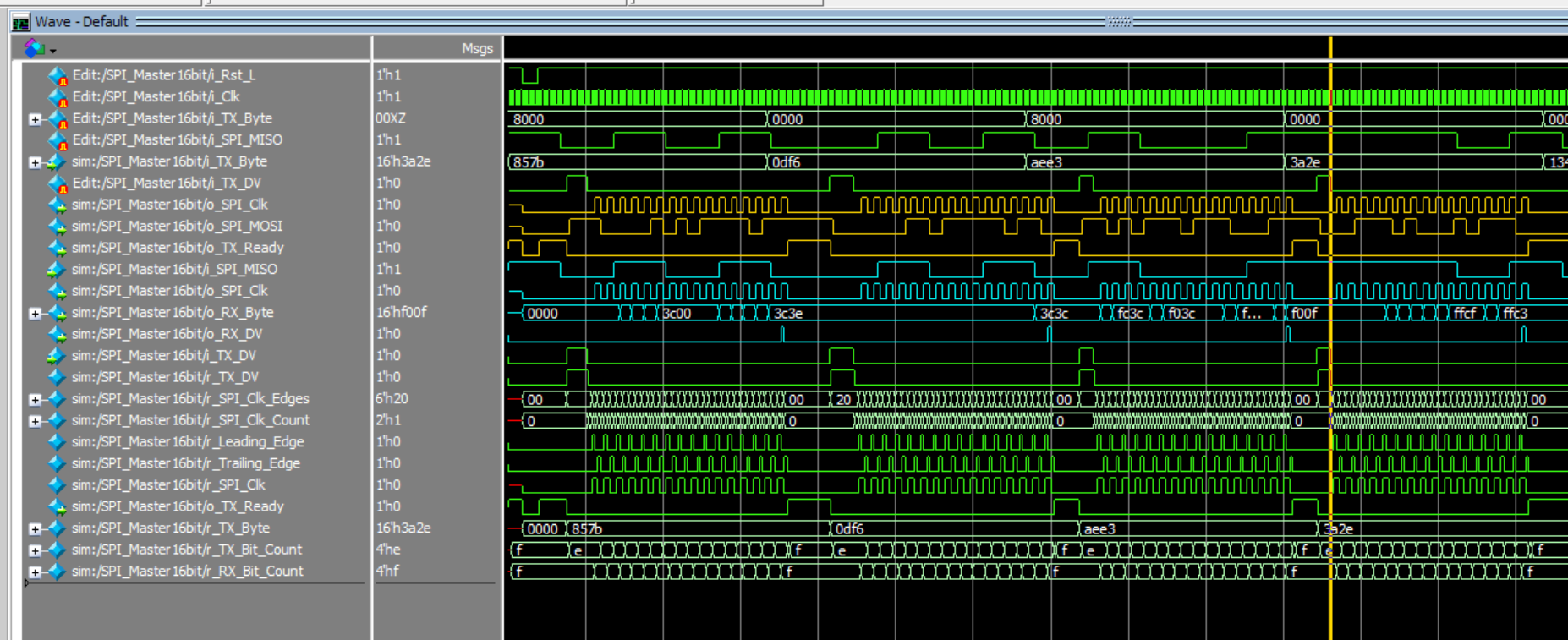
> OK

> OK

Si vada ora a creare dei segnali significativi per tutti gli ingressi del modulo, in particolare:

* Un impulso unico negativo sul segnale i\_Rst\_L
* Un clock su i\_Clk
* I dati che si vorrebbe trasmettere sul bus i\_TX\_Byte (si noti che per un bug del tool di questi dati, quando visualizzati in formato Ottale o Esadecimale, sono rappresentati, quando appaiono come “stimoli”, solo in base al loro bit più significativo. Pertanto ad essi va abbinata una visualizzazione come “segnale interno” che ne consente una rappresentazione con tutti i bit significativi)
* Degli Impulsi per il segnale “data valid” i\_TX\_DV intercalati ai dati precedenti (si noti che questi, insieme ai precedenti, devono essere sufficientemente spaziati da consentire al dispositivo di ultimare la trasmissione prima che venga chiesta la trasmissione di un nuovo dato – eventualmente provare a vedere cosa accade se dati ed impulsi “data\_valid” dovessero risultare troppo ravvicinati)
* Gli eventuali segnali in lettura MISO per testare la capacità di lettura del dispositivo i\_SPI\_MISO

Si può ora simulare il modulo evidenziandone tutti i segnali sia presenti nei vari registri che alle uscite.



Notare in particolare (impiegando lo zoom e/o evidenziando i segnali specifici) e comparandone il funzionamento col codice Sorgente VerilogHDL.

La generazione dei sincronismi:

* Che la procedura viene inizializzata quando i\_TX\_DV è a livello alto (resettando il contatore r\_SPI\_Clk\_Edges) e prosegue solo quando questo passa dallo allo stato alto a quello basso
* Il contatore dei cicli di clock “r\_SPI\_Clk\_Count” rispetto al quale vengono definiti gli impulsi relativi all’arrivo di un Leading\_edge o di un Trailing\_Edge r\_Leading\_Edge, r\_Trailing\_Edge salvati negli opportuni registri
* La generazione del segnale di registro r\_SPI\_Clk
* Il contatore (a ritroso) dei fronti r\_SPI\_Clk\_Edges
* La generazione di “o\_TX\_Ready” usato per richiedere un nuovo dato e attivo solo quando il contatore precedente ha raggiunto lo 0

Successivamente notare la memorizzazione di due versioni “registrate” **r\_TX\_Byte** e **r\_TX\_DV** dei segnali rispettivi di ingresso **i\_TX\_Byte** e **i\_TX\_DV** in modo da mantenere stabile il dato da trasmettere anche qualora dovessero modificarsi gli ingressi.

Notare quindi il contatore relativo al bit da trasmettere “r\_TX\_Bit\_Count”, il suo ripristino (reset) all valore massimo (0xF = 15) in seguito ad un reset globale oppure in seguito al segnale “o\_TX\_Ready” ed al suo decremento in corrispondenza ad un “Leading Edge” o ad un “Trailing edge” in base alla modalità di trasmissione prescelta, e di come in base a al valore assunto da quest’ultimo sull’uscita si viene a formare il bit desiderato:

1. o\_SPI\_MOSI <= r\_TX\_Byte[r\_TX\_Bit\_Count];

Notare quindi la generazione del segnale d’uscita “o\_SPI\_Clk” che assume il valore di default i n caso di reset oppure il valore presente nel registro “r\_SPI\_Clk”

Si noti da ultimo il funzionamento in lettura del protocollo SPI: sulla linea i\_SPI\_MISO vengono forniti sequenzialmente dei dati, questi, in base ad un contatore r\_RX\_Bit\_Count vengono forniti in modo continuo direttamente all’uscita o\_RX\_Byte, che pertanto continua ad assestarsi bit a bit ad ogni modifica del dato d’ingresso.

221 o\_RX\_Byte[r\_RX\_Bit\_Count] <= i\_SPI\_MISO;

Quando tutti I dati sono presenti e stabili sul bus d’uscita viene attivato il segnale o\_RX\_DV che potrebbe essere impiegato da un eventuale ricevitore per determinare l’istante in qui effettuare la lettura del dato completo.

### Simulazione di SPI\_LUT

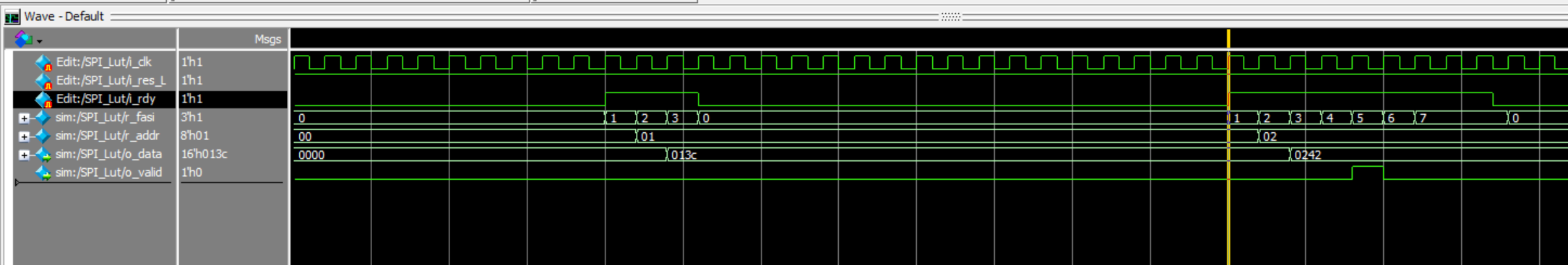
Con la procedura suggerita al punto precedente si passi ora a simulare il modulo “SPI\_LUT”:

Si generino gli ingressi necessari ovvero

* Un segnale di clock
* Un impulso negativo sul reset
* Una serie di impulsi più o meno larghi sul segnale i\_rdy

Si può pertanto constatare, per come è stato concepito il sistema

* Che quando arriva una richiesta attraverso i\_rdy parte la procedura per fornire il dato
* Questo fa partire un “contatore delle fasi” r\_fasi che
  + All’istante 2 incrementa l’indirizzo della ROM r\_addr dalla quale prelevare il dato
  + All’istante 3 questo dato viene fornito in uscita o\_data
  + All’istante 5 si attiva il segnale di o\_valid
* Quando i\_rdy torna ad abbassarsi il contatore delle fasi si resetta e alla prossima richiesta riparte la procedura. Una volta esauriti i dati in memoria la trasmissione riparte dal primo.
* Si noti ancora che per come realizzato il dispositivo il segnale i\_rdy deve rimanere alto per un numero sufficiente di clock per garantire la generazione del segnale o\_valid



Comunque va sottolineato che il segnale i\_rdy verrà generato dal blocco ***Master\_spi16b*** e sarà da questo riportato allo stato basso solamente dopo aver ricevuto da SPI\_Lut il dato da trasmettere e congiuntamente il segnale “data\_valid”.

