

Specifica del progetto

L'obiettivo di questo progetto consiste nell'analisi di LEON, un software processor già esistente e liberamente distribuito sotto le licenze GNU Public Licence (GPL) e Lesser GNU Public Licence (LGPL).

Quella che si ha a disposizione è quindi un'architettura molto più complessa di quella realizzabile da due studenti che si affacciano per la prima volta a questo mondo, e che permette di sfruttare maggiormente le potenzialità degli strumenti a disposizione per compiere operazioni più avanzate.

La prima fase del lavoro prevede un confronto tra LEON ed il software processor proprietario Microblaze, uno dei punti di riferimento per la realizzazione di sistemi embedded mediante i pacchetti software Xilinx. Tale comparazione è inizialmente architetturale e basata prevalentemente su informazioni estrapolate dalla documentazione fornita a corredo dei due prodotti. L'obiettivo è di mettere in evidenza similitudini e differenze tra le due architetture, al fine di valutare la possibilità di utilizzare Microblaze come termine di paragone durante il processo di sintesi di LEON mediante i medesimi strumenti software.

Successivamente il confronto si sposta sul piano delle prestazioni e dell'occupazione fisica di risorse. Essa viene valutata mediante i risultati derivanti da una sintesi dei due software processor sulle FPGA presenti sulle board normalmente utilizzate nel Laboratorio di Microarchitetture. Per procedere con questa sintesi è naturalmente necessario, basandosi sulla comparazione svolta in precedenza, definire a priori una configurazione che renda il più possibile simili le due architetture. In questo modo è possibile valutare la reale convenienza dell'utilizzo di LEON.

LEON

LEON è un processore RISC a 32 bit fedele allo standard SPARC V8. Inizialmente sviluppato in ambito universitario, è ora distribuito dalla società svedese Gaisler Research dal cui sito è liberamente scaricabile l'intero codice sorgente unitamente a vari tool e utility che facilitano il processo di configurazione, sintesi su FPGA e programmazione.

Fra i suoi pregi spicca la completa configurabilità. I vari moduli VHDL possono infatti essere o meno inclusi, e le caratteristiche proprie di ogni componente possono essere personalizzate per mezzo del semplice settaggio di alcuni parametri, rimanendo ovviamente all'interno di opportuni range di valori ammissibili.

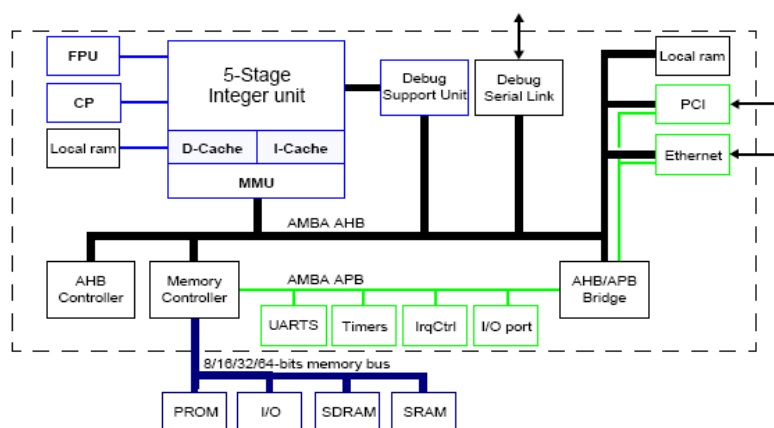


fig.1 – Schema Architeturale di Leon

Microblaze

Il software processor Microblaze appartiene anch'esso alla famiglia RISC a 32 bit. E' di proprietà di Xilinx, ed il suo codice non viene quindi distribuito gratuitamente. Per questo motivo non possiamo disporre del suo modello VHDL ma dobbiamo, per poterne fare uso, partire obbligatoriamente dai software distribuiti dalla stessa: EDK (Embedded Development Kit) e ISE (Integrated Software Environment). Questi strumenti rappresentano anche l'unico modo per configurare e personalizzare Microblaze.

Pur non entrando ora nel merito del confronto vero e proprio è immediato notare come questo processore sia comunque "meno complesso" e come solo alcuni dei moduli siano qui opzionali.

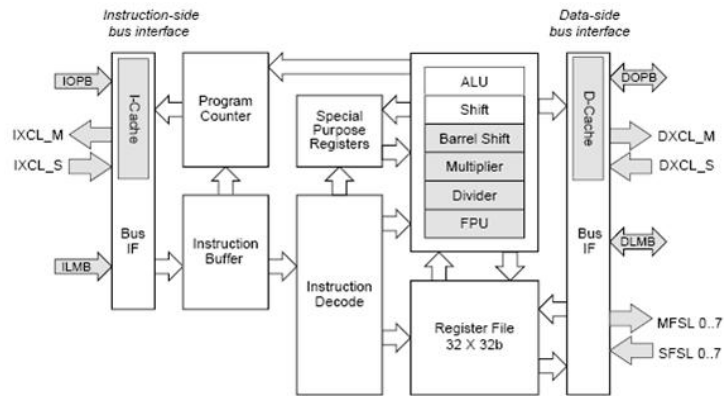


Fig.2 – Schema architetturale di Microblaze (in grigio i moduli opzionali)

