Corso FPGA

Nate alcune decine d'anni fa, rappresentano un'ottima soluzione nelle applicazioni in cui serva svolgere funzioni logiche programmabili. Impariamo a conoscerle.

di
MARIANO SEVERI

FPGA è l'acronimo di *Field Programmable Gate Array*, ovvero, letteralmente, un insieme di porte logiche programmabili; niente è più adatto a dire che un chip FPGA è una vera e propria matrice di elementi più o meno complessi che possono essere configurati per realizzare le funzioni logiche desiderate. Le stesse interconnessioni tra gli elementi possono essere programmate per creare strutture a più alto livello e connettere tra loro i diversi circuiti. Da un certo punto di vista, quindi, le FPGA ereditano il concetto di riprogrammabilità proprio del mondo dei microcontrollori: sebbene l'architettura sia la stessa, le funzionalità implementate variano a seconda della particolare applicazione. D'altra parte, però, rifanno il verso agli ASIC nella misura in cui queste stesse funzionalità sono realizzate per via hardware mediante l'interconnessione di porte logiche e non eseguendo un qualche codice applicativo. In queste pagine inizia un corso che vi introdurrà alla conoscenza e all'uso del chip FPGA; in questa prima puntata iniziamo da una breve storia dell'evoluzio-

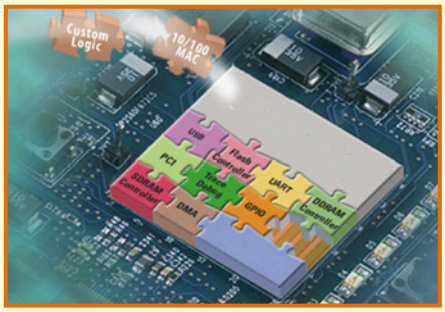


Fig. 1 - *FPGA, programmabilità hardware.*

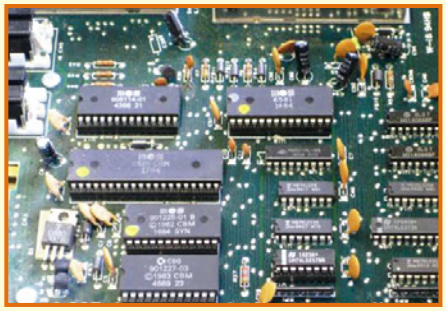


Fig. 2 - *Dispositivi FPLA nei primi calcolatori.*

ne dei dispositivi logici programmabili, dalle prime PLD alle moderne FPGA; proseguiremo poi con una panoramica sulle principali tecnologie ed architetture oggi disponibili. Infine tracceremo il metodo di progetto che interessa questi dispositivi.

UN PO' DI STORIA

I componenti *FPGA* sono stati introdotti nella seconda metà degli anni '80 del secolo scorso come terza generazione di *PLD* (*Programmable Logic Device*); i PLD erano nati a loro volta nella seconda metà degli anni '60, con lo sviluppo presso i laboratori della *Harris Semiconductor* di una '*matrice di diodi configurabile*'. Tale matrice consentiva di realizzare semplici funzioni logiche bruciando le connessioni tra i diodi, realizzate da piccolissimi fusibili che in fase di programmazione venivano interrotti facendovi scorrere una corrente ben più elevata di quella di normale funzionamento. Il problema di tali dispositivi è che la matrice aveva dimensioni limitate; inoltre non erano disponibili programmatori reperibili in commercio e quindi l'utente non poteva programmare da sé le PLD. La programmazione veniva fatta in fabbrica direttamente alla Harris sulla base delle maschere fornite dall'utente, quindi risultava possibile solo avere chip customizzati e non comperare PLD da customizzare.

In principio erano le ROM

Nel 1969 i ricercatori della *IBM* svilupparono la memoria *ROAM* (*Read Only Associative Memory*) che consisteva in una matrice di porte logiche, invece che di diodi, unite da interconnessioni programmabili. Nel 1970 la *Texas Instrument* realizzò il primo circuito integrato basato interamente su *ROAM*, coniando il termine *PLA* (*Programmable Logic Array*); il *TMS2000*, *ad esempio*, aveva 17 ingressi, 18 uscite ed 8 *flip-flop* di tipo *J-K* . Gli elevati costi di produzione non ricorrenti che l'utente doveva affrontare non favorirono, tuttavia, la diffusione di questo tipo di dispositivi. Più o meno contemporaneamente, la *Harris* introdusse nel mercato le *PROM* (*Programmable Read Only Memory*); oltre che come memorie a sola lettura, questi dispositivi trovarono impiego nella realizzazione di semplici macchine a stati o decodificatori di indirizzo. Era ormai chiara e sempre più pressante l'esigenza dell'industria di avere dispositivi con componenti programmabili a costi contenuti, con i quali realizzare funzionalità logiche specifiche per le varie applicazioni.

Nel 1973 *National Semiconductor* produsse una versione semplificata di componenti *PLA*, denominata *DM7575/DM8575*, caratterizzata da soli 14 ingressi ed 8 uscite e senza elementi di memoria integrati. Questa costituì il prototipo dei dispositivi *FPLA* (*Field Programmable Logic Array*) introdotti in seguito dalla *Intersil* e dalla *Signetics*. Nel giugno del 1975 la *Intersil* annunciò la produzione degli IM5200, che introducevano una nuova tecnologia, definita *AIM* (*Avalanche Induced Migration*), per la realizzazione delle logiche programmabili; questa nuova tecnologia si basava su celle nelle quali come elemento programmabile veniva usato un transistor: inducendo una corrente molto elevata tra l'emettitore ed il collettore si produceva un cortocircuito della giunzione base-emettitore, che configura il transistor come un semplice diodo.

Sfortunatamente il nuovo dispositivo non si rivelò molto affidabile e non ebbe diffusione commerciale significativa; una sorte migliore toccò, invece, ai componenti *82S100 FPLA* prodotti dalla *Signetics*, grazie anche all'intensa campagna di promozione e sostegno attuata. I dispositivi *82S100* utilizzavano la vecchia tecnologia delle connessioni fusibili ed erano integrati in un *package DIP* (*Dual Inline Packa-*

ge) a 28 pin di dimensioni intorno ai 15 cm; tuttavia, come tutti i dispositivi *FPLA*, avevano prestazioni limitate per quel che riguardava la velocità di lavoro, erano ancora costosi e ritenuti non facilmente collaudabili. Inoltre le funzioni logiche che si intendeva realizzare in questi dispositivi dovevano essere convertite in uno standard tabulare definito *H&L*, poco familiare ai progettisti.

Fig. 3 - *H. T. Chua e J. Birkner realizzano nel 1978 i primi dispositivi PAL.*

La nascita degli array programmabili

Una prima significativa svolta si ebbe nel 1978, con l'introduzione nel mercato dei primi componenti *PAL* (Programmable Array Logic) ad opera della *Monolithic Memories, Inc.* La *MMI* veniva da una lunga collaborazione con il *General Electric Research and Development Center* durante la quale aveva sviluppato a fini di ricerca dei dispositivi riprogrammabili denominati *PALA* (*Programmable Associative Logic Array*) basati sulla tecnologia delle *PROM* riscrivibili introdotta nel 1971 dalla Intel. Il successo delle *PAL* fu determinato anche dalla definizione di un linguaggio di programmazione, denominato *PALASM* (*PALA ASseMbler*), simile al *FORTRAN*, che consentiva di descrivere funzioni logiche complesse. Il linguaggio, per la sua semplicità, fu adottato da diverse compagnie produttrici di programmatori di dispositivi PAL. A testimonianza della grande diffusione delle PAL basti ricordare come, anche dieci anni dopo l'introduzione dei primi componenti, questi rappresentavano ancora la soluzione principale adottata dai progettisti, sebbene già allora fossero commercialmente disponibili oltre 200 *PLD*, ognuna dalle proprie caratteristiche. Tuttavia la *MMI* non fu in grado di adeguare, almeno inizialmente, la produzione di *PAL* alla enorme richiesta. La *Digital Equipment Corporation*, che impiegava questi dispositivi nella realizzazione dei calcolatori *VAX730*, suggerì alla *MMI* lo sviluppo di una versione programmabile a maschera di questi componenti, denominata in seguito *HAL* (*Hard Array Logic*). I componenti *HAL* ebbero un enorme successo che consentì alla *MMI* una rapida espansione. In seguito, gli sforzi della società furono tesi a sviluppare *PAL* dalle prestazioni più elevate, in concorrenza, inizialmente, con la *AMD* (*Advanced Micro Device*). Furono prodotti ad esempio i dispositivi *MegaPAL 64R32* ad 84 pin, che si rivelarono però un insuccesso, in quanto presentavano soltanto piccoli vantaggi rispetto all'utilizzo di un numero maggiore di *PLD* dalle prestazioni inferiori.

L'introduzione della tecnologia CMOS

A quei tempi la tecnologia dominante nello sviluppo dei dispositivi *PAL* era la bipolare, con cui erano state realizzate le prime *PROM*. Si dovette aspettare lo sviluppo degli EP300 da parte dell'*Altera*, nella primavera del 1984, per avere i primi componenti programmabili realizzati in tecnologia *CMOS*. La nuova tecnologia consentiva di produrre dispositivi a densità di integrazione più elevata, caratterizzati da maggiore complessità e migliori prestazioni, anche se va detto che i nuovi componenti richiedevano lo sviluppo di *tool* di supporto più sofisticati.
Alcuni mesi dopo l'introduzione degli *EP300*, l'*Altera* presentò il sistema denominato *A + PLUS*, per personal computer *IBM*. L'ambiente di sviluppo comprendeva, inoltre, un semplice programmatore che si inseriva in una delle schede di espansione del calcolatore. La strada aperta dall'*Altera* fu percorsa in seguito anche da altre compagnie, quali la *Cypress Semiconductor* in collaborazione con la *MMI*, o la *Lattice Semiconductor* e la *ICT* (*International CMOS Technology)*.
Due anni più tardi, usando la tecnologia CMOS, Xilinx - fondata solo un anno prima da Ross Freeman, Bernie Vonderschmitt e Jim Barnett - irruppe sul mercato con i dispositivi *LCA* (*Logic Cell Array*) XC2064 che a tutti gli effetti possono essere considerati il primo

Fig. 4 *Nel 1985 Ross Freeman inventa gli FPGA.*

esempio di FPGA. Nel febbraio 2009, le sue invenzioni sono valse a Ross Freeman l'entrata nella *National Inventors Hall of Fame*.
Diversamente dai dispositivi *PLD*, che presentano un'unica struttura programmabile, le LCA XC2064 si basavano su una matrice di celle configurabili denominate *CLB* (*Configurable Logic Block*) in grado di realizzare qualunque funzione logica formata da un numero limitato di variabili e su una rete programmabile di interconnessioni tra queste.
Dopo la realizzazione dell'XC2064 l'industria delle FPGA ha avuta una forte espansione, grazie anche ai continui progressi tecnologici nel settore dei semiconduttori, raggiungendo nel 2005 una quota di mercato vicina ai 2 miliardi di dollari. Oggi sono disponibili sul mercato dispositivi di capacità logica equivalente a milioni di gate, che integrano in un singolo chip funzionalità eterogenee quali *core* di processori o transceiver seriali; la tecnologia ha raggiunto i 40 nm, riducendo consumi di potenza e ingombri, incrementando prestazioni ed affidabilità.
Di seguito riportiamo una panoramica delle più significative soluzioni e di alcune delle principali famiglie di dispositivi FPGA oggi presenti sul mercato.

UN UNICO MONDO, DIVERSE TECNOLOGIE

Attualmente per la realizzazione dei dispositivi FPGA vengono utilizzate principalmente tre tecnologie:

- antifuse
- flash
- RAM statica.

Funzionalità ed architetture possono essere in certi casi anche profondamente differenti. Qui di seguito esaminiamo una ad una le possibili tecnologie.

La tecnologia antifuse

I dispositivi *antifuse* sono caratterizzati dalla presenza da tantissimi moduli logici piuttosto semplici, con funzionalità predefinite, che sono connessi tra loro mediante antifusibili. Questi elementi, al contrario dei fusibili, sono normalmente aperti ma diventano dei cortocircuiti quando vi viene fatta scorrere una corrente maggiore di un certo valore di soglia. In questo modo sono abilitate le interconnessioni tra i moduli. I dispositivi *antifuse* sono quindi non volatili, nel senso che la configurazione del dispositivo resta memorizzata anche quando la tensione di alimentazione viene rimossa; però non sono riprogrammabili. Tra i vantaggi principali delle FPGA *antifuse* vi sono una minore dissipazione di potenza, l'operatività immediata all'accensione, minori ritardi di propagazione delle interconnessioni, una riduzione dell'area di silicio occupata; i dispositivi *antifuse* sono inoltre tipicamente più affidabili in applicazioni critiche ed

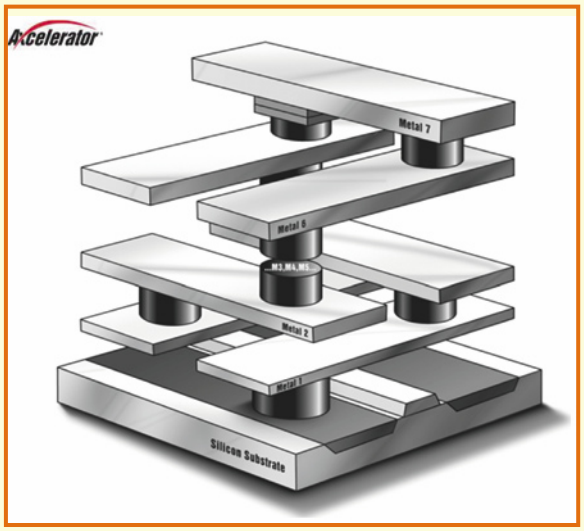


Fig. 5 - *Tecnologia antifuse proprietaria per i dispositivi Actel Axcelerator.*

assicurano migliore protezione del progetto da tentativi di *hacking* o *cloning*, in quanto non prevedono l'impiego di memorie esterne per la configurazione del dispositivo.
I principali svantaggi della tecnica *antifuse* sono invece l'impiego di un processo di fabbricazione non proprio standard, che ha determinato negli anni una crescita delle prestazioni più lenta rispetto a quanto osservato invece per i dispositivi riprogrammabili in tecnologia RAM statica, una minore capacità logica, la non riprogrammabilità del componente e, di contro, la necessità di programmare il dispositivo prima del montaggio su scheda utilizzando, per di più, programmatori dedicati.
Il principale produttore al mondo di logiche programmabili in tecnologia *antifuse* è la Actel; uno dei suoi prodotti è la serie *Axcelerator*, composta da FPGA di capacità logica fino a 4 milioni di gate, disponibili in package di tipo tradizionale (CSP, PQFP e CQFP) o *grid-array* (BGA, FBGA, CCGA). I moduli logici disponibili sono di tipo R-Cell, che implementano flip-flop con abilitazione ed ingressi asincroni di preset e clear, e C-Cell, che includono, invece, un multiplexer a quattro ingressi e logica di riporto dedicata. Programmando le interconnessioni tra C-Cell è possibile realizzare una qualunque funzione logica di *n* variabili. I diversi moduli, come mostrato nella **Fig. 5**, sono diffusi negli strati di silicio, mentre la matrice di interconnessioni è realizzata impiegando i piani di metallo più esterni del "die" (il chip...) con una tecnologia antifusibile proprietaria che consente di ridurre occupazione di area e ritardi di propagazione.