Da Ultimo si noti che il primo dato in memoria alla posizione 0 non viene trasmesso per cui in quella posizione si può introdurre un dato vuoto “dummy”.

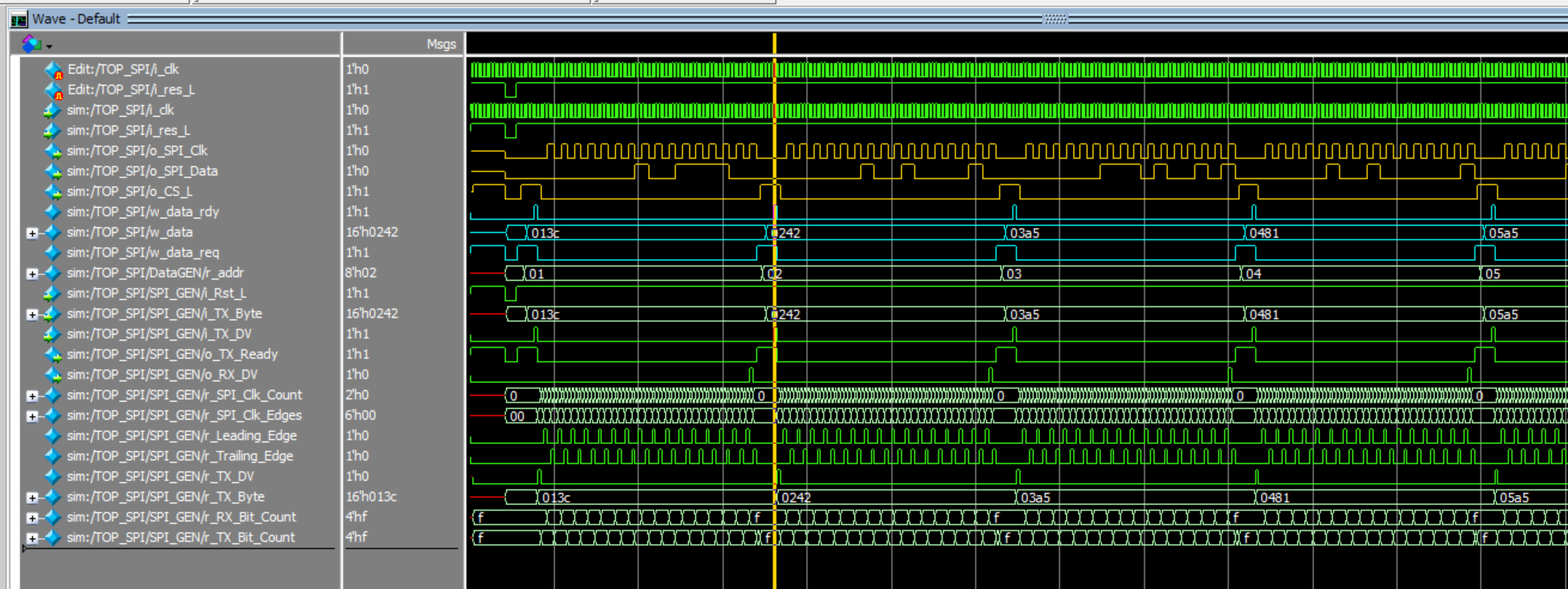
Per vedere il funzionamento congiunto dei due blocchi si può quindi passare alla simulazione del blocco TOP\_SPI che riunisce in un unico sistema i due blocchi precedentemente sviluppati e studiati.

### Simulazione di TOP\_SPI

Con la procedura suggerita al punto precedente si passi ora a simulare il modulo “SPI\_LUT”:

Si generino gli ingressi necessari ovvero

* Un segnale di clock
* Un impulso negativo sul reset



E si visualizzino i segnali ritenuti più significativi quali ad esempio

* Il dato da trasmettere w\_data
* i vari sincronismi di handshaking w\_data\_rdy, w\_data\_req
* I segnali d’uscita del protocollo SPI o\_SPI\_Clk, o\_SPI\_Data, o\_SPI\_CS
* Si visualizzi inoltre la congruità del dato da trasmettere col suo relativo protocollo seriale, ad esempio il primo dato 0x013C = 0000\_0001\_0011\_1100

# Implementazione su FPGA

Impiegando il tool “SystemBuilder” creare lo scheletro di un sistema che impieghi

* Clock
* Swithes + Buttons
* GPIO1 (modo Default)

Scegliere una directory nel quale salvare il progetto e dare un nome a quest’ultimo (ad esempio Tut4SPI)

Copiare o spostare i tre files VerilogHDL all’interno della directory del progetto appena generato.

Aprire il progetto all’interno di Quartus.

Importare i tre files VerilogHDL

Project > Add/Remove Files in project

Modificare la Finestra “Project Navigator” ponendola in modalità Files.

Cliccare col tasto desto sul file TOP\_Spi 🡪 Create Verilog Instanstation Template File from Current File

Nella Directory del progetto appare il file: TOP\_SPI\_inst.v che può essere esaminato per vedere come istanziare il modulo.

RiModificare la Finestra “Project Navigator” ponendola in modalità Hierarchy

Doppio Click sulla Top Level Entity ( Tut4SPI)

Includere nel file Verilog l’istanziazione del modulo TOP\_SPI\_inst (visto ai passi precedenti)

Modificare i collegamenti in modo da rispecchiare i collegamenti con le periferiche. In particolare

* Collegare il clock con il segnale CLOCK\_50
* Collegare il reset col segnale KEY[0]
* Collegare rispettivamente
  + o\_SPI\_Clk con GPIO1GPIO[0]
  + o\_SPI\_Data con GPIO1GPIO[1]
  + o\_SPI\_CS\_L\_sig con GPIO1GPIO[2]

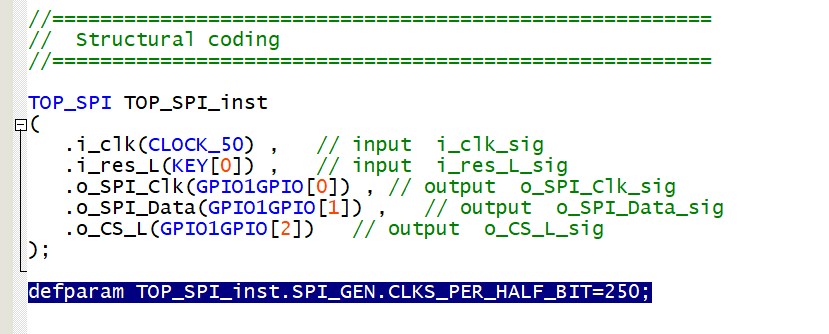
Inoltre modificare la frequenza di lavoro dell’uscita SPI: considerando che la frequenza di lavoro SPI va circa da 10KHz a 10MHz bisogna settare quest’ultima modificando il parametro CLKS\_PER\_HALF\_BIT contenuto all’interno del modulo SPI\_master16b. Ad esempio portando il valore a 250 vi saranno 500 cicli di clock primario (a 50MHz) per ogni ciclo di clock SPI portando quindi la frequenza SPI a 100KHz

Questa operazione può essere fatta in vari modi, ma la più comoda è introducendo l’istruzione

defparam TOP\_SPI\_inst.SPI\_GEN.CLKS\_PER\_HALF\_BIT=250;

che sovrascrive I parametri di default in forma gerarchica per i moduli interessati.

Alla fine il codice da fornire alla Top Level Entity potrebbe essere il seguente:



# Collegamenti e Download

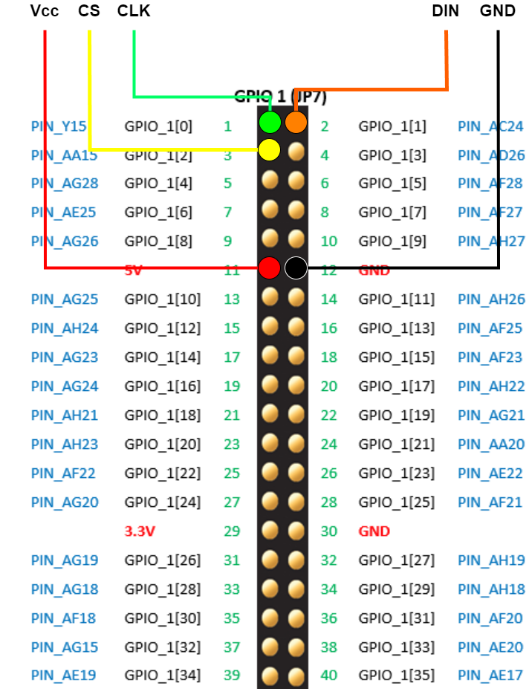
Il sistema completo può quindi essere compilato fino al file finale “.sof” pronto per configurare l’FPGA. Prima di ciò però bisogna collegare opportunamente il dispositivo.