LEON vs Microblaze: confronto architetturale

	LEON	Microblaze
<i>Registri general pur pose</i>	2-32 (9)	32 (11)
<i>Dimensioni di dati ed istruzioni</i>	32 bit	32 bit
<i>Integer Unit</i>	comprende istruzioni di moltiplicazione e divisione (9)	istruzioni di moltiplicazione e divisione opzionali
<i>Floating Point Unit</i>	opzionale supporta le FPU proprietarie GRFPU (Gaisler Research), Meiko FPU (Sun), nonché LTH FPU (free, ma attualmente incompleta) (9) supporta singola e doppia precisione	opzionale supporta singola precisione
<i>Instruction Cache e Data Cache</i>	separate (Harvard architecture) multi-associativa con 1 – 4 set da 1 – 64 kbyte/set e 16 – 32 bytes per linea supporta politiche di sostituzione LRU, LRR e casuale (25)	separate (Harvard architecture) mappatura diretta, equivalente alla multi-associativa con un unico set (31)
<i>Memory Management Unit</i>	Opzionale	n.d.
<i>Unità di supporto per il debug</i>	Presente	presente
<i>Pipeline</i>	5 stadi: fetch, decode, execute, memory, write (19)	3 stadi: fetch, decode, execute (26)
<i>Tipologie di BUS</i>	architettura Amba AHB per accesso ai registri interni e APB per il trasferimento dati (33)	interfacce separate per accesso a dati ed istruzioni (Harvard architecture) tre bus supportati: Local Memory Bus (LMB), IBM's On-chip Peripheral Bus (OPB), Xilinx CacheLink (XCL) (41)

tab. 1 – Confronto delle caratteristiche principali.
I numeri fra parentesi indicano la pagina relativa nel manuale di riferimento.

I dati raccolti confermano quanto già accennato. Si osserva subito, infatti, come Microblaze abbia un'architettura più semplificata rispetto a LEON. Inoltre è altrettanto immediato osservare come, al di là di alcune particolarità, i possibili gradi di libertà della configurazione siano minori.

Scendendo a un livello di dettaglio maggiore si possono inoltre focalizzare le seguenti differenze chiave:

- **FPU**

In LEON l'uso di una FPU porta con sé alcune problematiche e non può quindi dirsi essere immediato.

In primo luogo, per essa, non viene fornito nessun soft-core. E' prevista solamente (strettamente in linea con quanto previsto dallo standard SPARC V8) un'interfaccia a cui collegare un componente esterno che svolga tale funzione. Per utilizzare una qualunque delle FPU supportate è quindi innanzitutto necessario procurarsi il relativo codice (occorre a questo proposito considerare che nessuna di queste è distribuita con licenza di tipo free) e successivamente provvedere a interfacciare il nuovo componente in maniera corretta.

Sotto questo aspetto Microblaze si dimostra invece molto immediato. L'FPU è infatti stata introdotta con l'attuale versione del processore (v4.00a) e, sebbene opzionale, risulta essere quindi direttamente disponibile.

- **CACHE**

L'organizzazione della memoria in entrambi i processori segue lo schema Harvard. Esso prevede un completo sdoppiamento, sia logico che fisico, della parte istruzioni rispetto alla parte dati. Nella pratica questo sdoppiamento si riflette su spazio indirizzi, bus e controller di memoria.

La cache di LEON può presumibilmente raggiungere livelli di efficienza maggiore essendo di tipo multi-associativo. In questo tipo di cache a ogni indirizzo di memoria non viene infatti associato direttamente un corrispondente indirizzo di cache ma bensì un corrispondente set di indirizzi, dalle dimensioni più o meno ampie. Il dato o l'istruzione in questione può quindi essere dinamicamente fatto corrispondere a una qualunque delle linee del set. La gestione di questo tipo di cache è ovviamente maggiormente complessa dovendo, in ogni linea, essere memorizzato un indicatore che consenta di risalire a quale sia, fra i vari possibili, il reale dato contenuto. Su questo punto LEON non si discosta dal metodo comune. Nella linea di cache è infatti presente un campo definito ATAG che riporta la parte di indirizzo necessaria. Le dimensioni di questo campo sono automaticamente settate secondo quanto richiesto dagli altri parametri. L'altra problematica che si pone nell'uso di questo tipo di cache è la scelta della linea da sostituire quando occorre inserire un dato in un set già completamente pieno. LEON supporta tutte le principali politiche di sostituzione mantenendo però un unico formato dati. I campi necessari alle politiche LRR e LRU vengono, negli altri casi semplicemente ignorati.