La tecnologia flash

Simile alla *antifuse*, almeno da un punto di vista di principio, è l'architettura delle FPGA *flash*: in esse gli elementi di interconnessioni programmabili si basano su semplici switch realizzati, appunto, in tecnologia *flash*. *ProAsic3*, ad esempio, è la nuova serie di logiche programmabili realizzata dalla *Actel* con questa tecnologia; elemento fondamentale è, in questo caso, il *VersaTile*, cioè un insieme di semplici elementi logici di base come multiplexer e porte NOT (le cui connessioni possono essere programmate per realizzare una qualunque funzione logica di tre variabili) un latch con ingressi di set e clear o un flip-flop di tipo D con ingressi di clock_enable e set/reset. Ulteriori switch consentono la connessione delle linee di ingresso/uscita dei *VersaTile* alla rete di distribuzione dei segnali lungo tutta l'FPGA. Ogni switch è realizzato mediante un transistor con *floating-gate*, ovvero materiale conduttivo, rivestito di isolante e disposto tra gate e canale, tipicamente utilizzato per memorizzare la carica; un secondo transistor è invece utilizzato per il read-back e la verifica della configurazione dello switch. Quando è abilitato, lo switch assicura una connessione a bassa resistenza.
La struttura descritta consente una minore occupazione di area rispetto alle soluzioni adottate, invece, nel caso di dispositivi basati su tecnologia SRAM: in questo caso sono tipicamente utilizzati da quattro a sei transistor per ogni elemento programmabile; per questo motivo, le FPGA *ProAsic3/E* possono essere realizzate impiegando un processo costruttivo a minore miniaturizzazione, che risulta quindi più conveniente. L'impiego della tecnologia *flash*, inoltre, assicura l'immunità del dispositivo ai *firm-error*, ovvero ai difetti di funzionamento conseguenza di un cambiamento di stato dell'elemento di configurazione, cambiamento dovuto all'impatto di neutroni ad elevata energia prodotti nelle zone alte dell'atmosfera terrestre. Anche le FPGA in tecnologia *flash* sono non volatili ma a differenza delle *antifuse* possono essere riprogrammate, sebbene la programmazione richieda tensioni accessorie a quelle necessarie per il normale funzionamento; i dispositivi *ProAsic3*, ad esempio, supportano fino a 500 cicli di cancellazione e programmazione e possono essere

Fig. 6 *- ProAsic3, FPGA in tecnologia flash: soluzioni single-chip low-power.*

programmate in-circuit dopo il montaggio sul circuito stampato. Per proteggere il dispositivo da tentativi di clonazione possono essere usati chiavi di cifratura del bitstream di configurazione; nei casi più critici si può completamente inibire l'accesso alla matrice di configurazione, ma in tale evenienza si perde la possibilità di riconfigurare l'FPGA. La famiglia *ProAsic3* include dispositivi di capacità logica fino a tre milioni di gate, anch'essi in package tradizionali (QN64, VQ100, PQ208) oppure di tipo *fine-pitch grid-array* fino a 896 pin, destinati ad applicazioni che richiedano un elevato numero di segnali di ingresso/uscita. Estremamente interessanti, inoltre, sono le versioni *ARM-enabled*, che vengono distribuite con una sezione del dispositivo già programmata per implementare un processore ARM di tipo *CoreMP7* o *Cortex-M1*; il progettista può utilizzare la restante parte di risorse per realizzare la propria logica utente che diventa quindi una periferica accessibile da parte del processore.

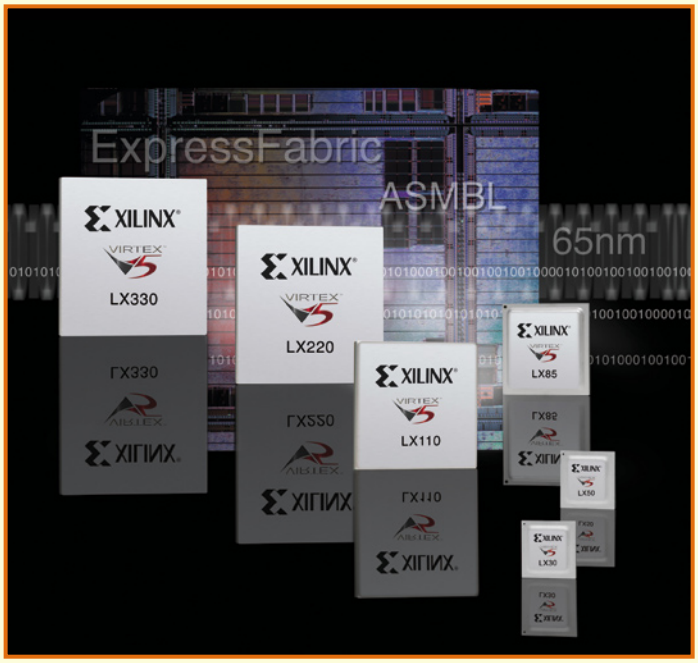


***Fig. 7** - Xilinx VIrtex-5: le prime FPGA in tecnologia triplo-ossido a 65 nm con processore PowerPC440, end-point PCIe ed Ethernet MAC embedded.*

La tecnologia SRAM

Piuttosto diversa è, invece, la struttura dei dispositivi logici programmabili basati su tecnologia SRAM; infatti in questo caso le funzioni logiche non vengono realizzate connettendo tra di loro delle *primitive* ma impiegando LUT (*Look-Up Table*). Una LUT è una memoria statica a *n* bit di indirizzamento; data una funzione logica di *n* variabili, e nota la sua tabella della verità, il valore della funzione per una qualsiasi combinazione degli ingressi viene programmato nella LUT all'indirizzo corrispondente a quella stessa combinazione di ingressi. Ad esempio nel caso di una funzione a 4 ingressi che restituisce "1" quando i suoi ingressi valgono "1001", la LUT sarà programmata con il valore "1" all'indirizzo 0b1001 = 0d9. In questo modo, una LUT ad *n* ingressi può realizzare una qualsiasi funzione logica di *n* variabili. Diversamente, nel caso dei dispositivi *antifuse* e *flash*, il numero di *primitive* connesse e le loro modalità di connessione dipendono dal tipo di funzione che si intende realizzare.
Quindi nelle FPGA in tecnologia SRAM, per creare le LUT vengono utilizzate celle di memoria riprogrammabili. Una fitta rete di linee di distribuzione di segnali consente poi di connettere le LUT tra loro; ulteriori celle statiche vengono utilizzate per collegare gli ingressi/uscite delle LUT alla rete di interconnessioni. Le FPGA basate su RAM statica sono realizzate con una tecnologia CMOS standard che consente di integrare piuttosto facilmente anche elementi diversi; ad esempio i moderni dispositivi incorporano estesi blocchi di memoria RAM embedded, circuiti di riporto veloci per la realizzazione di addizionatori, blocchi DSP ed accumulatori per applicazioni di elaborazione digitale dei dati, interfacce SerDes ad elevato data-rate (dell'ordine dei gigabit/s), controller Ethernet e *core* di processori embedded. I vantaggi principali dei dispositivi SRAM sono l'elevata capacità logica, l'elevato livello di integrazione, le migliori prestazioni in termini di frequenza di funzionamento nella maggior parte delle applicazioni, i costi minori a parità di funzionalità e risorse. Non meno importante è la possibilità di riprogrammare i dispositivi, che in fase di debug consente di fissare velocemente errori di progettazione e che in applicazioni su campo permette di modificare le

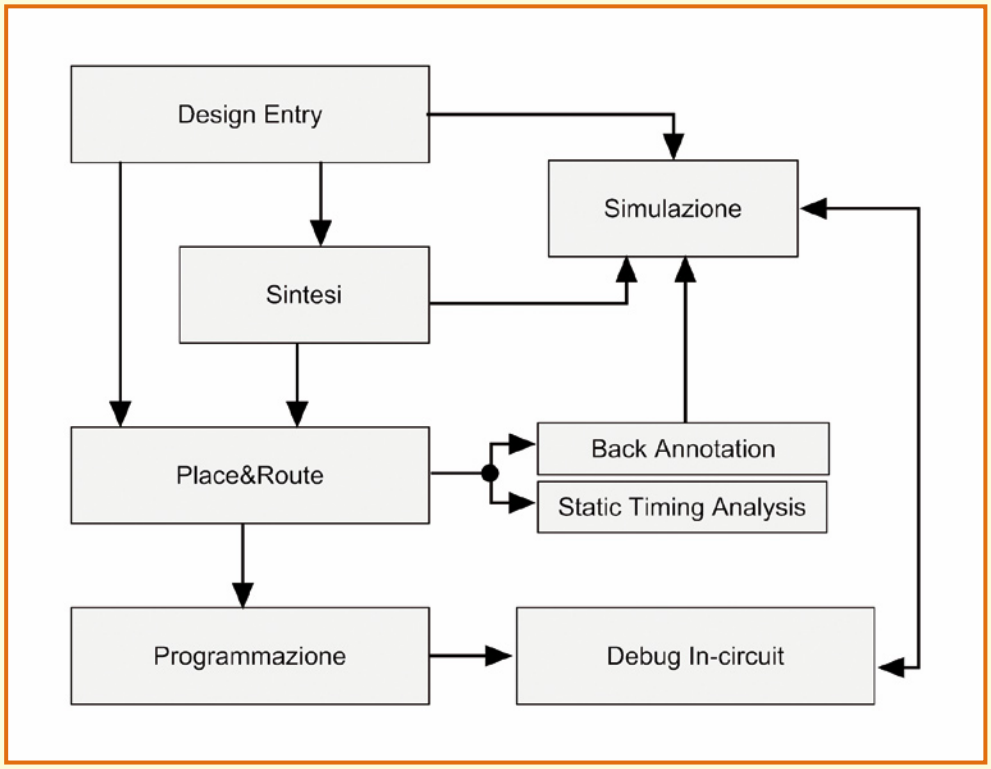


Fig. 8 *- Il flusso di progetto con FPGA.*

funzionalità del dispositivo in base ai diversi scenari operativi. Le FPGA di tipo SRAM sono dispositivi volatili e quindi per poter essere usate, ad ogni accensione richiedono il caricamento della configurazione da una memoria esterna; anche in questo caso sono previsti meccanismi di cifratura del bit-stream di configurazione per evitare clonazioni del dispositivo nelle applicazioni critiche.
In funzione della complessità e del tipo di dispositivo, i tempi di programmazione possono variare da pochi millisecondi fino ad oltre un secondo. Il dispositivo quindi non è *live* al *power-up*, il che potrebbe non essere accettabile in alcune applicazioni critiche. Minore è poi la tolleranza ai *firm-error* e maggiore la dissipazione di potenza. I principali produttori al mondo di logiche programmabili in tecnologia SRAM sono *Xilinx* e *Altera*. I nuovi dispositivi *Virtex-5* di *Xilinx*, in particolare, utilizzano LUT a sei ingressi ed integrano fino a 11 Mbit di RAM, transceiver *RocketIO GTP* fino a 3,75 Gbps che supportano, ad esempio, i principali standard di comunicazione seriale per applicazioni telecom (OC-48, OBSAI), video (HD-SDI) o computing (*PCIexpress, SRIO, Infiniband*); integrano inoltre fino a 4 *Endpoint PCI-Express* e 8 *Ethernet MAC*, monitor di sistema e convertitore analogico-digitale per misure di parametri vitali come temperatura del chip, tensioni e correnti assorbite. Incorporano anche fino a due processori IBM *PowerPC 440*. I dispositivi della serie *Spartan-3*, invece, rappresentano la soluzione per i sistemi low-cost e le produzioni di massa nella maggior parte delle applicazioni consumer. *Stratix-IV* e *Cyclone-III*, infine, sono i dispositivi concorrenti distribuiti da *Altera*.

IL FLUSSO DI PROGETTO

Il flusso di progetto riguardante la programmazione di circuiti contenenti FPGA segue uno schema di principio che tipicamente è quello rappresentato nella **Fig. 8**. Le operazioni del caso sono divise nelle fasi che esamineremo qui di seguito.