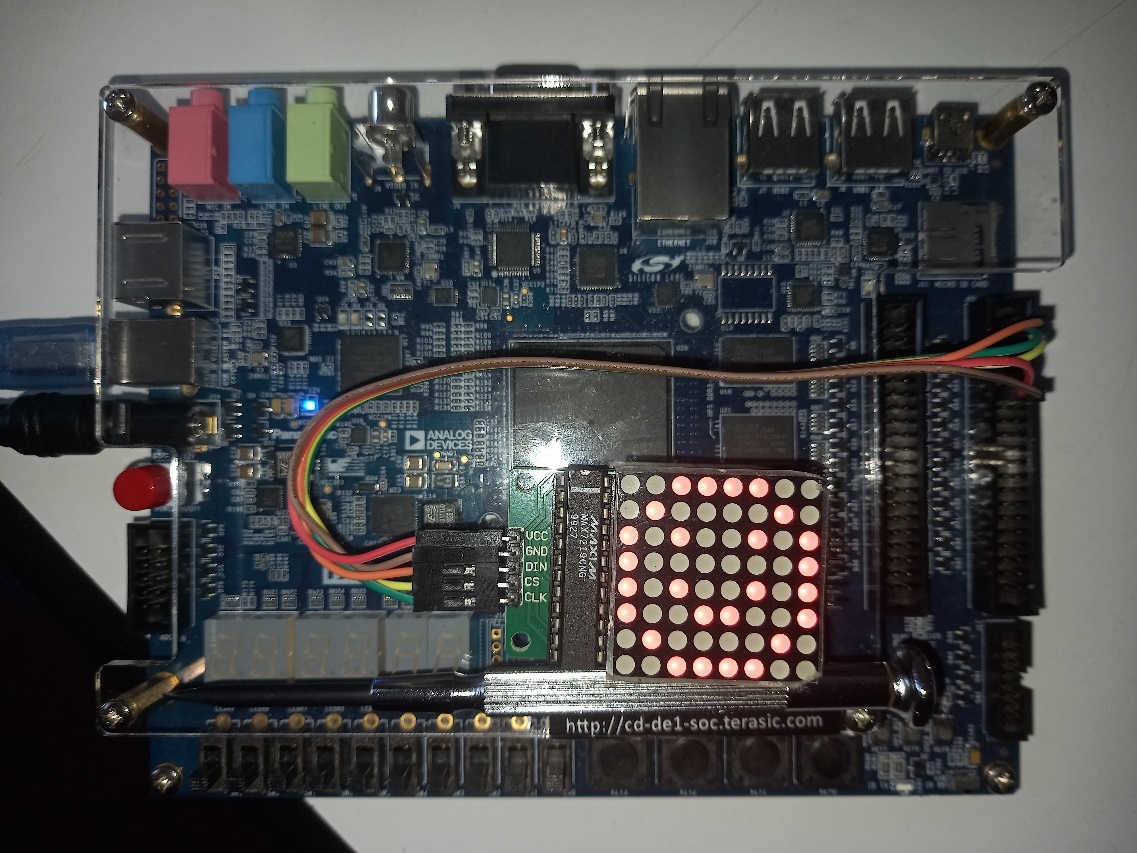
Esso viene controllato attraverso 3 segnali SPI (CLK,DIN,CS) e naturalmente è alimentato con VCC e GND.

Le alimentazioni Vcc e GND possono essere prelevate direttamente dalla scheda DE1-SoC rispettivamente sui piedini 11 e 12 del connettore GPIO1, mentre per come disposto nel codice Verilog realizzato al passo precedente si colleghi

* CLK con GPIO1GPIO[0]
* DIN con GPIO1GPIO[1]
* CS con GPIO1GPIO[2]

Come da immagine sotto riportata





Nota: il sistema così configurato invia continuamente attraverso il protocollo SPI i dati di configurazione dei 15 registri, i cui valori sono contenuti all’interno della ROM contenuta nel modulo SPI\_LUT.

Per cambiare la visualizzazione si ricorda che:

* I registri tra 1 e 8 contengono rispettivamente i dati da visualizzare in ciascuna colonna
* Il registro 9 la modalità di decodifica (Normale =0x00)
* Il registro 10 l’intensità luminosa (sui 4 bit meno significativi) compresa tra 0x00 e 0x0F
* Il registro 11 il numero di colonne per il quale eseguire lo scanning
* Il registro 12 modalità Shutdown/Normal sul bit meno significativo (Normal=0x01)
* I registri 13 e 14 inutilizzati
* Il registro 15 modalità Test/Normal sul bit meno significativo (Normal=0x00)

# Logic Analizer

Un tool particolarmente interessante che viene messo a disposizione all’interno di Quartus è il “Signal Tap Logic Analyzer”. E’ questo un sistema atto a monitorare segnali sia interni che eventualmente esterni all’FPGA per validarne il corretto formato e la loro evoluzione temporale.

In Breve: questo tool serve, una volta stabiliti quali segnali da monitorare, a memorizzarli internamente all’FPGA ed a fornirne l’evoluzione in forma grafica su una finestra di interfaccia. Per fare questo però il sistema dovrà stabilire un segnale di campionamento e soprattutto **riservare opportune risorse interne all’FPGA** atte sia a memorizzare i segnali campionati, sia al loro trasferimento verso il tool di interfaccia. Tali risorse possono appesantire non poco la realizzazione del circuito finale pertanto è caldamente suggerito

* di non eccedere con la profondità di memoria o con la frequenza di campionamento
* di utilizzare questo strumento solamente in fase di debug e di eliminarlo dal sistema finale quando se ne fosse verificato il corretto funzionamento.

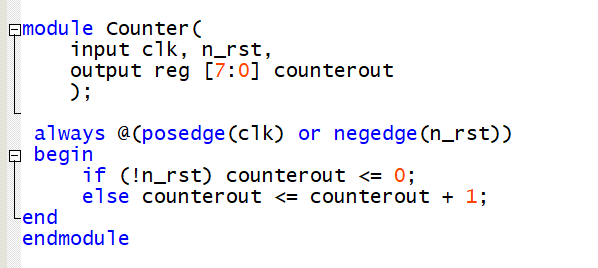
Si inizi prendendo nota delle risorse impiegate attualmente per la realizzazione del sistema

* analizzando sia i vari “report log”
* ricorrendo al Tool “Chip Planner”

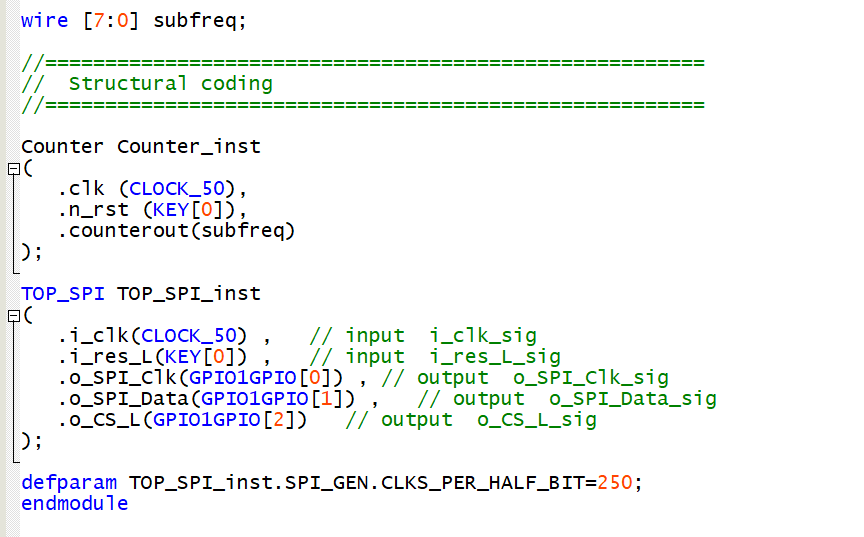
(in particolare quante ALM, Memorie e registri sono stati impiegati)

Innanzitutto, per evitare di dover campionare troppo fittamente (ovvero a 50MHz) i segnali di interesse si crei un “divisore di frequenza” ovvero un contatore binario ciclico che ricevendo in ingresso il clock di sistema a 50MHz fornisca in uscite diversi segnali ciascuno di frequenza dimezzata rispetto il precedente, ecco pertanto che alla quinta uscita del contatore troveremo un segnale a circa 1,5 MHz utile per campionare i segnali SPI che abbiamo scelto avere una frequenza di circa 100 KHz.

Si modifichi pertanto il codice inserendo un nuovo modulo, o all’interno di uno dei files già facenti parte del progetto oppure attraverso un nuovo file



e successivamente istanziandolo nella Top Level Entity creando degli opportuni segnali (subfreq) e collegando opportunamente tutti i segnali.