Microblaze permette invece il solo uso di una semplice cache a mappatura diretta. E' bene anticipare che questo non costituisce un problema per le fasi successive del nostro lavoro essendo quest'ultima equivalente a un che multi-associativa con numero di set pari a uno.

- **UNITA' di SUPPORTO per il DEBUG**

Entrambi i processori presentano soluzioni per facilitare il debug di configurazioni e applicazioni.

LEON prevede la possibilità di inserire un'unità hardware (DSU – Debug Support Unit) attraverso la quale è possibile gestire il processore mentre questo si trova in “debug mode”.

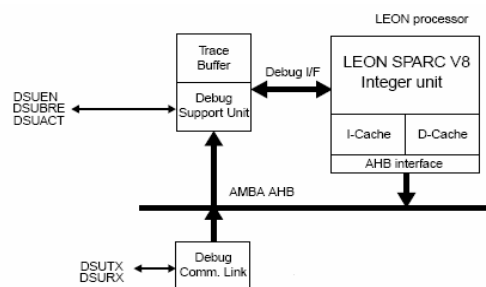


Fig.3 – Unità DSU, segnali esterni e collegamenti

Il passaggio a questa modalità può avvenire in vari casi settabili attraverso un registro di controllo proprio della DSU e alcuni segnali esterni. Fra di essi va sottolineata l'esecuzione di un'istruzione di breakpoint e il sollevamento di un'eccezione che porti il processore in modalità d'errore.

Una volta entrati in questa modalità si dispone di un accesso di tipo read/write incondizionato sulla cache e su tutti i registri del processore.

Inoltre la DSU dispone di un “trace buffer” organizzato circolarmente e avente dimensioni variabili in cui viene memorizzata una sorta di cronologia delle istruzioni eseguite e delle istruzioni trasferite sull' AHB bus. L'accesso ai dati memorizzati dalla DSU avviene attraverso un'interfaccia seriale.

Microblaze utilizza una strategia equivalente. In alternativa permette di realizzare, per la funzione di debug, un modulo software. Questa alternativa viene però sconsigliata in quanto molto penalizzante in termini di risorse e prestazioni.

• PIPELINE

A prima vista questo è il punto su cui si concentrano le maggiori differenze e che potrebbe quindi creare dei problemi a livello di confronto.

LEON utilizza una pipeline a 5 stadi suddividendo le operazioni in questo modo:

- Fetch*: carica l'istruzione dalla cache se questa è presente, viceversa la richiesta è inoltrata al controller della memoria.
- Decode*: l'istruzione, che risulta essere completamente disponibile al termine del ciclo precedente, viene ora decodificata. Vengono inoltre letti gli operandi necessari.
- Execute*: vengono eseguite le operazioni che coinvolgono ALU, shifter e confronti logici. Per le operazioni di accesso alla memoria o di salto vengono generati gli indirizzi necessari.
- Memory*: in questa fase la cache viene letta o scritta.
- Write*: i risultati delle operazioni che coinvolgono ALU, shifter e i dati eventualmente provenienti dalla cache vengono memorizzati negli opportuni registri.

Microblaze dispone invece di una pipeline che è suddivisa in tre sole fasi:

- Fetch*
- Decode*
- Execute*

Per quanto riguarda le prime due il significato è simile a quanto già detto per LEON. Execute racchiude invece al suo interno, evidentemente, anche ciò che LEON suddivide in Memory e Write.

Tuttavia non si ritiene sia il caso di focalizzarsi eccessivamente su questo aspetto: la pipeline è infatti uno degli elementi più critici nel determinare l'efficienza di un'architettura. E' quindi logico aspettarsi differenze del genere proprio per la necessità di ottimizzarne l'organizzazione, plasmandola sulle caratteristiche e sulle temporizzazioni proprie di ogni singolo processore.

Si ritiene invece sia più utile concentrarsi sui risultati che le due permettono di ottenere. In particolare, secondo quanto dichiarato, in entrambi i casi è possibile, a regime e in assenza di salti, concludere un'operazione per ciclo. Viene inoltre specificato come la maggior parte delle operazioni richiedano un'unica iterazione per poter essere eseguite e che, nei pochi casi in cui ciò non risulta essere vero, si procede in entrambi i casi aggiungendo cicli di stallo. Infine, fra le istruzioni che più intaccano l'efficienza della pipeline, vi sono ovviamente gli accessi alla memoria. Microblaze prevede l'uso di una coda di prefetch (ovvero un'area in cui, mentre la pipeline è bloccata nell'esecuzione di un'istruzione che necessita di più di un ciclo, vengono caricate sequenzialmente le operazioni successive facendo sì che la seguente fase di fetch possa accedervi velocemente senza dover attendere i tempi richiesti da un accesso in memoria) per ridurre questo problema. Al di là di questa differenza si può quindi, per quanto precedentemente detto, considerare le due pipeline perlomeno "funzionalmente" equivalenti.