Design Entry

La prima fase è quella di *Design Entry*, che consiste nella descrizione delle funzioni logiche che il dispositivo dovrà implementare. Esistono per questo diverse modalità che vanno dal disegno dello schema circuitale in ambienti CAD appositi ed utilizzando *primitive* di base come flip-flop e porte logiche, a descrizioni di più alto livello basate su diagrammi a

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
 generic ( c_width : integer := 8 );
 port
 (
  arst : in std_logic;
  clk  : in std_logic;
  ce   : in std_logic;
  q    : out unsigned(c_width-1 downto 0)
 );
end entity counter;

architecture rtl of counter is
 signal q_i  : unsigned(c_width-1 downto 0);
begin

 Cnt : process (arst, clk)
 begin
  if arst='1' then
   q_i <= (others=>'0');
  elsif clk'event and clk='1' then
   if ce='1' then q_i <= q_i+1; end if;
  end if;
 end process Cnt;

 q <= q_i;

end rtl;
```

Fig. 9 - *Descrizione di un semplice contatore in VHDL.*

bolle o linguaggi di descrizione hardware. Tra questi ultimi, i due più diffusi sono il *Verilog* ed il *VHDL*. Nella prossima puntata vedremo con un semplice esempio come descrivere un circuito elettrico in *VHDL*; a titolo di semplice curiosità, nella riquadro illustrato nella **Fig. 9** è riportata la descrizione di un contatore di tipo *up* con ingresso di abilitazione e reset asincrono. Mentre agli inizi il disegno degli schemi elettrici era la metodologia principale, oggi la progettazione di FPGA è praticamente interamente basata sui linguaggi di descrizione hardware. In effetti, la crescente complessità dei dispositivi logici programmabili e dei sistemi in essi implementati, unitamente alla necessità di ridurre i tempi di sviluppo, sta spingendo sempre più all'utilizzo di metodi e linguaggi di alto livello. Da questo punto di vista, ad esempio, esistono librerie e supporti per la descrizione di circuiti logici in *Matlab/Simulink* con generazione automatica di codice *HDL*. Più o meno allo stesso modo, *Catapult*, un tool di *Mentor Graphics*, consente di generare automaticamente una descrizione *HDL* a partire da una in linguaggio C++; tali soluzioni sono principalmente utilizzabili per algoritmi di elaborazione dati.
Il *SystemC*, invece, è una libreria per C++ che introduce in questo linguaggio i concetti di tempo, parallelismo e concorrenza e permette quindi di creare modelli di circuiti logici; un sottoinsieme dello standard che risulti traducibile automaticamente in porte logiche e registri è in fase di standardizzazione.

Simulazione
Come mostrato nella **Fig. 8**, alla fase di *design entry* seguono le attività di simulazione e verifica funzionale del circuito; obiettivo è verificare che la descrizione che se ne è fatta corrisponda effettivamente al comportamento atteso per il circuito. In maniera piuttosto semplicistica, si tratta di stimolare gli ingressi del nostro circuito e verificare che i segnali di uscita seguano l'andamento atteso. Esistono diversi tool che consentono di interpretare le descrizioni dei circuiti in linguaggio *HDL* per simularne il comportamento. Due dei più diffusi sono *ModelSim* di *Mentor Graphics* e *ActiveHDL* di *Aldec*. La **Fig. 10** mostra ad esempio l'ambiente di lavoro di *ModelSim* con le diverse finestre disponibili durante una simulazione; sono previste, ad esempio, una finestra di editor delle forme d'onda dei segnali che consente di visualizzarne l'andamento ed una di debug che consente di eseguire invece passo passo il codice che rappresenta il circuito. Descrizioni in linguaggio *SystemC* possono invece essere simulate all'interno di un tradizionale catena di debugger C++ come quella *GNU/GCC* o *Microsoft Visual Studio*.

Sintesi
La fase successiva nel flusso di progetto con FPGA prevede la traduzione della descrizione del circuito in un formato che i tool proprietari della case produttrici di FPGA siano in grado

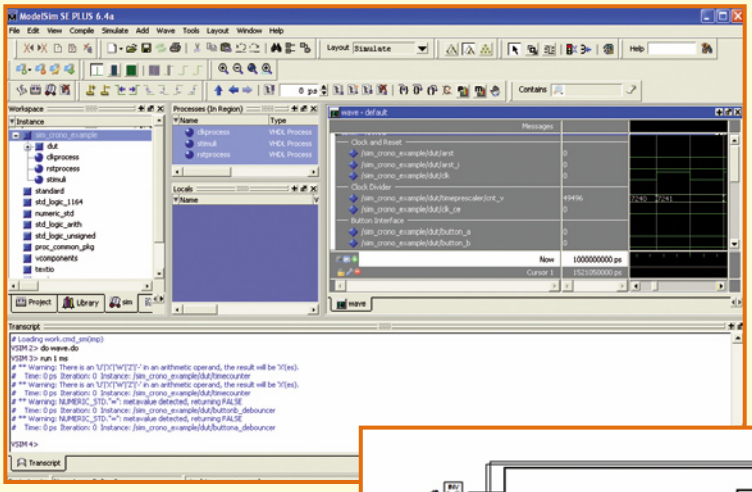
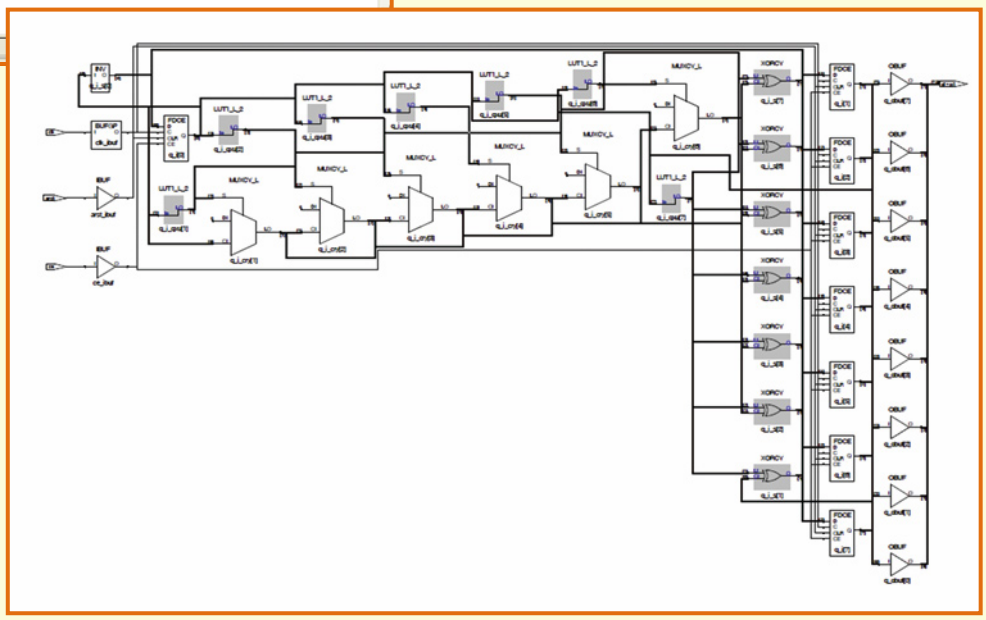


Fig. 10 - *Simulazione VHDL in ModelSim.*

Fig. 11 - *Sintesi di un contatore con Synplicity Synplify.*



di interpretare per poter generare alla, fine, il bitstream di programmazione. Nel caso si abbia a che fare con descrizioni fatte in linguaggio *HDL* tale fase viene altrimenti indicata come *sintesi* del circuito. Esistono diversi tool di *sintesi*, tra cui *Leonardo Spectrum* e *Precision RTL* di *Mentor Graphics* o *Synplify* di *Synplicity*; a questi si affiancano tool proprietari dei produttori di FPGA come ad esempio *XST* di *Xilinx* o *Quartus* di *Altera*. Il processo di sintesi astrae, in definitiva, le funzionalità logiche del circuito dalla descrizione e le traduce in porte logiche, flip-flop e connessioni tra questi; in **figura 11** è mostrato ad esempio il circuito sintetizzato con *Synplify* per il contatore la cui descrizione è stata presentata in precedenza (**figura 9**). La descrizione in termini di *primitive* logiche viene indicata con il termine di *netlist*; esistono diversi formati, di cui alcuni sono standard, altri proprietari. Tra i più diffusi vi è il formato EDIF (*Electronic Design Interchange Format*). Oltre a generare la *netlist*, il processo di *sintesi* fornisce una indicazione preliminare sulle risorse occupate nel dispositivo target e della frequenza di funzionamento massima che si potrà ottenere; permette inoltre di generare automaticamente una nuova descrizione in linguaggio HDL del circuito che si basi direttamente su porte logiche e segnali di interconnessione. Ove lo si ritenga opportuno, è possibile simulare tale descrizione per verificare che il circuito corrispondente sia corretto. In realtà, i moderni tool sono diventati piuttosto affidabili e, supposto che non vi siano errori nella descrizione di partenza, solo in casi rari (legati principalmente a particolari stili di codifica del codice HDL di partenza da parte del progettista) i risultati della *sintesi* non sono corretti.

Place&Route

La fase successiva a quella di *sintesi* è denominata *Place&Route* e serve ad associare le porte e le funzioni logiche presenti nella *netlist* alle risorse del dispositivo, definendo poi i *path* di connessione tra queste; è possibile fornire dei *constraint* sulla disposizione delle risorse

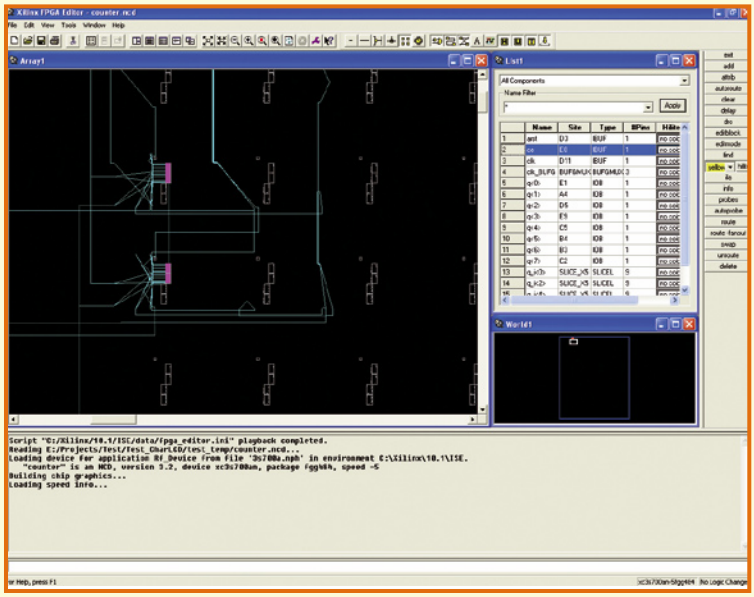


***Fig. 12** - Implementazione di un contatore in FPGA (in azzurro le risorse occupate.*

utilizzate all'interno della FPGA, sulla associazione dei segnali di ingresso/uscita ai pin del dispositivo o, ancora, sulla massima frequenza di funzionamento che si intende raggiungere. I tool di *place&route* usano algoritmi di natura casuale per il piazzamento ed il routing delle *primitive*; il processo quindi non è mai ripetibile.
Anche se la *netlist* è la stessa, è del tutto improbabile che il layout della FPGA al termine del processo di *place&route* appaia lo stesso in due diverse implementazioni.
Successivamente alla fase di *place&route*, viene creato il database con i ritardi dei singoli segnali. Tale database può essere estratto in un file di formato standard SDF (*Standard Delay Format*) unitamente ad una descrizione del circuito in linguaggio HDL basata direttamente su modelli delle *primitive* del dispositivo cui possono, appunto, essere associati i ritardi di propagazione stimati dei segnali.
Il layout della FPGA può essere analizzato e verificato rispetto ai parametri temporali imposti (massima frequenza di funzionamento, rispetto dei *constraint* di *setup/hold* e *clock-to-output*) ed ai requisiti in termini di dissipazione di potenza.

Programmazione e debug-in-circuit
L'ultima fase del processo prevede la generazione del file di programmazione della FPGA. Come abbiamo visto in precedenza, per i dispositivi *antifuse* dovrà essere usato un apposito programmatore per programmare il dispositivo.
Nel caso invece dei dispositivi *flash*, il file di programmazione viene convertito in altri formati e solitamente mediante cavo parallelo connesso al PC scaricato nella memoria esterna di configurazione del dispositivo. Una volta programmato il dispositivo, non resta che verificarne nella realtà il corretto funzionamento.
Spesso, a seguito della crescente complessità dei sistemi implementati in logiche programmabili, si è determinata negli ultimi anni la necessità di utilizzare metodologie di debug in circuit per risolvere bug di progettazione. Il problema, in effetti, è che proprio a causa di questa complessità è praticamente impossibile coprire tutti i casi di utilizzo del dispositivo in simulazione.
Con il debug-in-circuit, allora, si ha la possibilità di vedere l'andamento nel tempo dei segnali della FPGA durante il reale funzionamento di questa.
Analizzando questi è possibile scoprire l'origine del malfunzionamento e replicarlo in simulazione per risolverlo; in fondo è come avere la possibilità di simulare nella realtà il comportamento del circuito.
Esistono diversi tool di debug in circuit tra i quali *ChipScope* di *Xilinx* oppure *Identify* di *Synplicity*.

CONCLUSIONI

Come abbiamo visto, le FPGA rappresentano una delle principali innovazioni che hanno coinvolto il mercato dei semiconduttori negli ultimi vent'anni.
L'introduzione delle logiche programmabili ha definitivamente modificato il modo di progettare i sistemi digitali.
L'articolo ha presentato una introduzione al mondo delle FPGA, alle modalità di utilizzo ed al flusso di progettazione che viene seguito per il loro impiego.
Nella prossima puntata approfondiremo l'architettura delle FPGA *Xilinx Spartan-3AN* e presenteremo un kit di sviluppo che può essere impiegato per iniziare a progettare con questi dispositivi.

Corso FPGA

Continuiamo la panoramica sulle logiche programmabili, descrivendo il funzionamento e l'uso dei chip Spartan-3 prodotti dalla Xilinx.

di
MARIANO SEVERI

Nell'articolo apparso sul precedente numero della rivista abbiamo fatto un'introduzione alle FPGA, ripercorrendo la storia di questi importantissimi componenti elettronici e tracciando una panoramica del mercato attuale. In questo articolo conosceremo una particolare famiglia di dispositivi, che sono le FPGA della serie Spartan-3AN prodotte dalla Xilinx; presenteremo quindi un kit di sviluppo a basso costo che potrà essere utilizzato per imparare a progettare con questi nuovi dispositivi ed alcuni semplici esempi di riferimento.

LE FPGA XILINX SPARTAN-3

Le FPGA della serie Spartan-3 sono state i primi dispositivi logici riprogrammabili in tecnologia a 90 nm ad essere immessi sul mercato. Introdotte nel 2003 e specificatamente progettate per applicazioni a basso costo in produzioni di massa (ad esempio nelle più tradizionali apparecchiature consumer come sistemi di accesso a larga banda, home networking, video digitale, digital signal processing) hanno rappresentato una vera e propria rivoluzione nel mondo delle logiche programmabili,

rendendo disponibili più funzionalità e capacità di trasmissione per costo unitario di quanto fino ad allora offerto.
Le caratteristiche principali dei dispositivi Spartan-3 sono elencate qui di seguito.

- Elevata capacità logica:
  - fino ad oltre 74 mila celle logiche;
  - shift-register a 16-bit dedicati;
  - multiplexer efficienti per funzioni logiche complesse;
  - logica di riporto look-ahead veloce;
  - moltiplicatori embedded 18x18;
  - porta di programmazione/debug IEEE 1149.1/1532 JTAG.

- Memoria embedded:
  - fino a 2.268 kbit di memoria block RAM veloce;
  - fino a 373 kbit di memoria RAM distribuita.

- Tecnologia SelectIO™ con supporto multistandard:
  - fino a 633 pin di I/O in standard single-ended o 300 in standard differenziale;
  - supporto per standard single-ended LVCMOS, LVTTL, HSTL e SSTL;
  - supporto per standard differenziali LVDS, RSDS, mini-LVDS, PPDS, HSTL/SSTL;
  - capacità di driving configurabile fino a 24 mA;
  - uscite Double Data Rate (DDR);
  - transfer rate per I/O fino a 622 Mbps.

- Fino a 8 Digital Clock Managers:
- riduzione dello skew delle reti di distribuzione di clock;
- shift in fase con elevata risoluzione;
- sintesi in frequenza tra 5 MHz e 300 MHz.

- Otto linee globali per la distribuzione di clock e diverse risorse locali a basso skew.
- Compatibilità con applicazioni PCI 32-64 bit/66 MHz e disponibilità di core IP per sistemi PCI Express.
- Disponibilità di packaging QFP e BGA Pb-free (compatibili Rohs).
- Supporto da parte dei principali tool EDA.

Grazie a queste caratteristiche, le FPGA Spartan-3 rappresentano oggi una delle più valide alternative agli ASIC, riducendo i costi iniziali non ricorrenti, i tempi di sviluppo ed i rischi inerenti a questa tecnologia; la riprogrammabilità del dispositivo assicura flessibilità e permette l'aggiornamento direttamente sul campo delle funzionalità senza richiedere alcuna modifica hardware.
Le FPGA Spartan-3AN combinano caratteristiche e prestazioni della famiglia Spartan-3 con i vantaggi delle tecnologia flash non volatile in un prodotto ideale per applicazioni come blade-server, dispositivi medicali, apparecchiature di intrattenimento in ambito automotive, sistemi telematici, GPS e, in generale, qualunque altro sistema consumer che sia critico dal punto di vista dei consumi, degli ingombri e dei costi, ma che, contemporaneamente, richieda elevate prestazioni.