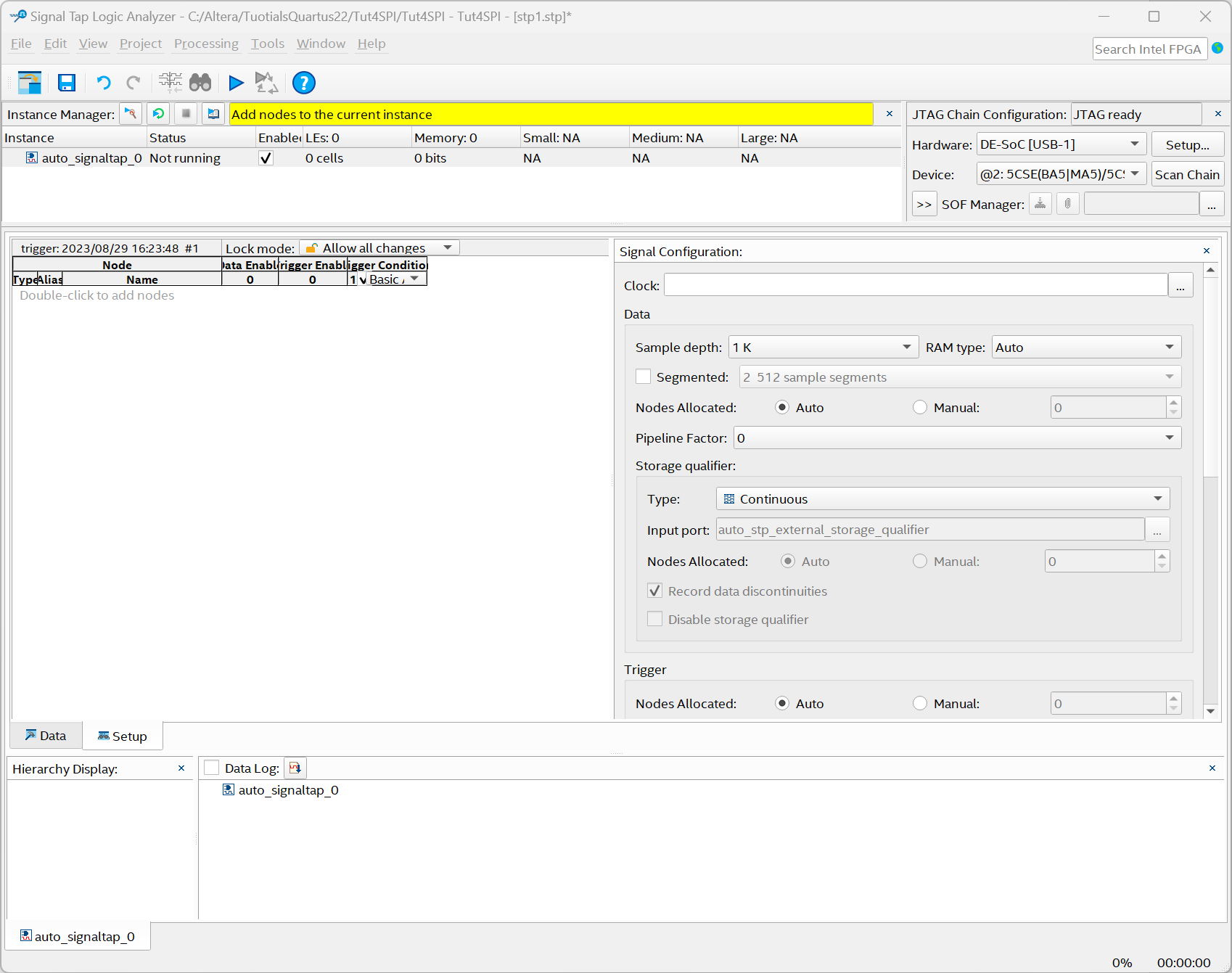


> Processig > Start > Start Analysis and Syntesis (Ctrl-K)

Successivamente si apra il tool “Signal Tap Logic Analyzer”:

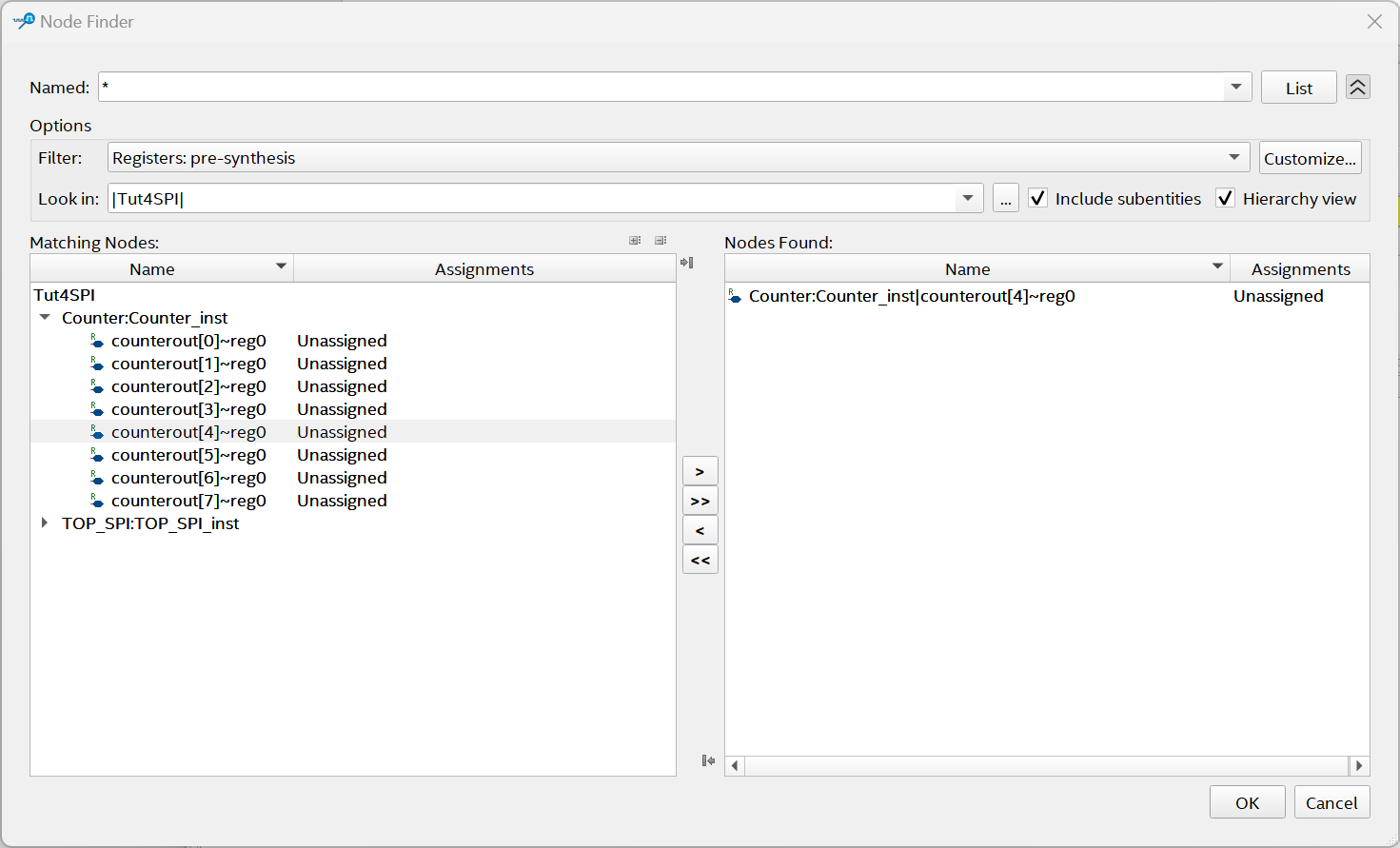
> Tools > Signal Tap Logic Analyzer

Nella finestra che si apre inizialmente definire quale segnale impiegare come clock (ovvero per campionare i segnali).



Nella scheda “Signal Configuration” si clicchi sui “…” a fianco della cella clock, si aprirà una nuova finestra.

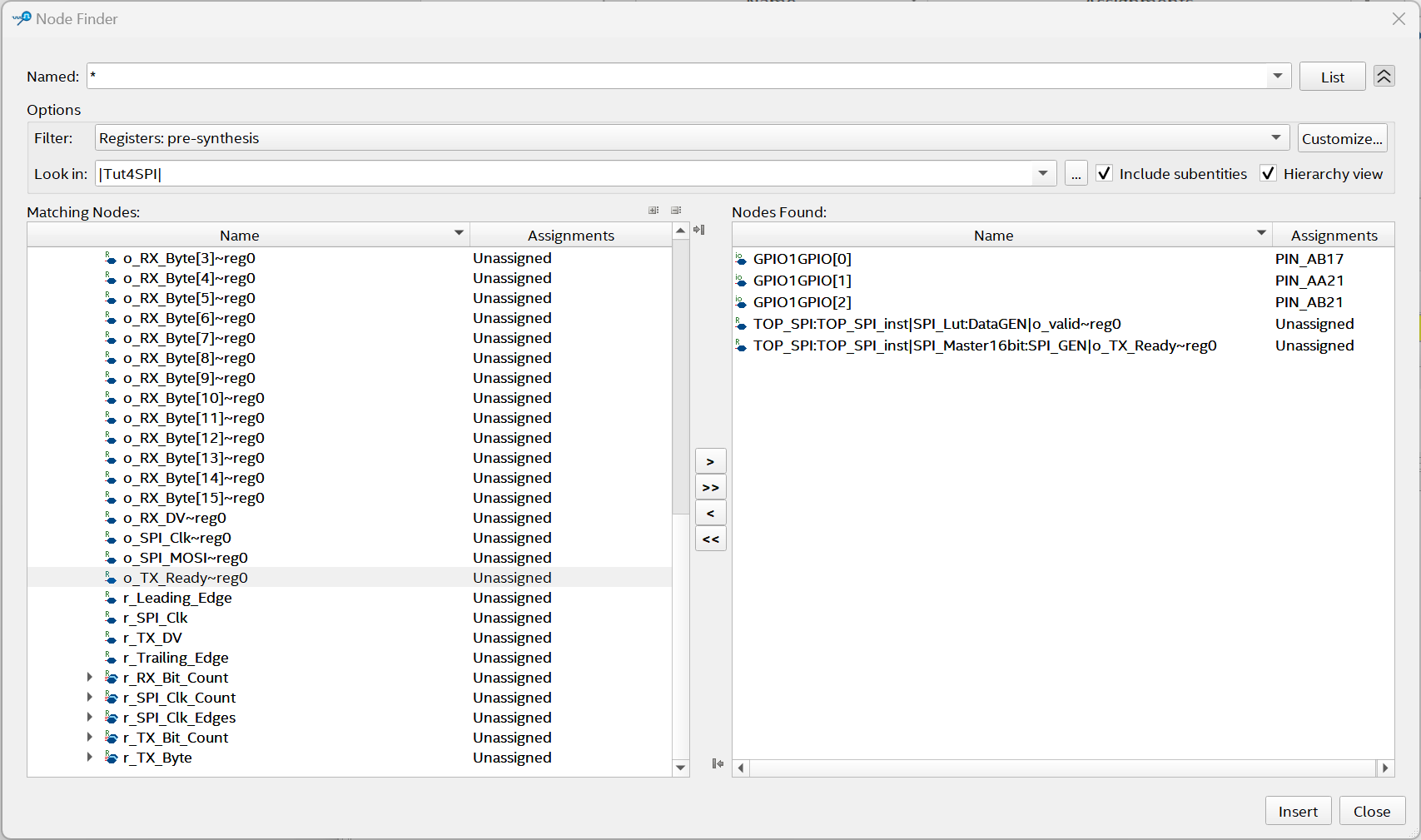
Settando il parametro “Filter” in modalità ad esempio “Registes:pre-synthesis” e successivamente cliccando su <List> comparirà una serie di segnali che possono essere scelti come clock di campionamento.