• BUS

Riguardo ai bus vi sono alcune differenze che risultano essere immediatamente evidenti.

LEON utilizza bus di tipo AMBA. Questo standard si pone, tra gli altri, obiettivi quali l'essere indipendente da qualsiasi tecnologia specifica e l'idoneità allo sviluppo di design modulari. Ciò ricalca in pieno le caratteristiche di LEON e quindi ha avuto sicuramente peso nella scelta. Sono utilizzati sia il bus di sistema sia il bus periferiche. Per il primo è stato scelto il tipo AHB (AMBA prevede anche un'altra possibilità: ASB) che meglio si adatta quando sono necessarie prestazioni particolarmente elevate.

Microblaze prevede invece più opzioni. In particolare il bus di tipo LMB permette di garantire un accesso agli on-chip RAM block in un solo ciclo e di avere una gestione semplificata dovuta all'uso di un ridotto numero di segnali di controllo. OPB (sviluppato da IBM) permette invece l'accesso alle periferiche, siano esse on-chip o off-chip. La vera differenza sta però nel potere utilizzare un tipo di collegamento particolare per l'accesso alle memorie esterne. Mediante due interfacce FSL (le quali forniscono una connessione

punto-punto con logica FIFO) è possibile utilizzare una soluzione che garantisce performance elevate denominata XCL – Xilinx Cache Link.

LEON vs Microblaze: confronto su FPGA

Il confronto appena concluso conferma come i due software processor in esame siano sufficientemente simili per poter essere tra loro paragonati anche a livello di sintesi. In questa fase si è scelto di iniziare con la sintesi di Microblaze, essendo dei due quello che offre minori possibilità di personalizzazione. Non vi saranno quindi problemi a settare i parametri di LEON al fine di renderlo il più possibile aderente alla medesima configurazione.

La fase di sintesi, che verrà ora descritta, ha come scopo quello di determinare l'occupazione in termini di area su due FPGA scelte come riferimento, entrambe appartenenti all'architettura Virtex2Pro prodotta da Xilinx. Le loro sigle sono:

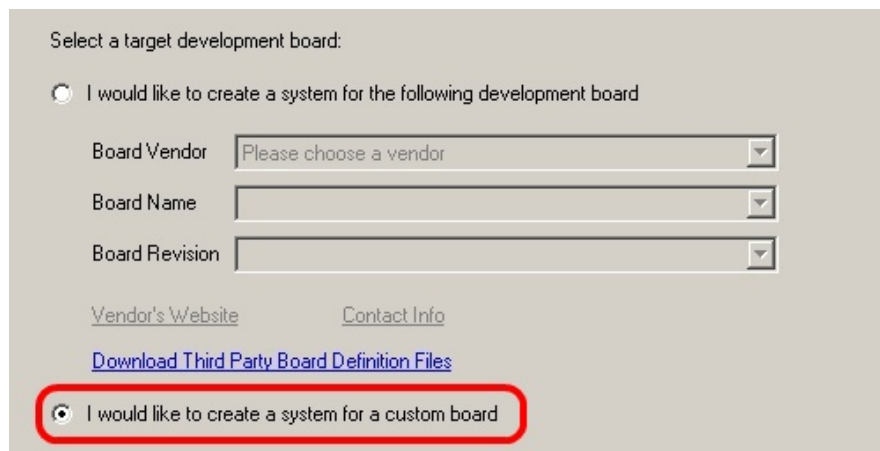
- xc2vp7-fg456-6
- xc2vp20-ff896-5

Al di là della scelta della specifica FPGA, il processo di sintesi risulta essere identico. Di seguito si farà riferimento a quello relativo alla xc2vp7-fg456-6.

Sintesi di Microblaze

La prima parte della sintesi è svolta utilizzando *EDK*, software che prevede una procedura guidata che permette di arrivare alla creazione del *Microprocessor Hardware Specification* file (MHS). Questo file contiene le indicazioni relative al processore scelto (nel nostro caso, Microblaze), al bus utilizzato, ed agli indirizzi associati alle periferiche connesse a tale bus. Questo file, quindi, risulta essere una descrizione completa del sistema embedded realizzato.

La procedura guidata prende il nome di *Base System Builder* e consente, in prima istanza, di creare un nuovo design su una delle board fornite da Xilinx. Per poter utilizzare le FPGA sopra citate, tuttavia, è necessario definire una board personalizzata, per la quale è possibile specificare architettura, size, package e speedgrade.



Select a target development board:

I would like to create a system for the following development board

Board Vendor

Board Name

Board Revision

[Vendor's Website](#) [Contact Info](#)

[Download Third Party Board Definition Files](#)

I would like to create a system for a custom board

fig.4 – Scelta della custom board

La scelta di utilizzare una custom board implica il dover specificare, nelle fasi successive, i device realmente presenti sulla scheda fisica. Inoltre, al fine di poter effettivamente realizzare il processore, è necessario modificare il file UCF, contenente le informazioni sull'assegnamento dei pin.