Oltre alle caratteristiche proprie della famiglia Spartan-3, infatti, i dispositivi Spartan-3AN dispongono fino a 11 Mbit di memoria flash interna che può essere utilizzata per la memorizzazione del bitstream di configurazione della FPGA (evitando così l'impiego di una memoria esterna dedicata) o come memoria scratchpad, supportando fino a 100 mila cicli di programmazione/cancellazione ed assicurando fino a 20 anni di data retention. Nella memoria della FPGA può essere programmata più di una configurazione, rendendo possibili applicazioni multi-boot o riconfigurazioni dinamiche in funzione delle applicazioni. Introdotte da Xilinx nel 2007, le FPGA Spartan-3AN sono state incluse nella lista dei 100 migliori prodotti dell'anno dalla EDN.

ARCHITETTURA INTERNA

La **Fig. 1** mostra uno schema di principio dell'architettura interna dei dispositivi Spartan-3 che ne evidenzia gli elementi principali:

- CLB (Configurable Logic Block): includono le LUT per la generazione delle funzioni logiche e gli elementi di memoria di tipo flip-flop o latch;
- Input/Output Block (IOB): consentono la connessione dei segnali interni ai pin di ingresso/uscita del dispositivo
- Block RAM: instanziano blocchi di memoria dual-port da 18 kbit;
- Multiplier: realizzano moltiplicatori embedded con due ingressi a 18-bit;

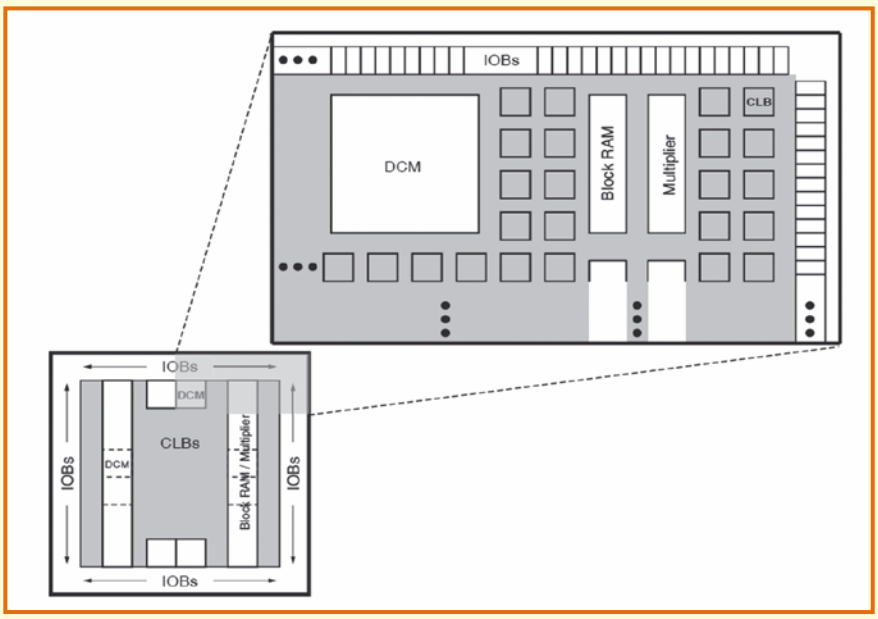


Fig. 1 *- Architettura interna dei dispositivi Spartan-3.*

- Digital Clock Manager : forniscono un circuito auto-calibrante e interamente digitale per la distribuzione, il ritardo, la moltiplicazione, la divisione o lo shift in fase dei segnali di clock.

Una fitta rete di interconnessioni assicura, quindi, una comunicazione flessibile ed efficace tra i diversi elementi. Di seguito sono descritte in dettaglio le principali caratteristiche di ognuno dei moduli sopra elencati.

Struttura delle CLB

Le CLB (Configurable Logic Blocks) rappresentano le risorse di base per la realizzazione delle funzionalità logiche in FPGA. Ogni CLB, come mostrato nella **Fig. 2**, consiste di quattro slice, divise in due gruppi di tipo SliceL e SliceM. Ogni Slice include due LUT a 4 ingressi per la generazione delle funzioni logiche, due elementi di memoria, due multiplexer e circuiti di riporto. L'uscita delle LUT può essere connessa alla logica di multiplexing, alla logica aritmetica e di riporto, agli elementi di memoria o, in ultimo, direttamente ai segnali di uscita delle slice.

Le LUT possono generare una qualsiasi funzione logica di 4 variabili. Impiegando le due LUT di una slice ed il multiplexer 2:1 F5MUX è quindi possibile realizzare una qualsiasi funzione di 5 variabili o alcune funzioni di un numero maggiore di variabili. Una generica funzione di 5 variabili, ad esempio, può essere realizzata implementando nelle due LUT le due funzioni corrispondenti ottenute fissando ad 1 e 0 rispettivamente uno dei 5 ingressi; quest'ultimo viene quindi usato per controllare F5MUX per multiplexare le due uscite. Un esempio di funzioni più complesse realizzabili in una singola CLB combinando LUT e multiplexer è invece quello di un multiplexer 4:1 (funzione di 6 variabili) implementato mediante due multiplexer 2:1 realizzati con LUT e multiplexati a loro volta mediante F5MUX.

Gli elementi programmabili di una CLB sono flip-flop di tipo D con segnale di abilitazione del clock, ingressi di set/reset sincroni e preset/clear asincroni; in alternativa possono essere configurati come latch. La polarità dei segnali di controllo è in ogni caso configurabile.

Nelle slice di tipo SliceM, le LUT, oltre che come generatori di funzioni, possono essere configurate come un banco di memoria distribuita da 16 bit oppure come un registro a scorrimento a 16 bit (ad esempio nella implementazione di memorie CAM). Quando configurate come memoria distribuita, supportano accesso sincrono in scrittura ed asincrono in lettura; un'interfaccia sincrona in lettura può essere realizzata connettendo l'uscita della memoria ai registri della CLB. Due LUT all'interno della stessa slice possono essere combinate per creare una RAM dual-port avente una porta accessibile in lettura/scrittura ed una in sola lettura. Il contenuto della memoria può essere inizializzato al power-up con valori definiti dall'utente.

I moduli di I/O

Le risorse di Input/Output, in altro modo indicate come IOB o I/O Block, rappresentano l'interfaccia per il trasferimento dei segnali da e verso l'esterno del dispositivo. Da questo punto di vista, le FPGA Spartan-3 semplificano il progetto di sistema soprattutto nelle applicazioni più critiche, consentendo di

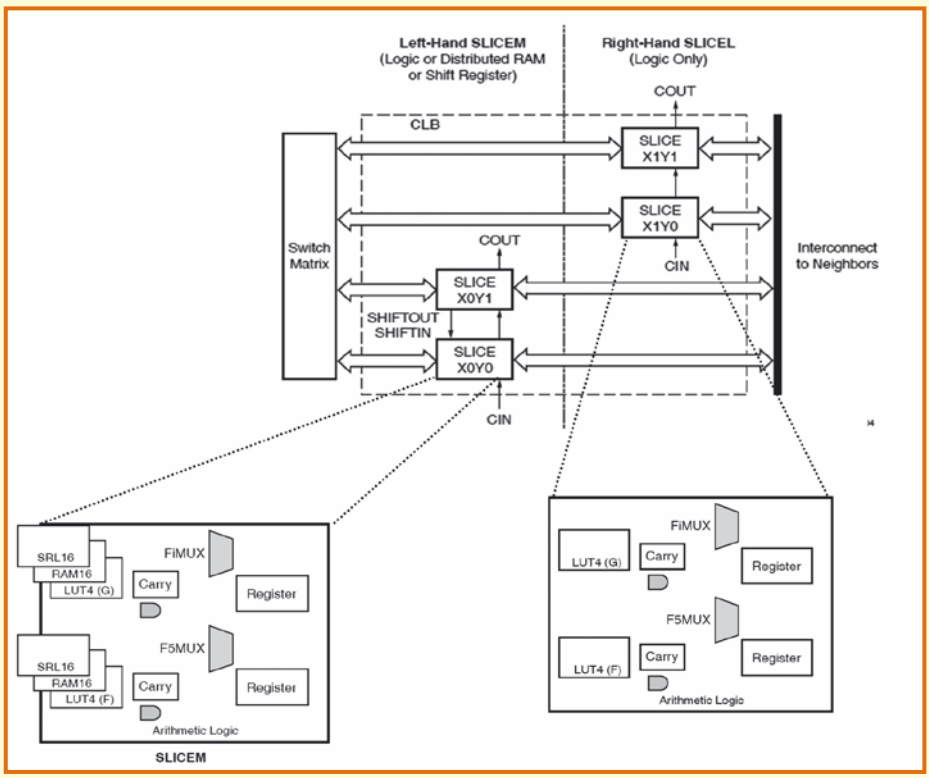


Fig. 2
Struttura delle CLB.

configurare fino a 20 diversi standard per le caratteristiche dei buffer di input/output con differenti curve tensione/corrente e supporto per diverse terminazioni. Evitando così il ricorso a traslatori di livello esterni si riducono complessità e costi delle schede.
In ogni IOB sono presenti tre diverse connessioni: di ingresso, di uscita e di abilitazione. Per ognuno di questi sono previsti due elementi di memoria configurabili come flip-flop o latch. In uscita sono supportati circuiti double-data rate; in ingresso sono previsti elementi di ritardo programmabili che consentono di ridurre a zero le specifiche di hold per i segnali.
È possibile abilitare resistori di pull-up e pull-down o un circuito di keeper che consente di mantenere sulla linea l'ultimo livello logico anche se il driver è disabilitato.

Memoria Embedded

Oltre alla RAM distribuita presente nelle SliceM, le FPGA Spartan3-AN contengono blocchi di memoria embedded (Block RAM o BRAM) da 18 kbit, di cui 16 kbit utilizzabili come dati e 2 kbit riservati, invece, per un eventuale controllo di parità. Sono supportate diverse configurazioni in termini di larghezza di parola che vanno dal singolo blocco da 16k locazione da 1 bit fino ad un banco da 512 locazioni da 36 bit. Il controllo di parità può essere abilitato per larghezze di parola maggiori di 8 bit. Più memorie possono essere connesse in cascata per realizzare banchi di capacità o larghezza di parola più estesa.
Le memorie sono native di tipo dual-port con entrambe le porte accessibili in lettura e scrittura; è possibile accedere ad entrambe le porte contemporaneamente. Dati, indirizzi e controlli sono infatti indipendenti per ogni porta. Le due porte possono essere inoltre configurate con larghezza di parola diversa. Nel caso di accessi simultanei alla stessa locazione da entrambe le porte, la memoria può essere configurata nelle diverse modalità *write_first* (la locazione indirizzata in scrittura viene effettivamente scritta prima di essere letta) *read_first* (l'operazione di lettura ha priorità) e *no_change* (i latch della porta di uscita sono disabilitati durante l'operazione di scrittura).

Moltiplicatori Embedded

Oltre alla logica di riporto dedicata presente all'interno delle CLB, le FPGA Spartan-3 dispongono di moltiplicatori embedded 18x18 in complemento a due in grado di funzionare a frequenze di clock fino a 250 MHz. Per realizzare operazioni più complesse, all'interno del dispositivo possono essere presenti fino a 104 moltiplicatori compatibili; si possono connettere in cascata più moltiplicatori. Un moltiplicatore a 32 bit in virgola mobile può ad esempio essere realizzato impiegando quattro moltiplicatori embedded e della logica accessoria in CLB.
Oltre che per moltiplicazioni di interi con e senza segno, i moltiplicatori embedded possono essere utilizzati per la realizzazione di alcune funzioni logiche, risparmiando così le risorse delle CLB. Un registro a scorrimento, ad esempio, può essere realizzato con un moltiplicatore se uno dei due operandi è il vettore

da scorrere e l'altro la potenza di due pari al numero di bit da scorrere. Allo stesso modo, se uno degli operandi è il dato ed il secondo è costruito così da avere 1 nella posizione LSB e l'MSB del primo nelle restanti, si ottiene in uscita il valore assoluto del dato; il secondo operando infatti sarà -1 o +1 a seconda che il primo sia negativo o positivo. Allo stesso modo, se il secondo operando ha posti ad 1 tutti i bit fino alla posizione corrispondente all'1 più significativo del primo, il moltiplicatore finisce per realizzare la conversione del primo operando da una rappresentazione in bit e segno ad una in complemento a due.

La rete di distribuzione dei segnali

Le diverse risorse delle FPGA Spartan-3 sono interconnesse da una complessa matrice di switching che consiste in canali di:

- 24 long lines che percorrono l'intero dispositivo in orizzontale e verticale, connettendo una CLB ogni sei (grazie alla loro bassa capacità, queste linee sono ideali per la distribuzione di segnali critici veloci a basso skew);
- 8 hex lines che connettono una CLB ogni tre;
- 8 double line che connettono una CLB ogni due;
- connessioni dirette tra CLB contigue.

Sono inoltre previste reti dedicate per la distribuzione del segnale di clock a basso skew e risultano disponibili buffer connessi direttamente ai pad di I/O e multiplexer di clock che consentono la generazione in uscita di segnali senza glitch. Sono presenti DCM (Digital Clock Manager) per la riduzione dello skew, lo shift in fase del segnale (di una frazione fissa del periodo di clock o di una quantità incrementabile), la sintesi di clock a diversa frequenza (mediante moltiplicazioni e divisione di quella di partenza), la correzione del duty-cycle (fondamentale nelle applicazioni double-data rate), il mirroring e il forwarding del segnale di clock.

In-System Flash

Come accennato in precedenza, la caratteristica peculiare dei dispositivi Spartan-3AN è la disponibilità di memoria flash (In-System Flash) riprogrammabile e non volatile. La capacità totale (fino a 11 Mbit) dipende dal tipo di dispositivo selezionato.
La memoria ISF è organizzata in pagine, blocchi e settori: una pagina consiste in 264 byte (528 nelle XC3S1400AN) e rappresenta l'unità più piccola cancellabile; un blocco comprende 8 pagine ed un settore 32 blocchi, ovvero 66 kByte. Sono inoltre previsti fino a due banchi di memoria SRAM che consentono di memorizzare un'intera pagina; è possibile accedere in lettura o scrittura al singolo byte del buffer. È inoltre disponibile un Security Register di capacità 128 byte scrivibile in qualunque momento ma una sola volta; può essere impiegato per memorizzare chiavi di cifratura o codici di sicurezza.
Diverse sono le modalità di accesso supportate verso la ISF. In lettura il comando *'fast read'*, ad esempio, permette di leggere dall'array di flash un blocco di dati ad indirizzi sequenziali senza utilizzare i banchi SRAM; *'page to buffer transfer'* consente di trasferire il contenuto nella pagina nel buffer di memoria. *'Buffer write'* permette di scrivere i dati nel buffer; *'buffer to page program'* copia il contenuto del buffer nell'array flash, mentre *'page to buffer compare'* ne verifica la corretta esecuzione.

UN KIT DI SVILUPPO

Come al solito, il modo più semplice per imparare a progettare con un nuovo componente e valutarne caratteristiche e prestazioni, è quello di utilizzare uno dei sistemi di sviluppo che vengono forniti direttamente dalla casa madre o da terze parti. Nel nostro caso abbiamo selezionato "Spartan-3AN Starter Kit" descritto qui di seguito, che può essere acquistato direttamente on-line dalla relativa pagina web del sito Xilinx *http://www.xilinx.com/products/devkits/HW-SPAR3AN-SK-UNI-G.htm* o tramite il distributore italiano Avnet – Silica. Il costo è di 199 dollari americani.
Grazie alle abbondanti risorse disponibili (vedi **Fig. 3**), al supporto adeguato ed alla ricchezza di esempi di riferimento, il kit rappresenta una soluzione ideale ed a basso costo per quanti si avvicinino per la prima volta alle FPGA o per i progettisti che, pur avendo esperienza nel settore, non hanno mai lavorato con i dispositivi Spartan-3AN.
Di seguito, la lista delle risorse disponibili.

- Dispositivi Xilinx:
  - Spartan-3AN XC3S700AN-FG484;
  - platform flash XCF04S-VOG20C;
  - interfaccia di programmazione USB.
- Dispositivi di memoria:
  - SDRAM DDR2 da 32Mx16;
  - flash NOR parallela a 32 Mbit;
  - 2 flash SPI seriale da 16 Mbit.
- Dispositivi di interfaccia analogica:
  - convertitore D/A a 4 canali ;
  - convertitore A/D a 2 canale;
  - amplificatore a guadagno programmabile.
- Dispositivi periferici:
  - porta Ethernet 10/100 (interfaccia fisica);
  - 2 porte RS232 (interfaccia fisica);
  - LCD a 16-caratteri, 2 linee;
  - oscillatore a 50 MHz, ingressi per 2 clock utente.
- Connettori:
  - porta VGA a 15-pin (conversione analogico-digitale realizzata sulla scheda mediante resistori);
  - jack audio per applicazioni stereo;
  - pulsanti, switch e LEDs;
  - porta di espansione a 100-pin mediante connettore Hirose e compatibile con schede add-on di Digilent (vedi *http://www.digilentinc.com/Products/Catalog.cfm?Cat=Accessory*);