Si scelga ad esempio l’uscita[4] del contatore

<OK>

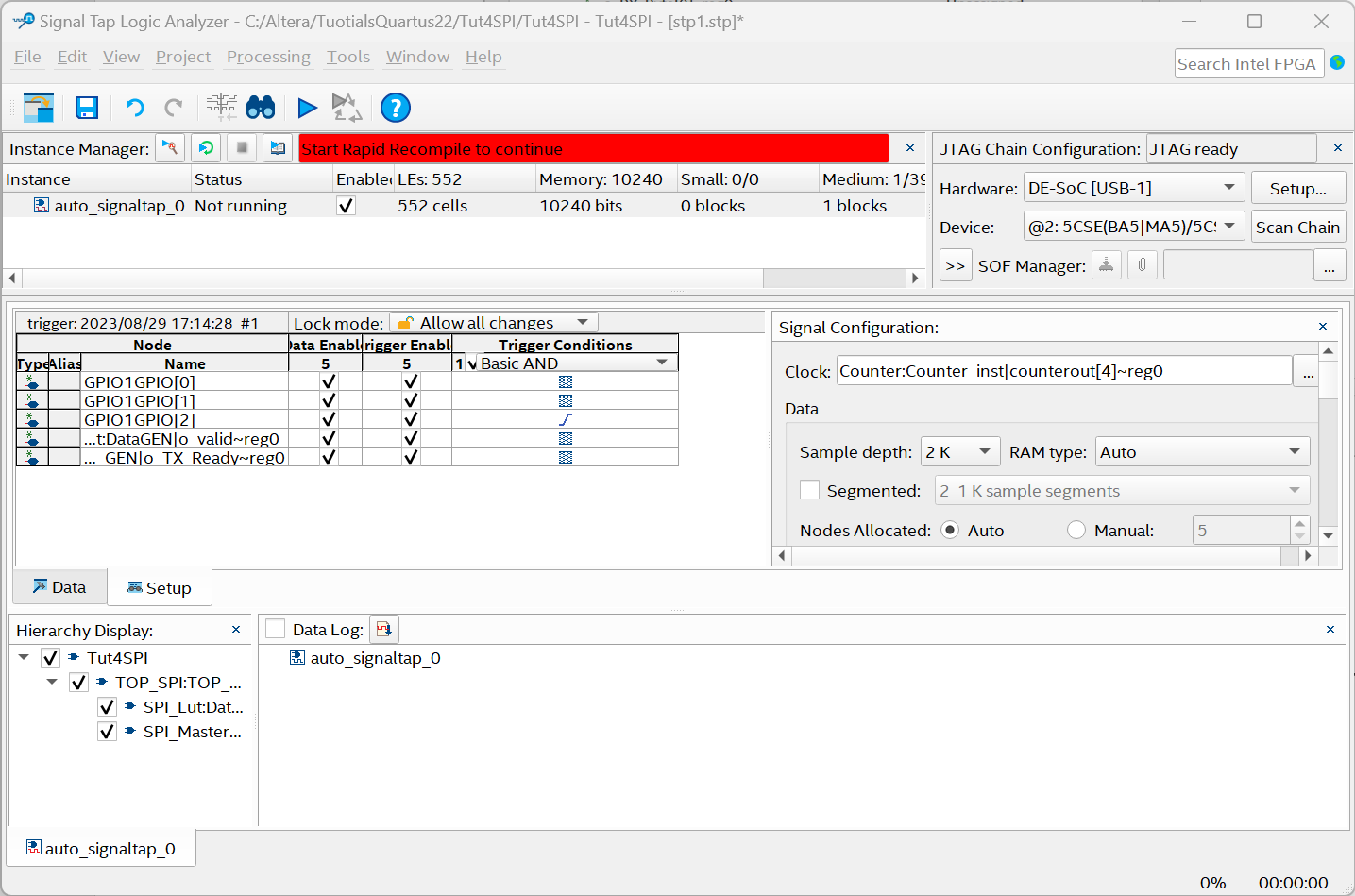
Si completi la schermata precedente fissando la profondità di memoria (ad Esempio 2K) e quali segnali si vogliono analizzare (ad esempio le tre uscite SPI ed i parametri interni di handshaking tra i due blocchi ) facendo doppio click nella finestra di sinistra e sfruttando lo stesso meccanismo visto sopra per filtrare ed individuare i segnali di interesse.

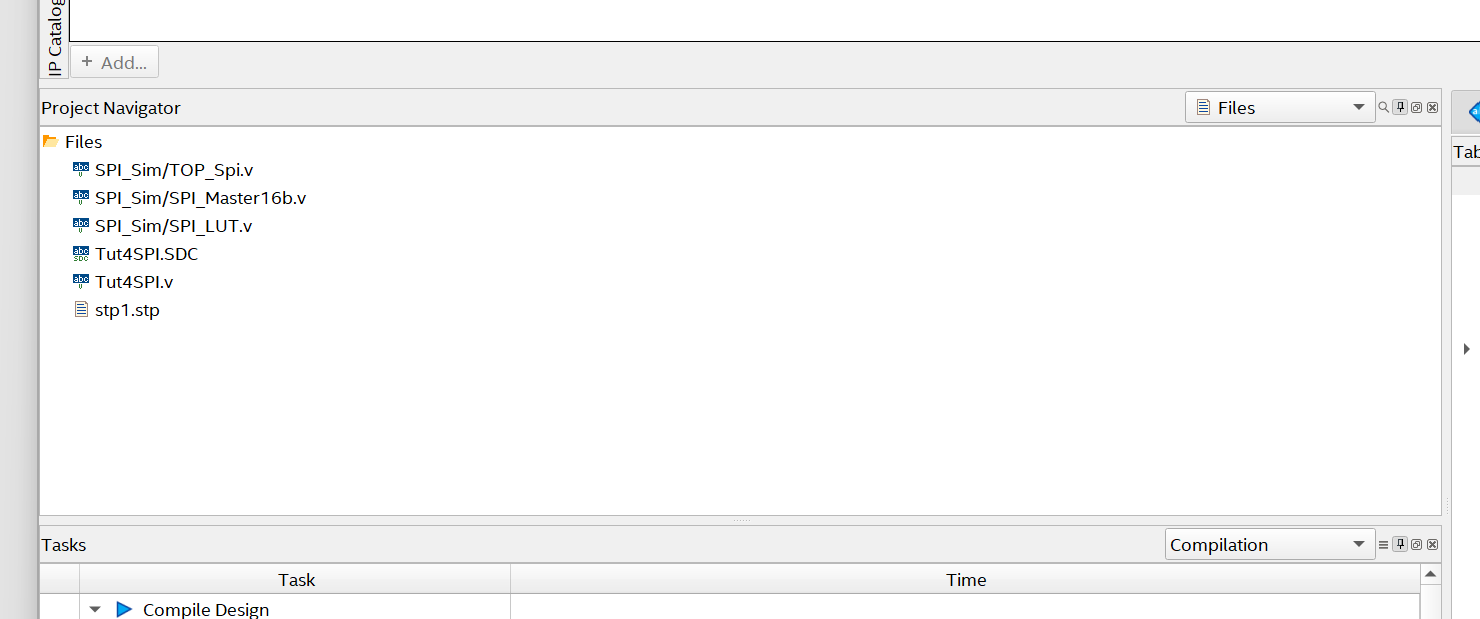


<Insert>

Nella finestra successiva cliccando col tasto destro sulla cella “Trigger condition” del segnale GPIO1GPIO[2] collegato col CS del segnale SPI si setti questa condizione allo stato di Raising Edge.

Si salvi il file (nome suggerito automaticamente stp1.stp) ed alla richiesta successiva se includerlo nel progetto Quartus si risponda affermativamente (In alternativa includere manualmente il file appena salvato tra i files del progetto)



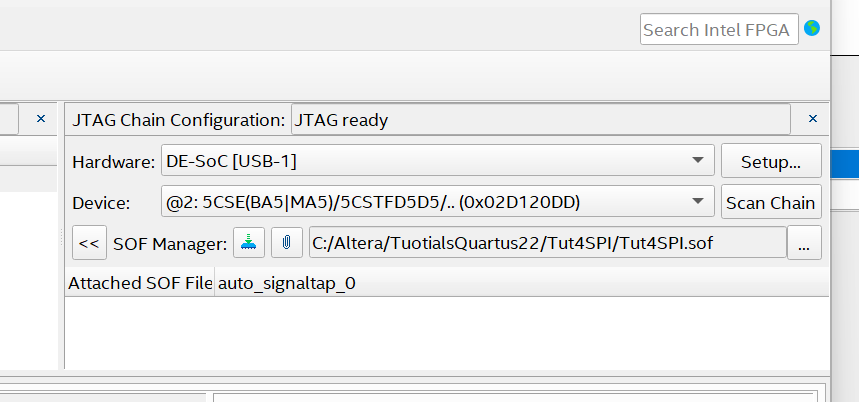


* Si ricomplili completamente il progetto (L’operazione può eventualmente essere eseguita direttamente all’interno della scheda relativa al “Signal Tap Analyzer” ).