Si passa quindi a selezionare i parametri che caratterizzano l'FPGA, specificando Microblaze quale processore da utilizzare.

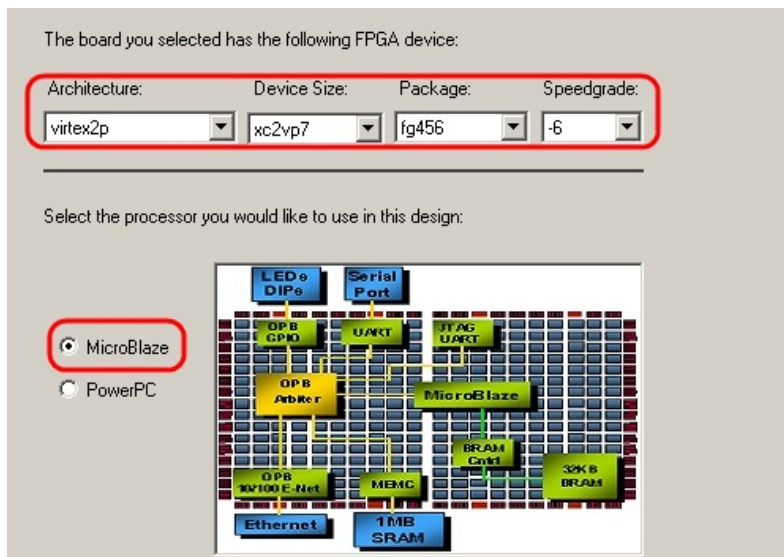


fig.5 – Specifica della FPGA e scelta di Microblaze come processore

Fase successiva è la configurazione del processore Microblaze. L'unico parametro che merita particolare attenzione risulta essere la definizione della memoria cache. Nell'ottica del confronto si è scelto di equipaggiare sia Microblaze sia LEON con 8KB di instruction e data cache. Il bus scelto per la trasmissione dei dati è l'*On-chip Peripheral Bus (OPB)*, in quanto il *CacheLink* fornito da Xilinx è studiato per controller specifici e non è quindi adatto a questo tipo di comparazione. I rimanenti parametri sono lasciati inalterati, compresa la presenza dell'unità di supporto per il debug a livello hardware.

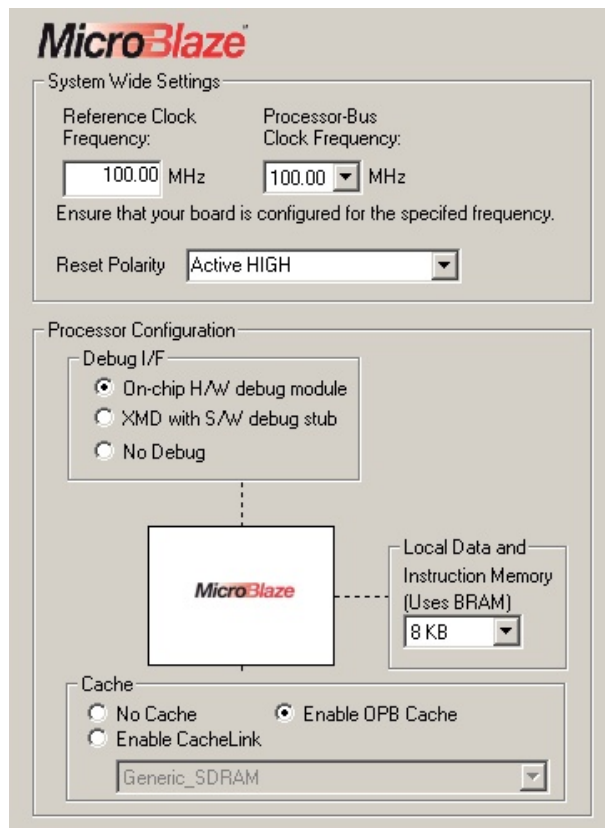


fig.5 – Configurazione di Microblaze

Successivamente, a causa della scelta di utilizzare una board personalizzata, andrebbero specificati i dispositivi di Input/Output e le varie periferiche presenti sulla scheda. Dal momento che la sintesi si concentra principalmente sul processore in sé, e non sui device a cui esso si interfaccia, in questa fase non verrà aggiunto tutto quanto viene offerto dalla board reale, ma solo ciò che è strettamente necessario per consentire la sintesi. Pertanto, l'unico dispositivo I/O inserito è un'interfaccia UART che controlla una porta seriale RS232, a cui è assegnato il ruolo di input/output standard.