  - 2 connettori di espansione a 6-pin compatibili con schede periferiche di Digilent (vedi *http://www.digilentinc.com/Products/Catalog.cfm?Cat=Peripheral*);
  - header a 24 pin utilizzabile dall'utente;
  - porta PS/2 per mouse o tastiera.

Grazie a queste risorse ed alle schede di espansione già disponibili, potranno essere sviluppate diverse interessanti applicazioni che includono, ad esempio, video/audio recorder, web-server, data logger su memorie SD/MMC, sistemi di comunicazione in RF che impiegano varie tecniche di modulazione, video processing (interpolazione delle componenti di crominanza, conversione dello spazio di colore, filtraggio e trasformazioni per algoritmi di compressione…), acquisizione dati e digital signal processing ecc.

ESEMPI DI RIFERIMENTO

Alcuni esempi di diversa complessità, che possono essere utilizzati per prendere confidenza con il kit di sviluppo, sono già disponibili sul sito Xilinx *http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm*.
Lo stesso kit, del resto, viene fornito con già programmata a bordo una demo che permette di utilizzare le principali interfacce disponibili tra cui la porta VGA, l'uscita audio, l'interfaccia PS/2, la porta seriale. Le funzionalità mostrate dalla demo sono l'elaborazione di immagini in tempo reale e le capacità di riprogrammazione della FPGA con una nuova configurazione.
Vediamo come funziona: la scheda richiede una tensione di alimentazione a 5 V, che può essere derivata direttamente dalla rete elettrica utilizzando l'alimentatore fornito a corredo con il kit. Prima di connettere l'alimentatore assicurarsi che lo swicth di potenza sulla scheda (in alto a sinistra nella **Fig. 3**) sia in posizione OFF e che lo switch *Suspend* (in alto a destra) sia in posizione RUN.
Per controllare l'esecuzione della demo, a seconda della interfacce che abbiamo a disposizione possiamo scegliere una delle seguenti modalità:

- mediante switch e pulsanti della scheda, visualizzando menu e messaggi di stato su un monitor VGA;
- mediante switch e pulsanti della scheda, visualizzando menu e messaggi di stato sul suo display LCD;
- mediante la porta seriale RS232 connessa

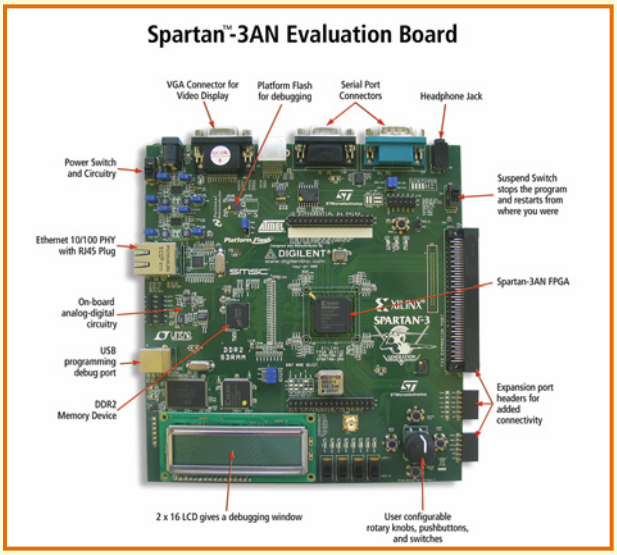


Fig. 3 *- Un kit di sviluppo per Spartan-3AN.*

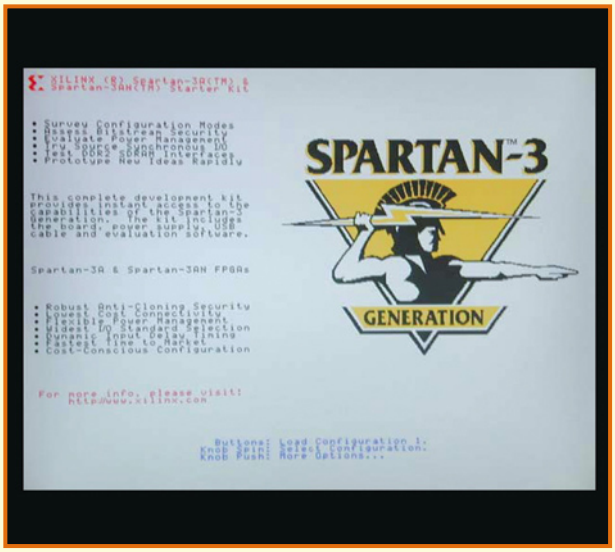


Fig. 4 *- Controllo della demo mediante uscita VGA.*

ad un PC ed utilizzando un terminale remoto quale, ad esempio, HyperTerminal.

All'accensione, un messaggio di benvenuto viene inviato sul display LCD, sulla porta VGA e sulla linea seriale. Nella **Fig. 4** è mostrata ad esempio l'immagine visualizzata sullo schermo VGA; in basso è presente il menu che consente di controllare la demo, il quale riporta il significato dei controlli derivanti dalla manopola e dai pulsanti (**Fig. 3** , in basso a destra).
La prima voce che appare seleziona la riprogrammazione della FPGA; come indicato nel menu, la configurazione da riprogrammare può essere selezionata tra le 4 disponibili ruotando la manopola mentre la riprogrammazione può essere avviata premendo uno qualsiasi dei pulsanti. Premendo, invece, la manopola, si scorrono le voci successive del menu della demo. Sono previsti le seguenti voci:

- scroll e rotazione dell'immagine; ruotando la manopola si osserva ruotare l'immagine sullo schermo mentre premendo i pulsanti la si fa scorrere;
- scroll e scaling dell'immagine; ruotando la manopola si ingrandisce o riduce l'immagine sullo schermo mentre premendo i pulsanti la si fa scorrere;
- autopilot : la demo entra in esecuzione automatica ed un messaggio è riprodotto sull'uscita audio.

Multi-boot
Torniamo alla prima voce del menu.
Come detto, possiamo in questo modo riprogrammare l'FPGA con una delle quattro configurazioni accessorie che sono previste a bordo della scheda. La riconfigurabilità delle FPGA è in effetti una caratteristica fondamentale in molti casi ed una peculiarità irrinunciabile; in questo modo un sistema può radicalmente modificare le sue funzionalità, pur utilizzando le stesse risorse.
Pensate all'infinità di possibilità che questo scenario apre in una applicazione reale e soprattutto alla semplificazione dell'hardware (con conseguente riduzione dei costi)!
Di seguito è riportata una breve descrizione delle funzionalità di ogni configurazione selezionabile.

- Configurazione 1: legge il DeviceDNA della FPGA (ovvero l'identificativo unico del dispositivo programmato in fabbrica e non modificabile) e lo visualizza sul display LCD a caratteri della scheda.
- Configurazione 2: genera immagini frattali che possono essere modificate (zoom o scroll) in tempo reale mediante manopola e pulsanti.
- Configurazione 3: implementa un terminale ASCII utilizzando la porta VGA e l'interfaccia PS/2 connessa ad una tastiera; in alternativa può essere controllato da remoto mediante la linea seriale.
- Configurazione 4: implementa un programmatore per la memoria flash parallela presente sulla scheda; controllabile mediante porta seriale, consente di cancellare e riprogrammare la memoria.

Durante il funzionamento di una qualsiasi di queste applicazioni è possibile sospendere l'attività della FPGA riportando lo switch *Suspend* nella posizione *Suspend*; se si hanno a disposizione dei multimetri è possibile connetterli ai circuiti di potenza della scheda che si trovano a destra in alto verificando così una riduzione della dissipazione di potenza. Occorre solo fare attenzione a non sospendere l'esecuzione della FPGA durante la programmazione della flash, perché il contenuto della memoria finirebbe per esserne corrotto!
I bitstream di programmazione delle FPGA corrispondente alla demo ed alle quattro

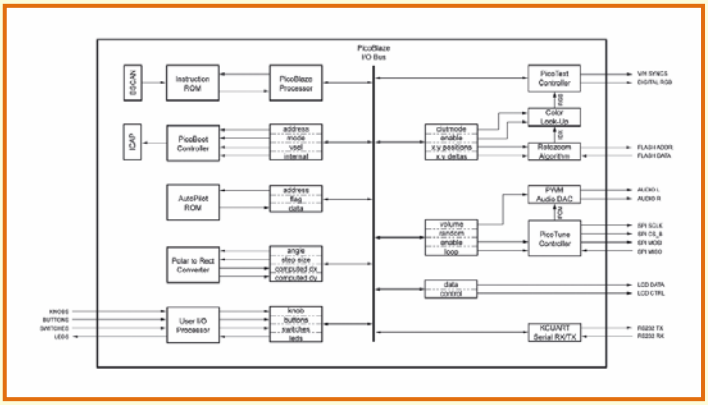


Fig. 5 *- Schema a blocchi della FPGA della demo dello starter kit.*

configurazioni sono memorizzati nella flash Atmel presente sulla scheda; i messaggi audio che vengono riprodotti sono memorizzati nella flash ST seriale, mentre in quella parallela sono presenti le immagini. La memoria Platform Flash, infine, include una configurazione che consente di eseguire un test dell'hardware. Nella guida utente del kit è indicato come modificare la configurazione dei jumper per consentire l'esecuzione del test. Nel caso in cui il contenuto di una delle memorie si dovesse corrompere è disponibile sul sito Xilinx *http://www.xilinx.com/products/boards/s3astarter/files/s3ask_out.pdf* la descrizione della procedura da seguire per ripristinare la configurazione di fabbrica.

L'FPGA della demo

Abbiamo visto in precedenza dei semplici esempi di quello che si può fare con una FPGA. Ma come possono realizzarsi queste applicazioni? Nella **Fig. 5,** a titolo di esempio, è riportato lo schema a blocchi della logica di controllo della demo del kit implementata all'interno FPGA.
PicoBlaze è un semplice microcontrollore a 8-bit programmabile in assembler caratterizzato da ridotta occupazione di risorse e che può essere utilmente impiegato per realizzare complesse macchine a stati, quale appunto il controllore della demo dello starter kit; riceve i controlli dai pulsanti della scheda e quindi attiva i moduli come richiesto. L'applicativo è memorizzato in BRAM. I moduli UART gestiscono la comunicazione su porta seriale. PicoText è un controller di porta VGA in grado di generare i segnali di controllo, PicoTune, invece, è il controller audio. PicoBoot, infine, è l'interfaccia che integra il modulo ICAP per gestire la riprogrammazione della FPGA (per ulteriori informazioni consultare la guida UG332 "Spartan-3 Generation Configuration User Guide" reperibile sul sito Xilinx). Due look-up table sono quindi utilizzate per memorizzare la sequenza di istruzioni preprogrammate che deve essere eseguita durante la riproduzione automatica della demo e la tabella di conversione da coordinate polari a coordinate rettangolari per implementare le funzionalità di scaling e rotazione dell'immagine. User I/O Processore include un decoder in quadratura ed i circuiti di anti-rimbalzo necessari per ricevere i comandi da manopola e pulsanti; integra inoltre l'interfaccia verso il display LCD a caratteri usato in alternativa alla porta VGA.
Il codice sorgente per tutti i moduli è disponibile sul sito Xilinx *http://www.xilinx.com/products/boards/s3astarter/reference_designs.htm* nella sezione "Demo Design for the Starter Kit Board".

CONCLUSIONI

In questa puntata abbiamo visto in dettaglio l'architettura delle FPGA Spartan-3 e presentato una scheda di sviluppo a basso costo ma dalle interessanti funzionalità, per i dispositivi della famiglia Spartan-3AN. Come accennato in precedenza, sul sito Xilinx sono disponibili molti altri esempi di riferimento per questa scheda oltre alla demo di base; due dei più interessanti sono l'implementazione di un web-server utilizzando un microprocessore proprietario di Xilinx che può essere implementato in FPGA e denominato MicroBlaze, o la realizzazione di un'applicazione di video editing che consente di disegnare su schermo un'immagine usando il mouse e salvarla sulla memoria flash. Purtroppo non abbiamo avuto ancora modo di affrontare alcuni dei concetti cui si fa riferimento in questi esempi che quindi potrebbero essere in certi punti non del tutto chiari. A partire dalla prossima puntata approfondiremo il discorso mostrando dapprima come realizzare una nostra semplice applicazione per FPGA in VHDL e poi sviluppando un sistema embedded basato su MicroBlaze. Pian piano si aprirà davanti a noi una nuova prospettiva nello sviluppo di sistemi digitali, del tutto configurabile in funzione delle nostre capacità ed esigenze.

Corso FPGA

Un primo progetto pratico da realizzare con il sistema di sviluppo per la serie SPARTAN-3AN Xilinx.

di
MARIANO SEVERI

Nella precedente puntata abbiamo visto in dettaglio l'architettura dei componenti FPGA della serie Spartan-3AN della Xilinx ed un interessante sistema di sviluppo che può essere utilizzato per imparare a progettare con i dispositivi logici programmabili di nuova generazione. Abbiamo anche provato ad eseguire alcuni degli esempi di riferimento che vengono forniti a corredo, acquistando così un po' di dimestichezza con questi nuovi dispositivi. Ora è giunto il momento di imparare a realizzare un semplice progetto partendo da zero.

Proveremo ad impiegare il nostro sistema di sviluppo come cronometro; utilizzeremo il display a caratteri per visualizzare il tempo ed i pulsanti per controllare il cronometro.
Purtroppo il display è piuttosto lento nell'aggiornare le informazioni, per cui nella misura del tempo abbiamo scelto come risoluzione 1 secondo; trattandosi di un semplice esempio dimostrativo per iniziare ad utilizzare i dispositivi FPGA, ciò è più che sufficiente. Tutti i file inerenti a questo progetto sono disponibili per il download.

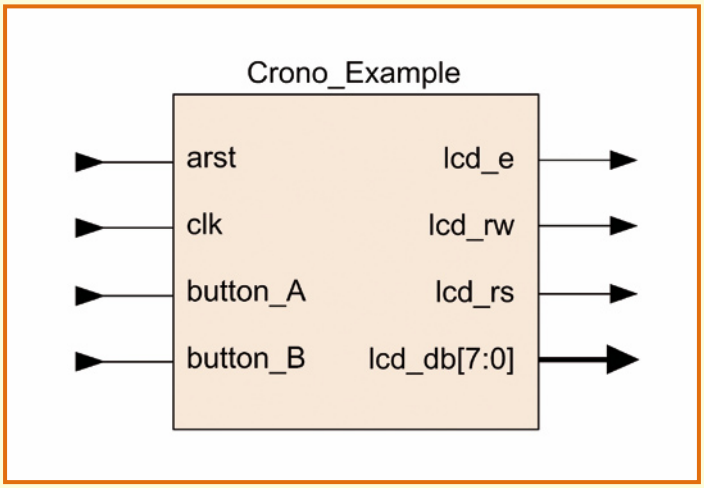


Fig. 1 - Un simbolo di alto livello per il nostro cronometro.

SCHEMA A BLOCCHI DEL CIRCUITO

Cominciamo col rappresentare il nostro progetto mediante il simbolo mostrato nella **Fig. 1**, che riporta i segnali di ingresso ed uscita. Il nostro esempio ha come ingressi il segnale di clock, di reset asincrono ed i pulsanti per gestire il cronometro; come uscite sono previsti invece i segnali di controllo/dati verso il display LCD.
Disegniamo poi un semplice schema a blocchi dell'architettura interna del dispositivo, come vedete nella **Fig. 2**.
In esso, **TimeCounter** è il contatore di ore, minuti e secondi, caratterizzato da una risoluzione di 1 secondo, dettata dal segnale di sincronismo **clk_ce** generato mediante un semplice prescaler, indicato in figura come **TimePrescaler**; il segnale di clock disponibile sulla scheda di sviluppo ha una frequenza di 50 MHz. I segnali **time_incr e time_rst** abilitano il conteggio ed azzerano, rispettivamente, il contatore. Sono generati dal modulo **CronoCtrl**, che implementa una semplice macchina a stati la cui evoluzione è dettata dagli ingressi **button_A** e **button_B**, come sarà descritto in seguito. **Debouncer** è un classico circuito di debouncing (antirimbalzo) che consente di rilevare la pressione ed il rilascio del pulsante; sono usati due moduli identici per gestire i pulsanti per il controllo del nostro cronometro. In corrispondenza di una variazione del tempo misurato, **TimeUpdate** genera il segnale che comanda l'aggiornamento del tempo sul display a caratteri. **LCDInterface** è l'interfaccia di controllo di questa periferica.

DESCRIZIONE IN VHDL

Il VHDL è un linguaggio utilizzato per descrivere i circuiti logici; qui lo utilizziamo per definire le funzioni del nostro progetto. Elementi fondamentali per una descrizione in VHDL sono le entità e le architetture. L'entità corrisponde al simbolo di più alto livello del nostro circuito e contiene la descrizione delle interfacce. Facendo riferimento al simbolo illustrato nella **Fig. 1**, l'entità nel caso del nostro contatore appare come riportato nel **Listato 1**.

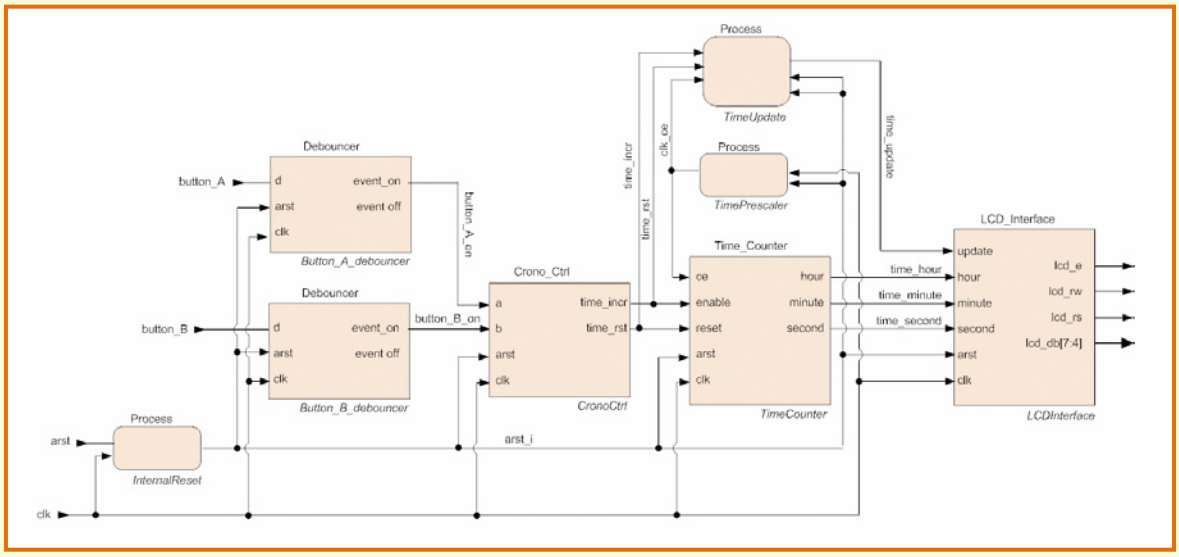


Fig. 2 - Una possibile realizzazione del cronometro.

Listato 1