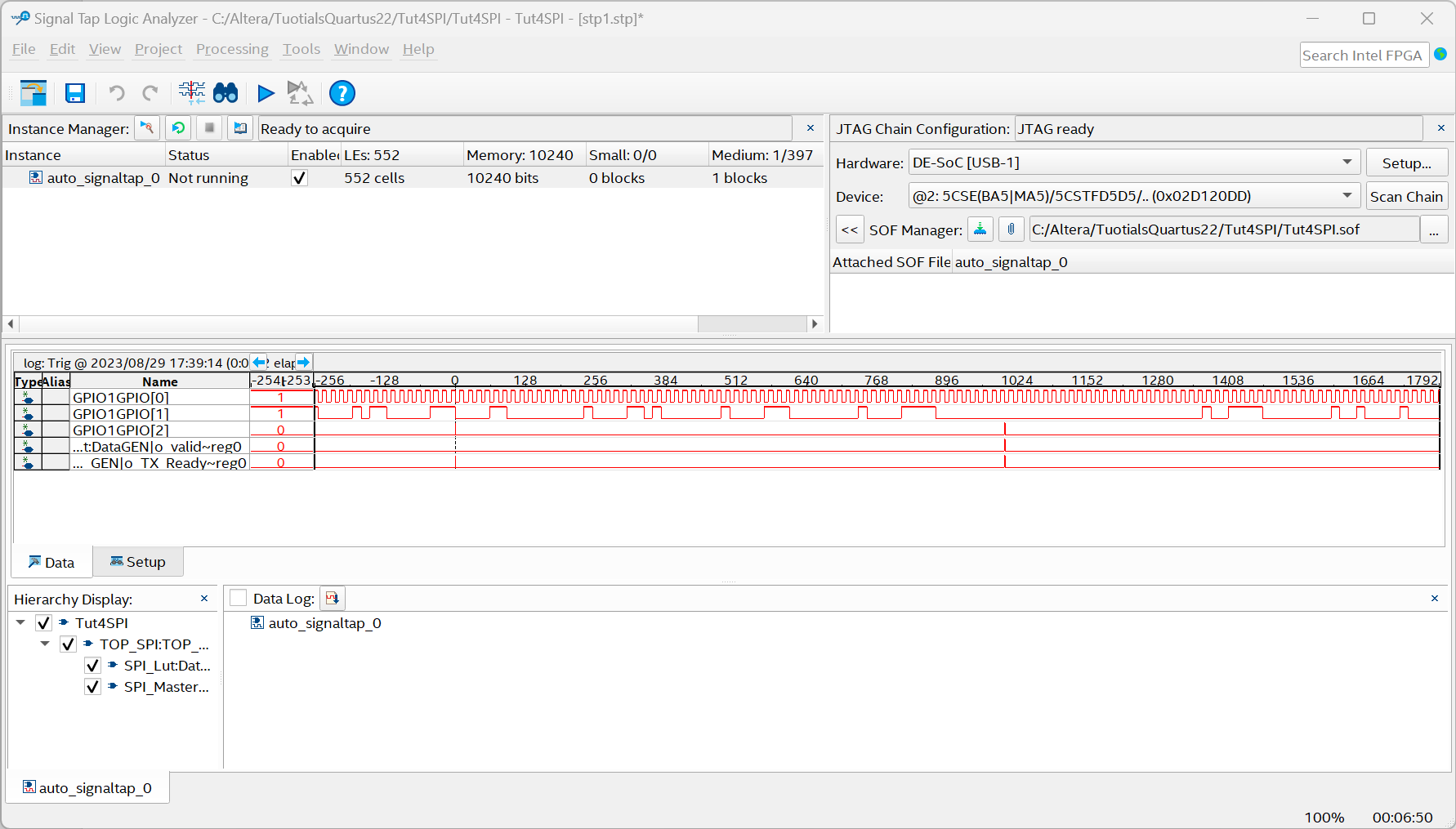
Processing > Start Compilation

* Si ri-esegua il download sulla scheda DE1-SoC

(anche questa parte può essere eseguita direttamente dall’interno della finestra relativa al “Signal Tap Analyzer”) configurando opportunamente la catena JTAG ed il file di configurazione.



A questo punto (eventualmente ri-aprendo il Tool Signal Tap Analyzer) qualora lo si fosse chiuso andando sulla scheda Data e utilizzando le scorciatoie F5 per un’analisi puntuale ed F6 per un’analisi continua saranno disponibili i dati fisicamente letti sull’FPGA sincronizzati come richiesto sul fronte di salita di CS



Si noti in particolare però che i due segnali di hand-shaking a volte vengono campionati, a volte invece non sono rilevati ed anche la loro successione temporale non risulta ben definita. Questo è dovuto al fatto che la frequenza di campionamento scelta è troppo bassa per avere un buon campionamento, d’altro canto aumentare la frequenza avrebbe diminuito il numero di campioni presenti per quanto riguarda il protocollo SPI.

Come ultima operazione si suggerisce di verificare le risorse impiegate per la realizzazione del sistema

* analizzando sia i vari “report log” (in particolare guardando quante ALM, Memorie e registri sono stati impiegati)
* ricorrendo al Tool “Chip Planner”

Paragonando i risultati con quelli ottenuti precedentemente si noterà come siano aumentate considerevolmente le risorse impiegate e come il sistema cominci ad impiegare blocchi di memoria proprio per poter valvare temporaneamente e con continuità i dati analizzati prima di inviarli al tool di visualizzazione ed analisi attraverso il protocollo JTAG.

# Memory Editor

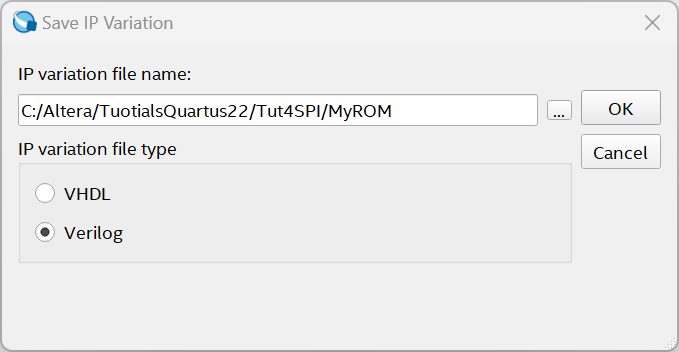
Il progetto fin qui svolto genera una sequenza di segnali SPI ripetitivi in base al contenuto di una memoria e quando si volesse modificare il codice da trasmettere non vi sarebbe altro modo se non quello di modificare il contenuto della memoria e ricompilare l’intero progetto.

Esiste però all’interno di quartus un sistema in grado di interagire attraverso il protocollo JTAG con le memorie scrivendo oppure leggendo il contenuto delle stesse, denominato “In-System Memory Control Editor”. Poiché però questo sistema all’atto pratico trasforma le memorie Single Port in memorie dual port e poi sfrutta una delle due porte per modificare il contenuto, esso deve far uso di specifiche memorie destinate allo scopo. L’intento è pertanto quello di modificare il progetto, integrandovi una memoria di tipo “General Purpose” per poi, interagendo con questa, modificare il segnale trasmesso.

Innanzitutto si acceda alla finestra dell’ IP Catalog (Alt-7) ove esso non sia già aperto e si cerchi, magari facendo uso dei filtri la primitiva ROM: -1PORT



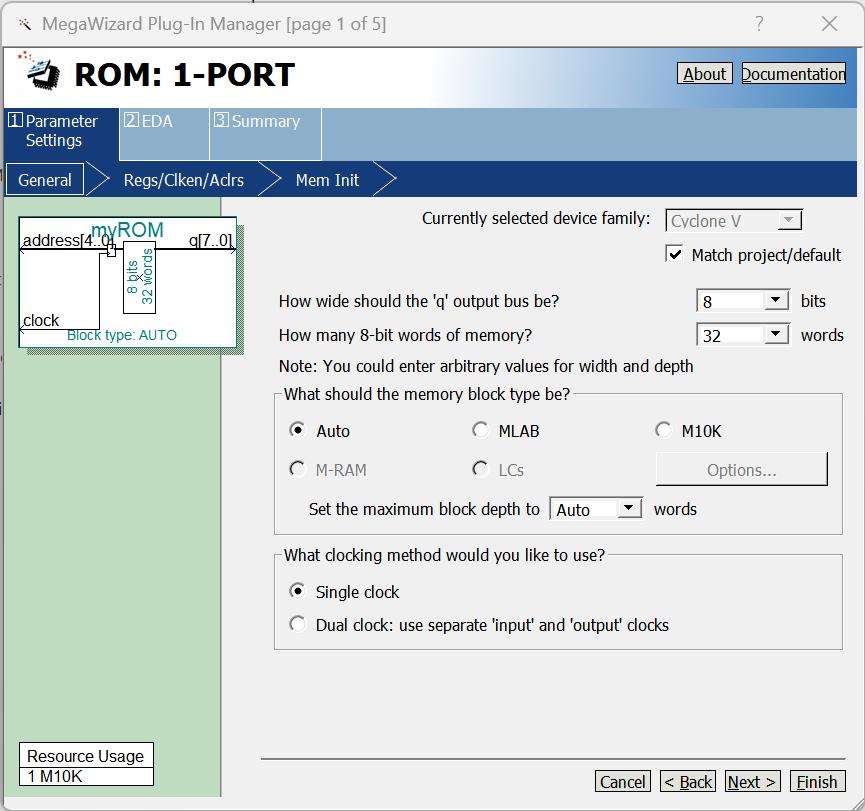
Attraverso un doppio click si apre la finestra di generazione:



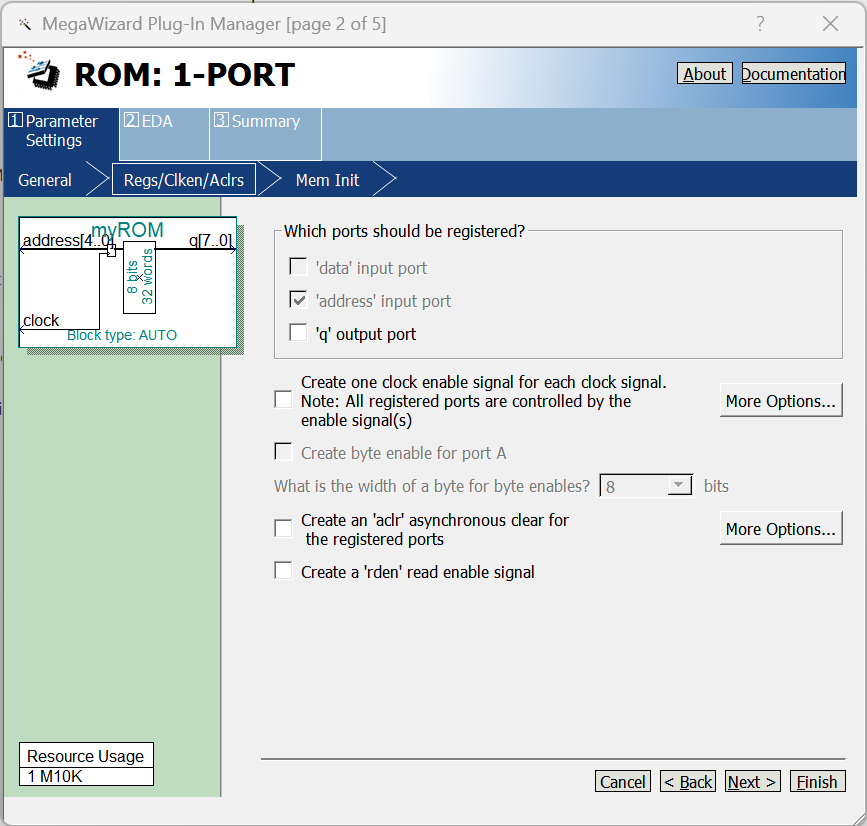
Si fornisca un nome opportuno e si scelga di generarne il codice Verilog

>OK

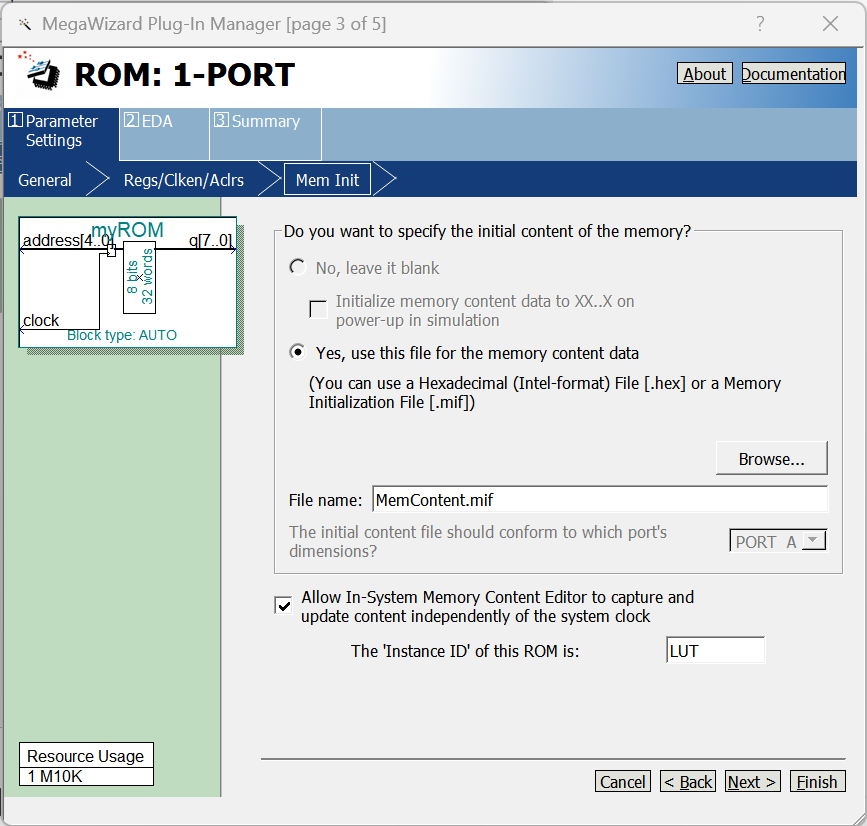
Nelle finestre di configurazione si scelga la dimensione della memoria, in questo caso 32 parole da 8 bit ciascuna (non è previsto l’uso di memorie con meno di 32 parole, pertanto useremo solo le prime 16)



Si Configuri l’uscita per essere priva di registri



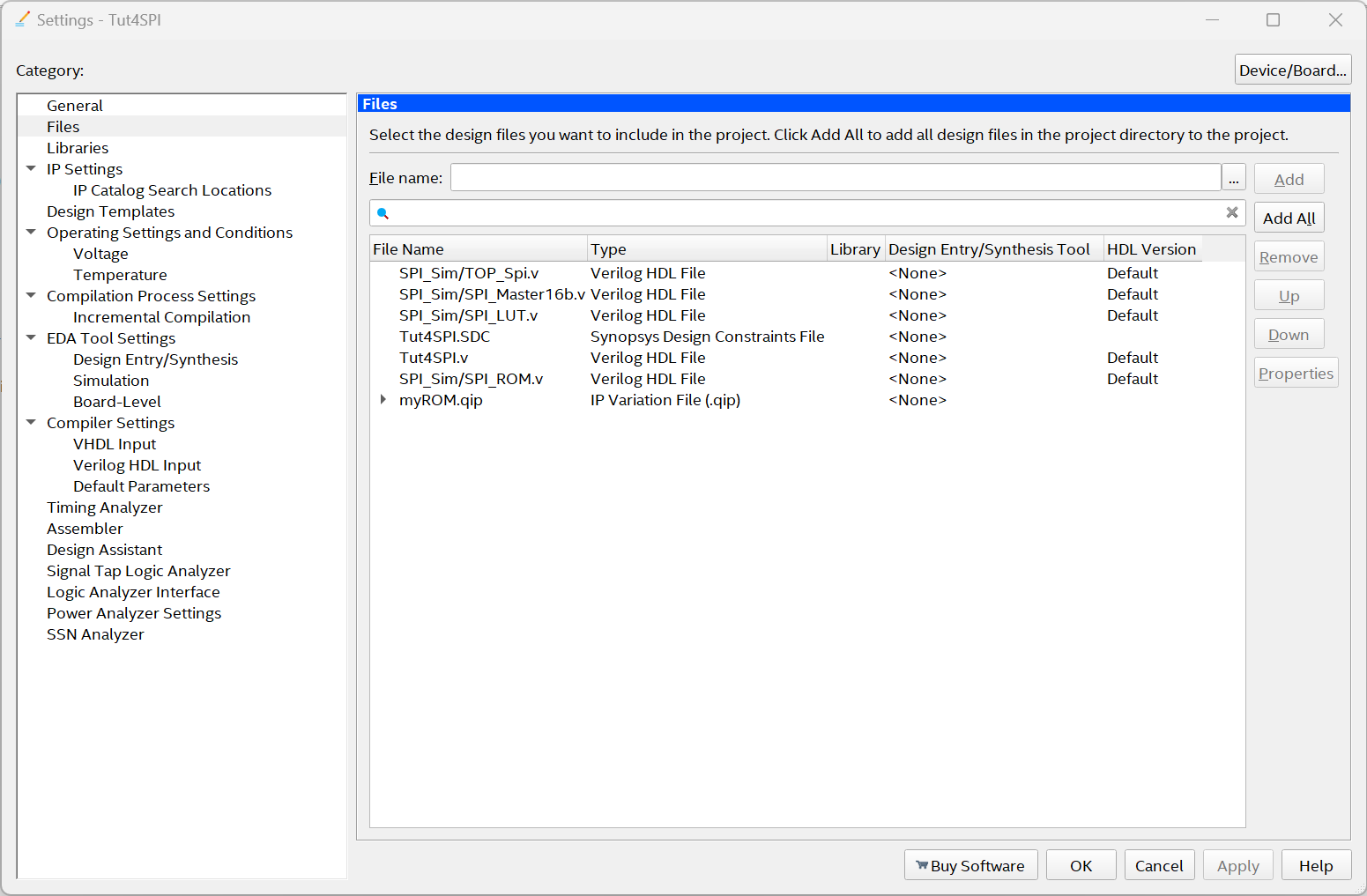
Si Configuri la memoria per essere accessibile al sistema “In-System Memory Control Editor” fornendo un nome mnemonico di 4 caratteri e si fornisca un file di configurazione della memoria (Un esempio è disponibile nella pagina del corso).