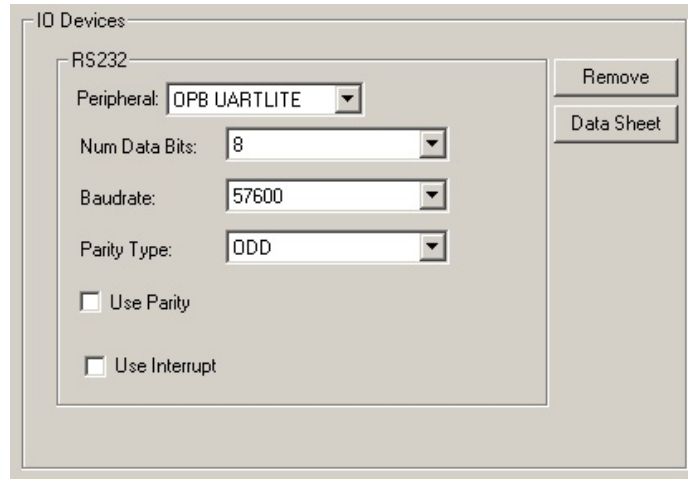
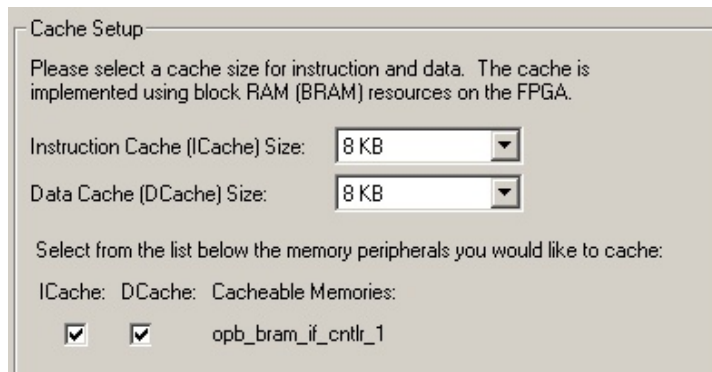
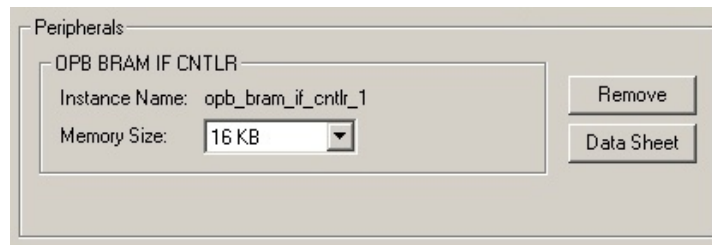


fig.6 – Configurazione dell'I/O

Inoltre occorre inserire, tra le periferiche del sistema, uno o più controller di memoria BRAM sul quale verranno mappate le cache dei dati e delle istruzioni, da 8KB cadauna. In questo caso, due sono le possibili scelte: la prima prevede un unico controller di un blocco di memoria da 16KB, equamente suddiviso tra le due cache, mentre la seconda prevede l'utilizzo di due controller separati. Dopo aver svolto dei test con entrambe le soluzioni, la soluzione con un unico controllore è risultata la più efficiente in termini di area occupata, anche se la differenza tra le due risulta minima. Dunque, l'alternativa adottata è la prima.



figg.7-8 – Configurazione delle periferiche e mappatura della cache

Infine, per completare la procedura guidata, è possibile ignorare la possibilità di includere un software predefinito per testare il funzionamento della memoria, chiamato *Memory Test*. L'aggiunta di software infatti, sebbene essenziale per un sistema di tipo embedded, non ha al momento alcuna rilevanza.

Si giunge dunque al termine della configurazione guidata, i cui risultati sono riassunti in una schermata simile a quella illustrata nella figura seguente.

Processor: Microblaze
 System clock frequency: 100.000000 MHz
 Debug interface: On-Chip HW Debug Module
 Data Cache: 8 KB
 Instruction Cache: 8 KB
 On Chip Memory : 24 KB

The address maps below have been automatically assigned. You can modify them using the editing features of XPS.

OPB Bus : OPB_V20 Inst. name: mb_opb Attached Components:			
Core Name	Instance Name	Base Addr	High Addr
opb_mdm	debug_module	0x41400000	0x4140FFFF
opb_uartlite	RS232	0x40600000	0x4060FFFF
opb_bram_if_cntrlr	opb_bram_if_cntrlr_1	0x00000000	0x00003FFF

LMB Bus : LMB_V10 Inst. name: ilmb Attached Components:			
Core Name	Instance Name	Base Addr	High Addr
lmb_bram_if_cntrlr	ilmb_cntrlr	0x20002000	0x20003FFF

LMB Bus : LMB_V10 Inst. name: dlmb Attached Components:			
Core Name	Instance Name	Base Addr	High Addr
lmb_bram_if_cntrlr	dlmb_cntrlr	0x20002000	0x20003FFF

fig.9 – Riassunto della configurazione di Microblaze

Gli indirizzi di tutti i dispositivi interfacciati al bus sono calcolati automaticamente dalla procedura guidata, e dunque non è necessario impostarli manualmente. Pertanto, è ora possibile generare il file MHS che descrive la configurazione prescelta, e che a questo punto deve essere trattato dal programma che ha la possibilità di implementare quanto definito finora: *ISE*.