```
entity crono_example is
port
(
-- clock and reset ---------------------------------
arst     : IN std_logic;
  clk      : IN std_logic;
-- control input -----------------------------------
  button_A : IN std_logic;
  button_B : IN std_logic;
-- LCD interface -----------------------------------
  lcd_e    : OUT std_logic;
  lcd_rw   : OUT std_logic;
  lcd_rs   : OUT std_logic;
  lcd_db   : OUT std_logic_vector(7 downto 0)
);
end entity crono_example;
```

Entity è la parola chiave del linguaggio che definisce l'entità e *crono_example* il nome che abbiamo dato al nostro progetto. L'istruzione **port** definisce l'insieme dei segnali di ingresso/uscita. Per ogni segnale sono specificate la direzione (ingresso od uscita) e il tipo. Il VHDL, in effetti, è un genere di linguaggio fortemente tipizzato; senza voler entrare nei dettagli, basti per ora osservare che *std_logic* è il tipo che serve per descrivere i segnali elettrici e può assumere, tra gli altri, i valori 0, 1 e Z per indicare i livelli logici, rispettivamente, basso, alto e tri-state. Il tipo *std_logic_vector* è un array di std_logic. Questi tipi non sono nativi del VHDL ma vengono dichiarati all'intero del package *std_logic_1164* della libreria *ieee*. Affinché il compilatore riconosca correttamente questi tipi, dobbiamo pertanto far precedere la dichiarazione dell'entità dalla definizione delle librerie e dei package che intendiamo collegare al nostro progetto. Ecco le istruzioni da aggiungere in testa al nostro file:

```
library ieee;
use ieee.std_logic_1164.all;
```

Se l'entità è la definizione delle interfacce di un modulo, l'architettura ne è l'implementazione e quindi, riprendendo la corrispondenza già evidenziata in precedenza, equivale allo schema a blocchi del nostro circuito. Nel **Listato 2** è riportata la descrizione dell'architettura del nostro cronometro.
In esso, *Architecture* è la parola chiave che definisce l'architettura, *struct* il nome che abbiamo dato alla corrispondente implementazione del nostro cronometro. La prima sezione dell'architettura fino all'istruzione *begin* è quella dichiarativa e contiene, nel nostro caso, la definizione dei segnali interni, ovvero quelli che consentiranno di collegare i diversi moduli.
La successiva sezione, che potremo chiamare assertiva, descrive invece quali moduli sono presenti all'interno del sistema e quindi come questi sono connessi. Per ognuno dei moduli usati, evidentemente, dovremmo aver già definito entità ed architettura da qualche altra parte; in seguito vedremo i dettagli.
Per istanziare un componente all'interno di un'architettura si usa una sintassi del tipo:

```
InstanceName : entity work.EntityName
port map
(
 port_name_0  => signal_name_0, …
)
;
```

dove *InstanceName* è il nome dell'istanza corrente, *EntityName* il nome dell'entità corrispondente (ovvero del modulo nella sua accezione astratta), *port_name* quello della porta dell'entità e *signal_name* quello del segnale ad essa connesso. All'interno di un sistema possono evidentemente essere presenti due istanze di uno stesso componente. Nel nostro caso, ad esempio, sono istanziati due circuiti di anti-rimbalzo per la gestione dei pulsanti; è sufficiente che le due istanze all'interno dell'architettura abbiano nomi diversi, pur facendo riferimento alla stessa entità.
Alla luce di questi pochi elementi, se riguardiamo la descrizione dell'architettura del nostro cronometro e lo schema a blocchi riportato nella **Fig. 2**, la maggior parte del codice che abbiamo riportato nel **Listato 2** per descrivere l'architettura del nostro cronometro ora dovrebbe risultarci piuttosto chiara.

PROCESSI IN VHDL E CIRCUITI COMBINATORI E SEQUENZIALI

Oltre a instanziare dei componenti, all'interno della nostra architettura sono anche descritte esplicitamente delle funzioni logiche, come ad esempio quelle per generare il segnale di TimeUpdate; altre saranno presenti nelle descrizioni dei moduli che compongono il nostro cronometro, come ad esempio la macchina a stati o la logica di interfaccia del display a caratteri. Vediamo come queste possono essere rappresentate.
In base alla descrizione che abbiamo dato

Listato 2

```
--=============================================================
architecture struct of crono_example is
--=============================================================
-------------------------------------------------------------
-- internal signal
-------------------------------------------------------------
  signal arst_i           : std_logic;
  signal clk_ce           : std_logic;
  signal time_incr        : std_logic;
  signal time_rst         : std_logic;
  signal time_hour        : std_logic_vector(7 downto 0);
  signal time_minute      : std_logic_vector(7 downto 0);
  signal time_second      : std_logic_vector(7 downto 0);
  signal time_update      : std_logic;

  signal button_A_on      : std_logic_vector(1 downto 0);
  signal button_B_on      : std_logic_vector(1 downto 0);
-------------------------------------------------------------

begin
-------------------------------------------------------------
-- Internal Reset
-------------------------------------------------------------
 InternalReset : process (arst, clk)
   variable v : std_logic_vector(3 downto 0);
 begin
   if arst='1' then
     v:=x"f";
   elsif clk'event and clk='1' then
     v(3):=v(2); v(2):=v(1); v(1):=v(0); v(0):='0';
   end if;
   arst_i <= v(3);
 end process InternalReset;
-------------------------------------------------------------

-------------------------------------------------------------
-- Controller
-------------------------------------------------------------
  ButtonA_Debouncer : entity work.debouncer
  port map
  (
   arst   => arst_i, clk => clk,
   d => button_A, event_on => button_A_On, event_off => open
  );

  ButtonB_Debouncer : entity work.debouncer
  port map
  (
   arst   => arst_i, clk => clk,
   d => button_B, event_on     => button_B_On, event_off => open
  );

  CronoCtrl : entity work.crono_ctrl
    port map
    ( arst => arst_i, clk => clk,
      A => button_A_on, B => button_B_son,
      time_incr => time_incr, time_rst => time_rst
    );
-------------------------------------------------------------

-------------------------------------------------------------
-- Time Prescaler
-------------------------------------------------------------
 TimePrescaler : process (arst_i, clk)
    variable cnt_v : integer;
  begin
    if arst_i='1' then
      cnt_v  :=0;
      clk_ce <= '0';
    elsif clk'event and clk='1' then
      if cnt_v=49999999 then             -- count 1 second
        clk_ce<='1'; cnt_v:=0;
      else
        clk_ce<='0'; cnt_v:=cnt_v+1;
      end if;
    end if;
  end process TimePrescaler;
-------------------------------------------------------------

-------------------------------------------------------------
-- Time Counter
-------------------------------------------------------------
  TimeCounter : entity work.time_counter
    port map
    ( arst => arst_i, clk => clk,
      ce => clk_ce, en => time_incr, reset => time_rst,
      hour => time_hour, minute => time_minute, second => time_se-
cond
     );
-------------------------------------------------------------

-------------------------------------------------------------
-- LCD Interface
-------------------------------------------------------------
 TimeUpdate : process (arst, clk)
 begin
  if arst='1' then
    Time_Update <= '0';
  elsif clk'event and clk='1' then
    if (time_incr='1' and clk_ce='1') then
      Time_Update <= '1';
    elsif time_rst='1' then
      Time_Update <= '1';
    else
      Time_Update <= '0';
    end if;
  end if;
 end process TimeUpdate;

 LCDInterface : entity work.LCD_Interface
   port map
   ( arst => arst_i, clk => clk,
     hour => time_hour, minute => time_minute, second => time_se-
cond,
     update => time_update,
     lcd_e => lcd_e, lcd_rw => lcd_rw, lcd_rs => lcd_rs, lcd_db =>
lcd_db(7 downto 4)
   );

  lcd_db(3 downto 0) <= x"f";
-------------------------------------------------------------
end architecture struct;
--=============================================================
```

inizialmente per il comportamento del nostro circuito, vogliamo che il segnale *TimeUpdate* sia attivo per un impulso di clock in seguito ad un'operazione di incremento (che avviene quando time_incr='1' e clk_ce='1') o di reset del contatore (corrispondente all'evento time_rst='1'). Come mostrato nella **Fig. 3**, quindi, il segnale può essere generato mediante un semplice flip-flop di tipo D il cui ingresso è dato da una funzione booleana dei segnali *time_incr*, *clk_ce*, e *time_rst*.
La sintassi per descrivere un flip-flop in linguaggio VHDL è la seguente:

```
flip-flop : process (arst, clk)
begin
  if arst='1' then
    q<='0';
  elsif clk'event and clk='1' then
    q <= d;
  end if;
end process flip-flop;
```

dove *arst* è il segnale di reset asincrono, *clk* il

clock e *d* e *q*, rispettivamente ingresso e uscita. Come si vede, abbiamo usato una nuova istruzione *process* che serve a definire i processi. Un processo non è altro che un contenitore per una serie di istruzioni che il simulatore esegue sequenzialmente; all'interno di un'architettura sono eseguiti diversi processi in parallelo, il che consente di rappresentare strutture concorrenti. In seguito avremo modo di vederne molti. La sintassi più semplice per definire un processo generico è del tipo:

```
process_name : process (sensitivity_list)
begin
-- sequential instruction
end process process_name;
```

dove *sensitivity_list* è l'insieme dei segnali in corrispondenza delle cui variazioni il processo viene eseguito.
Se ora torniamo alla descrizione del segnale *TimeUpdate*, in base a quanto appena detto, ci basterà scrivere la seguente parte di codice per avere il comportamento atteso:

```
TimeUpdateReg : process (arst, clk)
begin
  if arst='1' then
     Time_Update <='0';
  elsif clk'event and clk='1' then
     Time_Update <= (Time_Incr and Clk_ce) or
Time_Rst;
  end if;
end process TimeUpdateReg;
```

Qui è il caso di osservare che l'istruzione *q* <= *d* (che assegna l'ingresso all'uscita) può in genere essere sostituita da una serie di istruzioni sequenziali più complesse che contengono, ad esempio, costrutti if…then…else . Ciò consente di descrivere i circuiti combinatori che generano l'ingresso del nostro registro senza ricorrere direttamente alla espressione booleana, ma lasciando al tool di sintesi il compito di estrarla dalla nostra descrizione. Nel caso, ad esempio, del segnale *TimeUpdate* che stiamo considerando ed in accordo con la funzione logica che vogliamo realizzare, potremmo usare un costrutto del tipo:

```
if (time_incr='1' and clk_ce='1') then
 Time_Update <= '1';
```

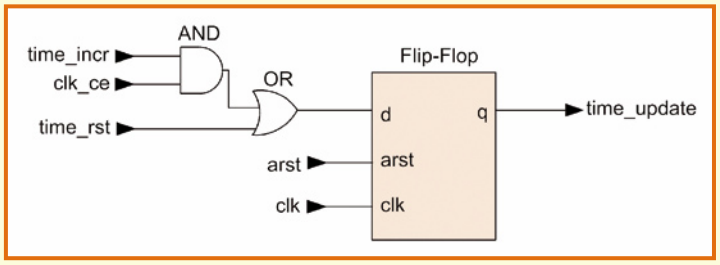


Fig. 3 *- Generazione del segnale TimeUpdate.*

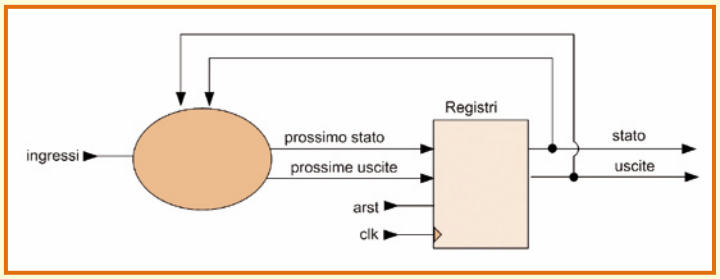


Fig. 4 *- Schema generico di una macchina a stati.*

```
elsif time_rst='1' then
  Time_Update <= '1';
else
  Time_Update <= '0';
end if;
```

DESCRIZIONE DI UNA MACCHINA A STATI

Una volta visto come instanziare dei moduli, in che modo connetterli e come creare dei processi per generare logica sequenziale e combinatoria, forse l'ultimo tassello che ci manca per avere in mano gli strumenti di base necessari a descrivere in VHDL un circuito elettrico è il modo per rappresentare una macchina a stati. Per colmare questa lacuna prenderemo ad esempio la descrizione della macchina a stati contenuta in CronoCtrl, che controlla il comportamento del nostro cronometro.
Prima di esaminarla in dettaglio, vediamo in generale come si può realizzare in hardware una macchina a stati. La **Fig. 4** ne mostra un esempio. La logica combinatoria descrive le equazioni di prossimo stato e prossima uscita, le quali determinano appunto il prossimo valore delle uscite e degli stati in funzione dello stato corrente e del valore attuale dei segnali di ingresso e di uscita. Quindi sono istanziati dei registri che rappresentano l'elemento di ritardo che consente di sincronizzare nella realtà il funzionamento della macchina a stati.

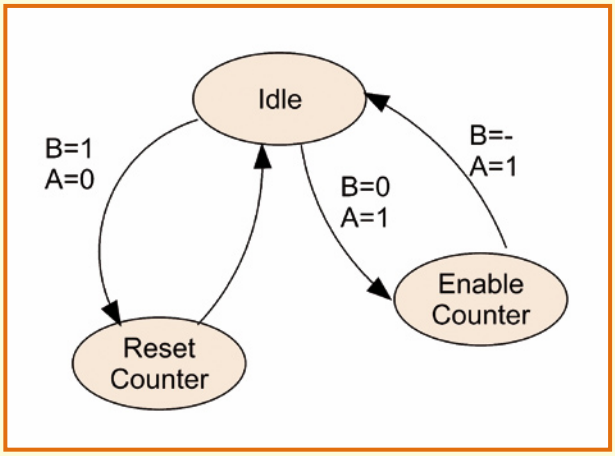


Fig. 5 *- Diagramma a bolle di CronoCtrl.*

Nella descrizione VHDL avremo quindi due processi: uno puramente combinatorio che rappresenta l'equazione di prossimo stato ed uno sequenziale che, invece, inferisce i registri; quest'ultimo abbiamo già visto come descriverlo. Nel nostro esempio il codice conterrà quindi un processo del tipo:

```
SMSeq : process (arst, clk)
begin
  if arst='1' then
    state_r   <= idle;
    time_incr <= '0';
    time_rst  <= '0';
  elsif clk'event and clk='1' then
    state_r   <= state_c;
    time_incr <= time_incr_c;
    time_rst  <= time_rst_c;
  end if;
end process SMSeq;
```

dove *state_r* è l'attuale stato della macchina (*state_c* il prossimo stato), *time_incr* e *time_rst* le due uscite (*time_incr_c* e *time_rst_c* le prossime uscite).
Per descrivere uno stato generico avremmo potuto usare un vettore di *std_logic*, definendo quindi esplicitamente, per ogni stato, la codifica. Il realtà il VHDL consente di definire dei tipi utenti tra cui quelli enumerativi che in questo caso trovano interessante applicazione. L'istruzione è la seguente:

```
type enum_type is (enum_list);
```

dove *enum_type* è il nome del tipo ed *enum_list* la lista dei valori che può assumere (separati mediante virgola). Durante la fase di sintesi, sarà il compilatore a trovare la migliore realizzazione di questo tipo in termini di bit e la migliore codifica per i diversi valori, consentendo così al progettista di risparmiare tempo. Una delle proprietà principali dei linguaggio di descrizione hardware, in effetti, è proprio la possibilità di portare la rappresentazione ad un livello più alto di astrazione che semplifichi il processo di progetto e consenta il riutilizzo di applicazioni già sviluppate.

Listato 3

```
SMComb : process (state_r, A, B)
 begin
 case state_r is

   when idle =>
     if B='1' then
       state_c <= reset_counter;
     elsif A='1' then
       state_c <= enable_counter;
     else
       state_c     <= idle;
     end if;
     time_incr_c   <= '0';
     time_rst_c    <= '0';

   when enable_counter =>
     if A='1' then
       state_c <= idle;
     else
       state_c <= enable_counter;
     end if;
     time_incr_c <= '1';
     time_rst_c  <= '0';

   when reset_counter =>
     state_c      <= idle;
     time_rst_c   <= '1';
     time_incr_c  <= '0';

 end case;
end process SMComb;
```

Torniamo alla nostra macchina a stati; nella **Fig. 5** ne è mostrato lo "schema a bolle". Seguendo quanto anticipato in precedenza, definiamo il tipo per descrivere gli stati e quindi i due segnali per il prossimo stato e lo stato corrente :