> Finish

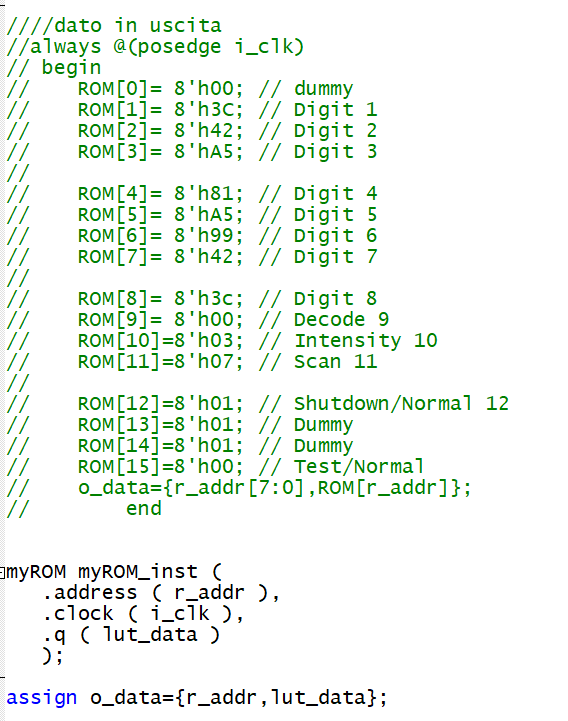
Eventualmente può risultare utile generare anche un esempio di “istanziazione” dell’elemento appena generato.

Ci si assicuri che il file .qip appena generato appartenga al progetto complessivo in cui lo si deve includere.



Si deve ora passare a modificare il progetto per sostituire la LUT realizzata come circuito combinatorio con la ROM appena generata.

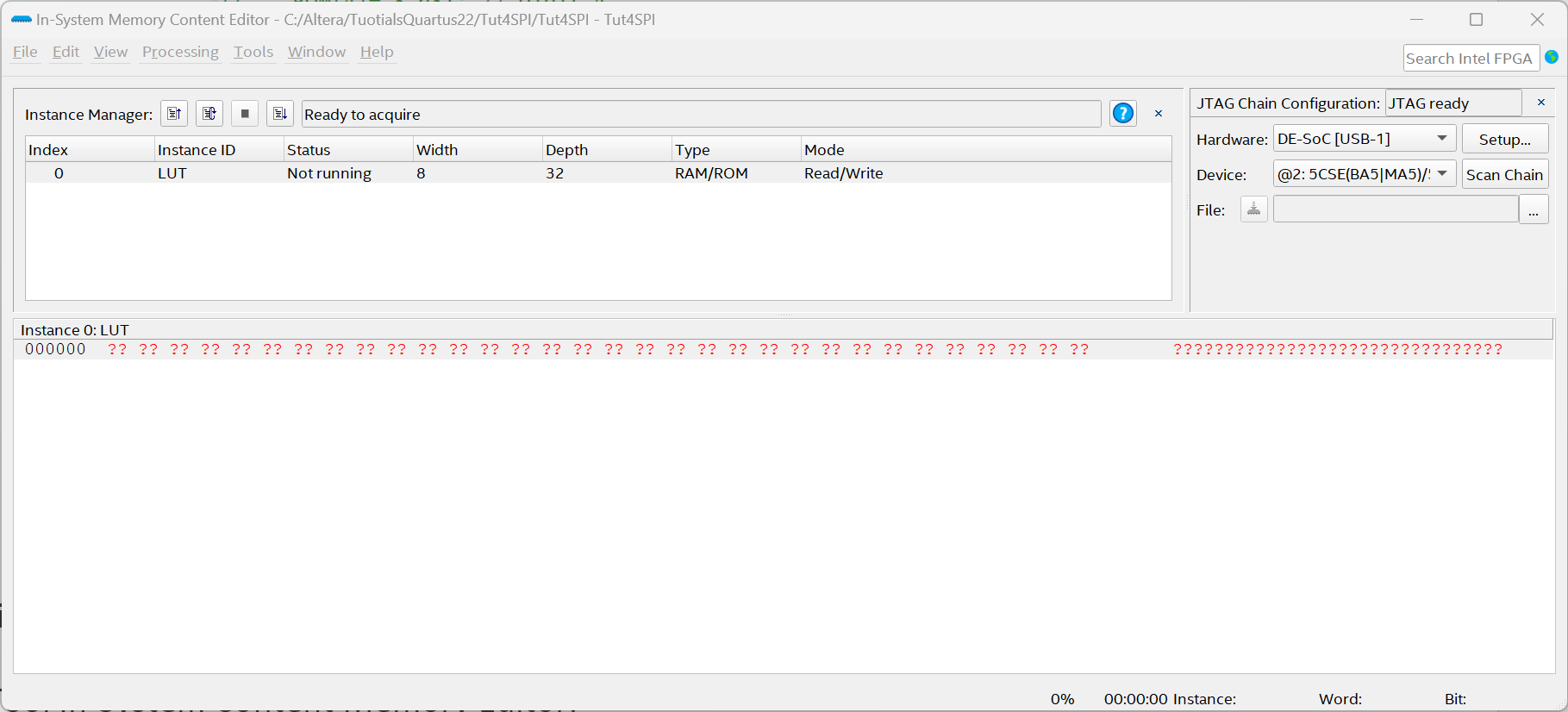
Il file Verilog “SPI\_LUT” per la parte relativa alla memoria dei dati istanziando la ROM appena generata e concatenando l’indirizzo con il dato d’uscita, non prima di aver modificato la tipologia della variabile o\_data da “reg” a “net” e di aver dichiarato il nuovo segnale intermedio “lut\_data”.



Si ricompili quindi l’intero progetto e lo si usi per configurare la Board.

Si può ora aprire il Tool In-System Content Memory Editor:

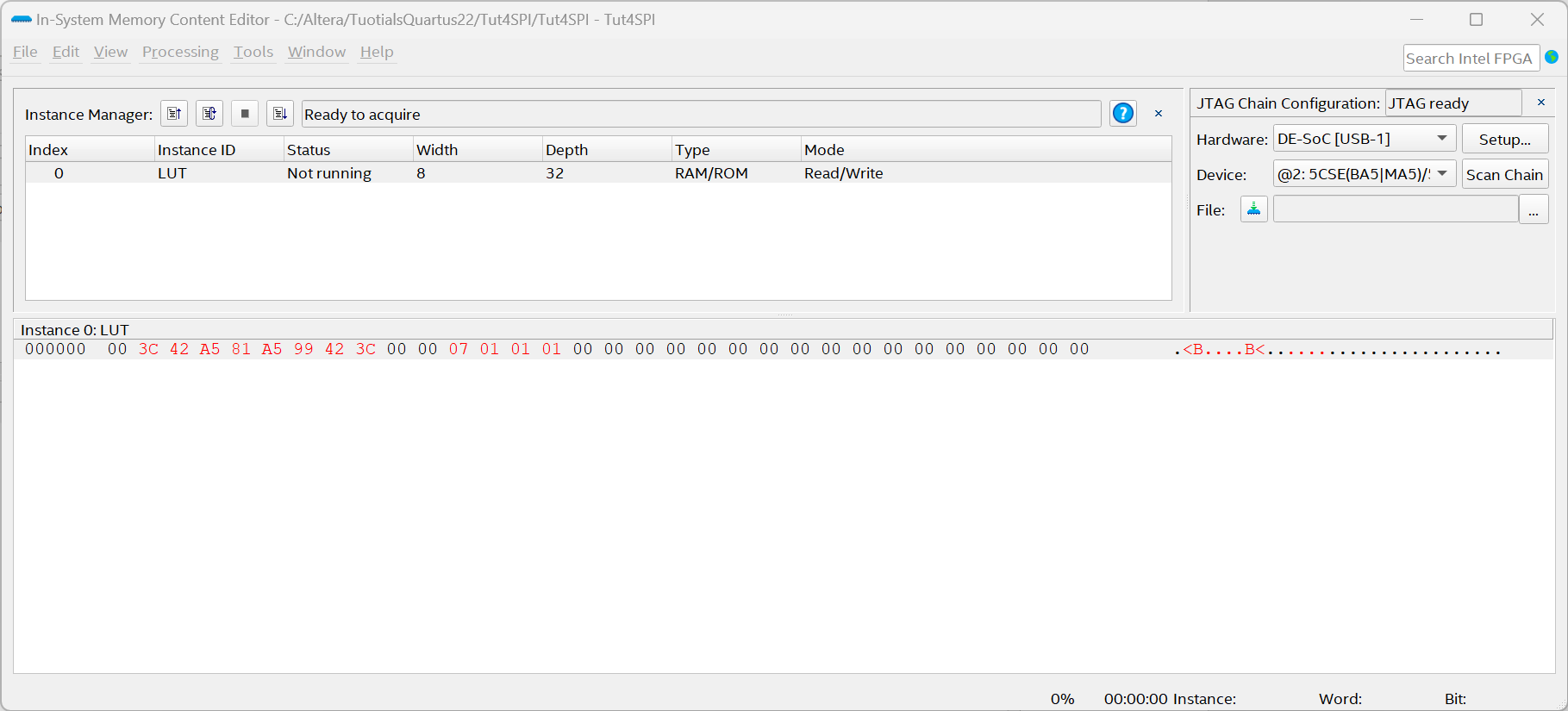
> Tool > In-System Content Memory Editor



Si clicchi sulla opportune “Instance ID” in questo caso solo una e si scelga

> Processing > Read Data From In-System Memory (F5)

E la finestra si popola con i dati ricavati dalla memoria



Oppure si modifichino i dati (Scrivendo dati congrui) e si attivi la scrittura dei dati:

> Processing > Write Data To In-System Memory (F7)

Così facendo si può provare ad alterare

- i dati visualizzati (nelle posizioni [1:8] )   
- la luminosità del display [10]  
- in numero di colonne su cui scrivere i dati [11]  
- la modalità Shutdown/Normal [12]  
- La modalità Normal/Test [14]