Prima di passare a ciò può però essere interessante, al fine di capire meglio quanto fatto, visualizzare la schermata *add/edit cores* dove sono visibili le componenti inserite e i relativi collegamenti ai bus.

Peripherals Bus Connections Addresses Ports Parameters

Click on squares to make master, slave or master-slave (M, S, MS) connections. Right click on any bus instance (column header) for a context menu.

	OPB_V20	LMB_V10	LMB_V10
microblaze_0 dlmb			M
microblaze_0 ilmb		M	
microblaze_0 dopb	M		
microblaze_0 iopb	M		
debug_module sopb	S		
debug_module sfs0			
debug_module mfs0			
dlmb_cntrlr slmb			S
ilmb_cntrlr slmb		S	
RS232 sopb	S		
opb_bram_if_cntrlr_1 sopb	S		

Choose one or more buses (use Shift or Ctrl). Click Add.

<< Add

Choose the BRAM port to connect to the controller port. Give a name to the connection.

Cntrl Port	BRAM Port	Connector
dlmb_cntrlr bram_...	lmb_bram PORTB	dlmb_port
ilmb_cntrlr bram_p...	lmb_bram PORTA	ilmb_port
opb_bram_if_cntrlr...	opb_bram_if_cntrlr...	opb_bram...

Other Transparent bus (point to point) connections

Source	Destination	Connector
--------	-------------	-----------

fig.10 – Interfacciamento periferiche / bus

Una volta aperto *Project Navigator*, il tool di sintesi *ISE*, occorre creare un nuovo progetto nel quale i parametri relativi alla FPGA utilizzata siano compatibili con quelli utilizzati con *EDK*.

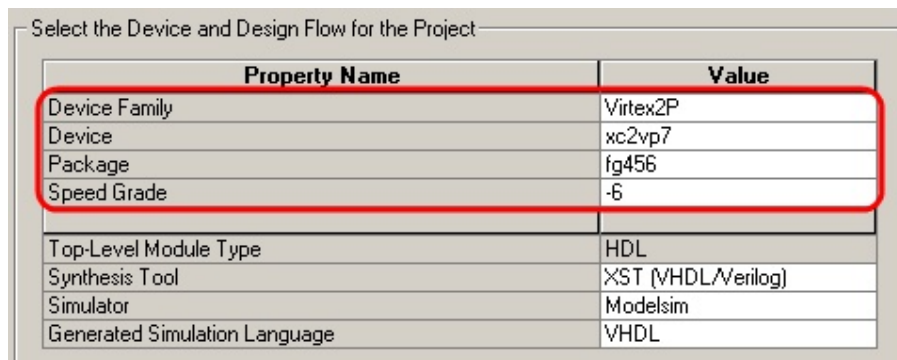


fig.11 – Definizione della FPGA compatibilmente a quanto fatto con EDK (cfr. fig.5)

Nelle fasi successive della creazione del progetto non occorre aggiungere alcun codice sorgente, ma arrivare al termine della procedura guidata senza compiere alcuna operazione. Dalla schermata principale del *Project Navigator* è ora possibile aggiungere un nuovo codice sorgente: tale codice è il file con estensione *.xmp* che si riferisce al progetto creato in precedenza con *EDK*. A questo punto, tale file va istanziato utilizzando *View HDL Instantiation Template*, che genera un file VHDL chiamato *system_stub.vhd*, a cui sono associati tutti i successivi processi utilizzati da ISE.

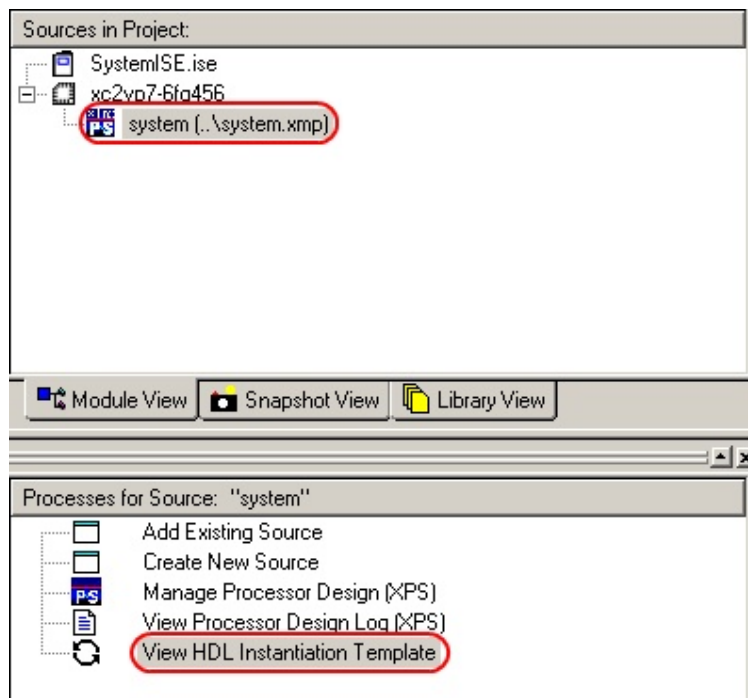


fig.12 – Importazione del file *.xmp* e creazione del codice VHDL corrispondente

Una volta creato, il file contenente lo stub va importato tra i sorgenti del progetto. Inoltre, al fine di consentire al tool di eseguire correttamente la sintesi, è necessario associare a tale sorgente una descrizione dei pin della FPGA. Essa è inclusa nel file con estensione *.ucf* contenuto nella cartella *data*, creata da *EDK*. Le informazioni contenute nel file UCF non sarebbero adeguate ad

implementare fisicamente il tutto su una board, ma sono comunque sufficienti per generare i dati di interesse in questa analisi.

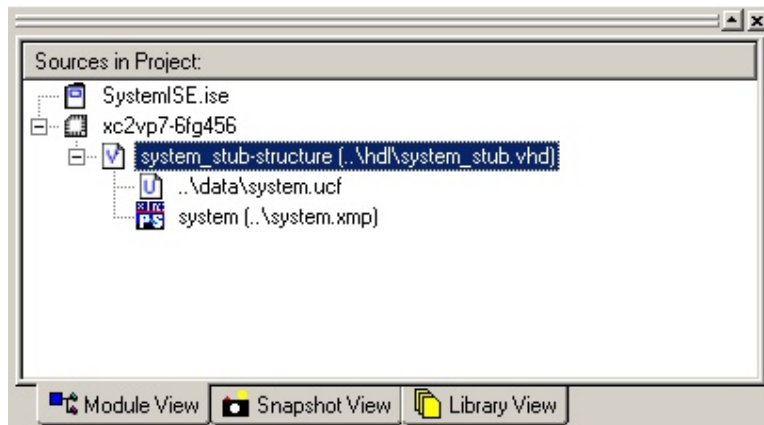


fig.13 – Stato del progetto dopo l’inserimento del codice VHDL e del file UCF ad esso associato

Per terminare la procedura ed ottenere i risultati della simulazione occorre ora eseguire tutti i processi disponibili per il file VHDL importato al passo precedente, fino ad arrivare a *Generate Programming File*. Al termine dell’esecuzione, un file in formato HTML contenente i dati sull’occupazione è rintracciabile nella cartella del progetto.

Eseguita la procedura descritta con entrambe le FPGA scelte come riferimento, sono stati ottenuti i risultati di seguito riportati.

Virtex2P xc2vp7-fg456-6

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	843	9,856	8%	
Number of 4 input LUTs:	1,429	9,856	14%	
Logic Distribution:				
Number of occupied Slices:	1,156	4,928	23%	
Number of Slices containing only related logic:	1,156	1,156	100%	
Number of Slices containing unrelated logic:	0	1,156	0%	
Total Number 4 input LUTs:	1,864	9,856	18%	
Number used as logic:	1,429			
Number used as a route-thru:	17			
Number used for Dual Port RAMs:	256			
Number used as Shift registers:	162			
Number of bonded IOBs:	5	248	2%	
Number of PPC405s:	0	1	0%	
Number of Block RAMs:	22	44	50%	
Number of MULT18X18s:	3	44	6%	
Number of GCLKs:	2	16	12%	
Number of BSCANs:	1	1	100%	

Number of GTs:	0	8	0%	
Number of GT10s:	0	0	0%	
Number of RPM macros:	5			

Virtex2P xc2vp20-ff896-5

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	843	18,560	4%	
Number of 4 input LUTs:	1,430	18,560	7%	
Logic Distribution:				
Number of occupied Slices:	1,156	9,280	12%	
Number of Slices containing only related logic:	1,156	1,156	100%	
Number of Slices containing unrelated logic:	0	1,156	0%	
Total Number 4 input LUTs:	1,865	18,560	10%	
Number used as logic:	1,430			
Number used as a route-thru:	17			
Number used for Dual Port RAMs:	256			
Number used as Shift registers:	162			
Number of bonded IOBs:	5	556	1%	
Number of PPC405s:	0	2	0%	
Number of Block RAMs:	22	88	25%	
Number of MULT18X18s:	3	88	3%	
Number of GCLKs:	2	16	12%	
Number of BSCANs:	1	1	100%	
Number of GTs:	0	8	0%	
Number of GT10s:	0	0	0%	
Number of RPM macros:	5			

Documentazione consultata

- [1] Leon2-1.0.30-xst User Manual – Gaisler Research
- [2] The SPARC Architecture Manual (v8) - SPARC International Inc.
- [3] Microblaze Processor Reference Guide – Xilinx EDK v.7.1
- [4] Amba Specification (v 2.0)