```
type State is (idle, enable_counter, reset_counter);
signal state_c    : state;
signal state_r    : state;
```

Infine scriviamo un processo per le equazioni di prossimo stato e prossima uscita usando

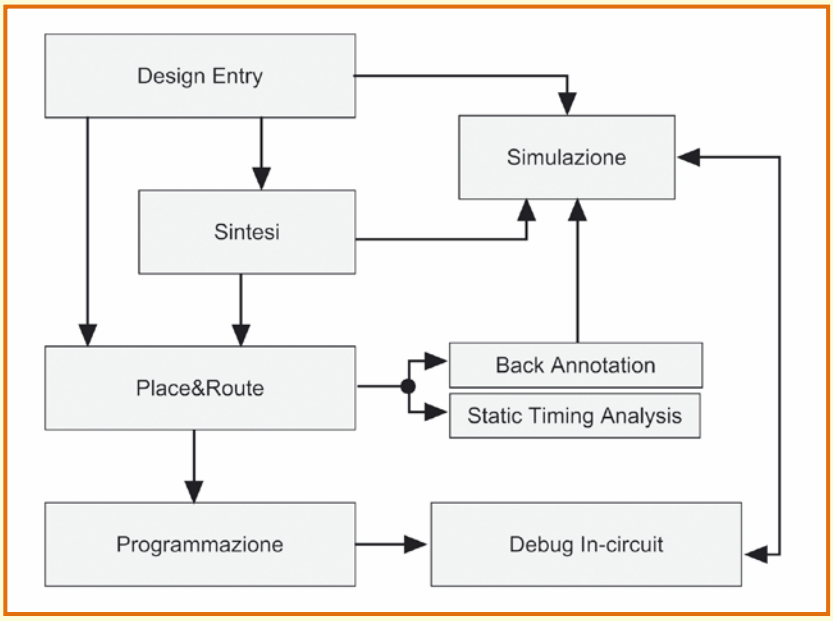


***Fig. 6** - Il flusso di progetto con FPGA.*

Graphics. Una versione free (*http://www.xilinx.com/ise/optional_prod/mxe.htm*) con alcune limitazioni relative al numero di linee di codice eseguibile simulabili ed al tipo di dispositivo supportato è già inclusa nell'ambiente di sviluppo per FPGA Xilinx ISE WebPack che descriveremo in seguito.
In alternativa, in Internet si possono trovare dei simulatori open-source tra cui GHDL (*http://ghdl.free.fr/*)

l'istruzione *case…when* come mostrato nel **Listato 3**.
Come si vede, viene dapprima definito (*case state_r is*) il segnale da valutare per selezionare le istruzioni successive da eseguire; quindi per ogni possibile valore del segnale (*when … =>*) viene definito l'insieme di queste. L'istruzione ritorna (*end case;*) non appena una delle condizioni viene eseguita.
Per avere chiara la corrispondenza tra quanto appena spiegato e la descrizione hardware, tenete sott'occhio il **Listato 3** e il diagramma a bolle visibile nella **Fig. 5**.

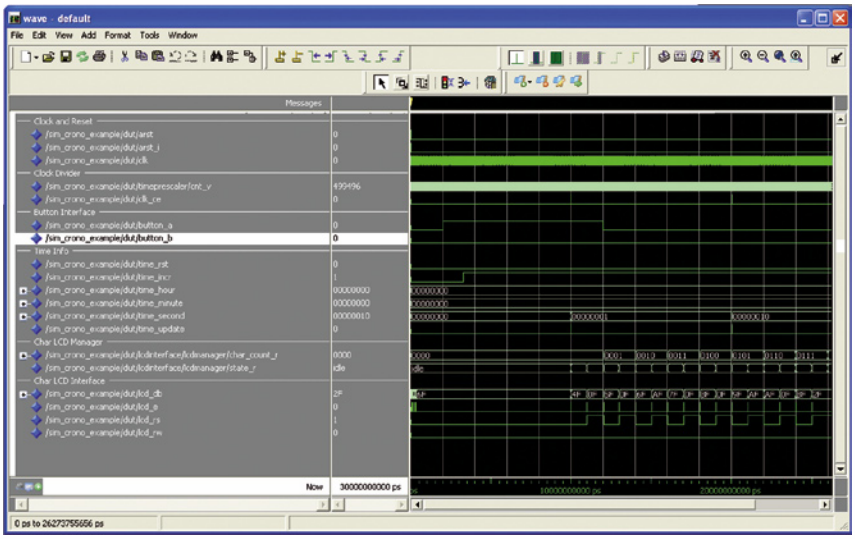


***Fig. 7** - Simulazione del nostro progetto in ModelSim.*

IMPLEMENTIAMO IL NOSTRO CIRCUITO

Nella **Fig. 6** è riportato come promemoria lo schema del flusso di progetto che viene seguito nella realizzazione di FPGA e che avevamo descritto nella prima puntata del nostro tutorial.

SIMULAZIONE

Dopo aver scritto il codice sorgente, dobbiamo evidentemente simularlo per controllare che risponda al comportamento atteso; allo scopo esistono diversi ambienti, tra i quali uno dei più diffusi è ModelSim di Mentor basato sulla catena di compilazione GNU.
Detto ciò, precisiamo che simulare un progetto vuol dire in pratica generare gli ingressi e verificare che le uscite seguano il comportamento atteso. Nel nostro caso significa generare i segnali di clock e reset per il contatore ed i controlli provenienti dai pulsanti. Il test-bench di simulazione che abbiamo creato (si veda il file **sim_crono_example.vhd**) definisce un modulo sim_crono_example che non ha porte di ingresso ed uscita ma è semplicemente un contenitore per il nostro progetto da simulare e per i processi che servono a definire l'andamento dei segnali di ingresso. Di seguito

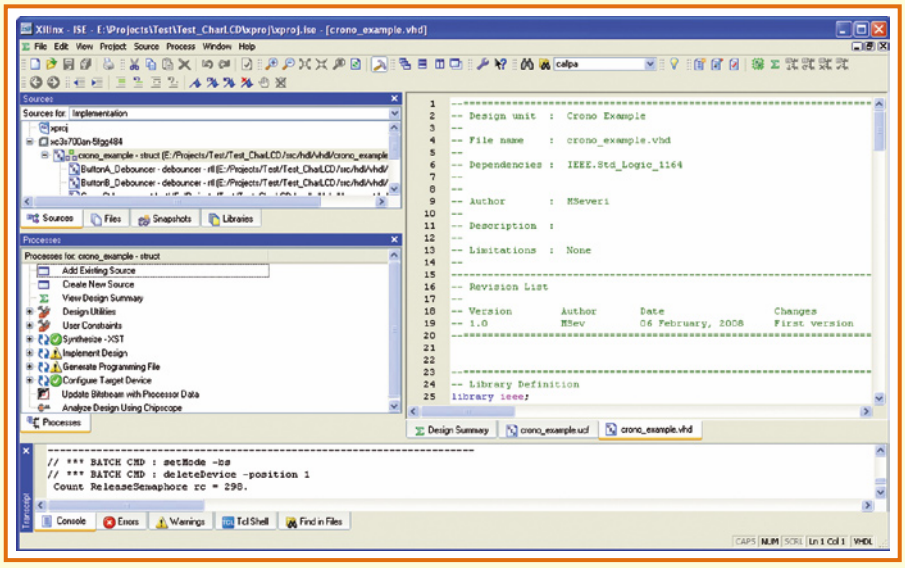


Fig. 8 *- L'ambiente di sviluppo ISE Foundation.*

è riportato come esempio il processo per la descrizione del segnale di clock :

```
ClkProcess : process
 begin
   clk <= '0'; wait for 10 ns;
   clk <= '1'; wait for 10 ns;
 end process ClkProcess;
```

Si noti che il processo non dispone di sensitivity_list; questo significa che viene rieseguito automaticamente al termine della lista di istruzioni contenute. L'istruzione *wait for … ns*, in particolare, è una semplice istruzione di attesa per il tempo specificato. Mediante questi semplici costrutti si riesce così a creare una forma d'onda con periodo 20 ns (frequenza 50 MHz) corrispondente al segnale di clock del nostro kit di sviluppo.
Nella **Fig. 7** è mostrato il viewer di forme d'onda di ModelSim al termine della simulazione; come si vede, in corrispondenza della pressione del pulsante A, la linea si attiva. La macchina a stati manda a livello alto la linea *time_incr* ed in corrispondenza dell'impulso di conteggio *clk_ce* il valore del contatore si incrementa. Vengono generati l'impulso *time_update* che richiede l'aggiornamento del display e, in seguito a questo, gli opportuni segnali di controllo per la periferica. Si noti che, per ridurre i tempi di simulazione, alcuni dei parametri del progetto (come ad esempio di tempi per la procedura di inizializzazione del display a carattere od il fattore di divisione del clock di sistema) sono stati ridotti.

SINTESI E PLACE&ROUTE

La fase successiva è quella di sintesi e place&route in cui la nostra descrizione in linguaggio VHDL viene tradotta automaticamente in porte logiche, che saranno associate alle risorse disponibili nel dispositivo selezionato; qui verranno definite le interconnessioni ed infine sarà generato il file di programmazione per l'FPGA.
ISE Design Suite è il tool proprietario di Xilinx che consente di gestire l'intero flusso. Nel kit di sviluppo è compreso il CD di installazione del software; la versione più aggiornata può essere scaricata direttamente dal sito Xilinx all'indirizzo: *http://www.xilinx.com/ise/logic_design_prod/webpack.htm.*
Tale versione WebPack è gratuita, ma presenta alcune limitazioni (in termini di funzionalità e tipi di dispositivi supportati) che tuttavia per la maggior parte dei progetti e soprattutto per il nostro caso, non hanno interesse. In alternativa può essere scaricata in valutazione – interamente funzionante per un periodo di 60 giorni – la versione completa a partire dall'indirizzo: *http://www.xilinx.com/ise_eval/index.htm*.
La **Fig. 8** mostra come appare l'ambiente di sviluppo. La finestra in alto a sinistra riporta i file inclusi nel progetto, mostrando la gerarchia di quest'ultimo; la finestra immediatamente sottostante illustra, invece, i processi che possono essere eseguiti. In basso è riportata una finestra di dialogo con indicazioni di messaggi ed errori oltre ad una shell Tcl; nel riquadro a destra un'area per finestre accessorie come ad esempio un editor di testo integrato.
L'utilizzo del tool non è molto diverso, almeno nelle linee di principio, dalla maggior parte degli ambienti di sviluppo hardware o software.
Dando per scontato che abbiate dimestichezza con uno di essi, di seguito riportiamo solo i

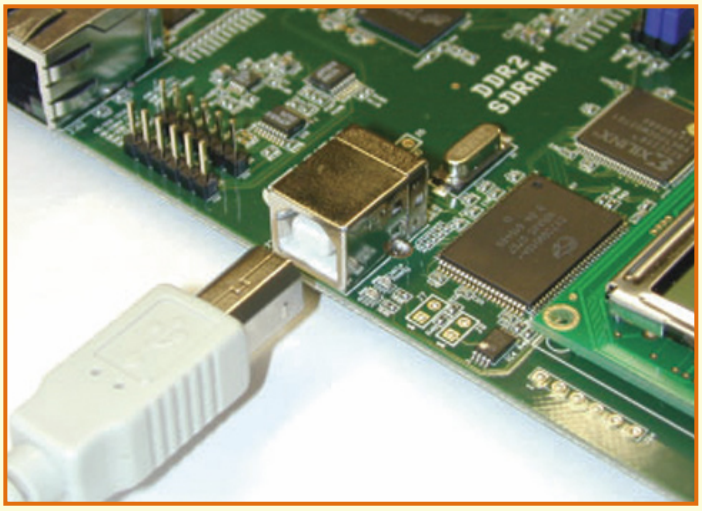


Fig. 9 - *Connessione del kit di sviluppo al PC per la programmazione della FPGA.*

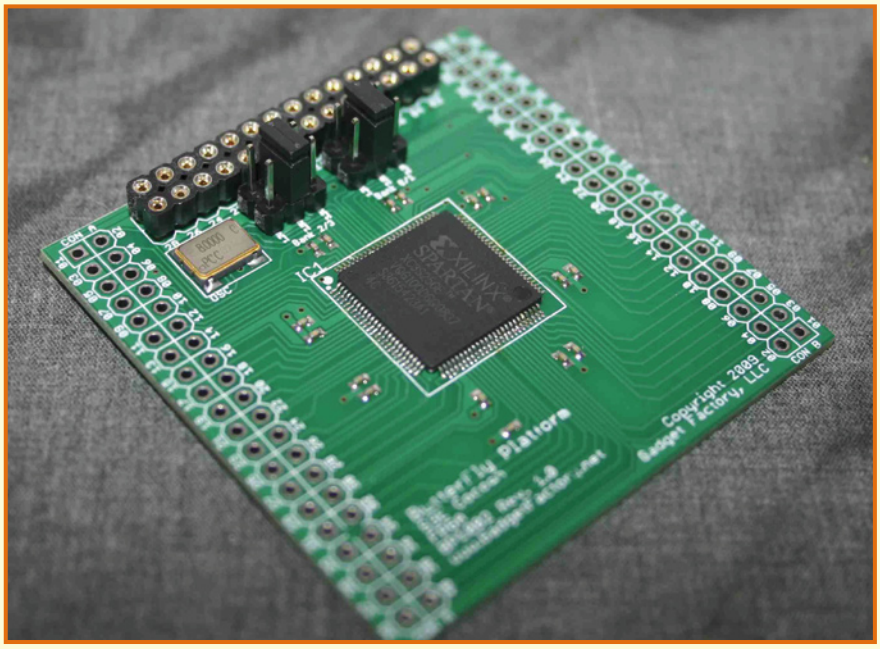
punti essenziali per arrivare in pochi passi alla generazione del file di configurazione della FPGA.
Iniziamo creando un nuovo progetto: dal menu *File* selezionate *New Project*; quindi nella finestra pop-up che appare, definite la locazione del progetto sul disco dove desiderate crearlo ed il nome che volete dargli.
La finestra successiva consente di selezionare il tipo di dispositivo da utilizzare; nel nostro caso definiamo i parametri corrispondenti alla FPGA del kit di sviluppo:

Family -> Spartan3A and Spartan3AN
Device -> XC3S700AN

Proseguendo, è possibile creare nuovi file da aggiungere al progetto o inserirne di esistenti; nel nostro caso salteremo la prima aggiungendo direttamente i file per la descrizione del contatore che abbiamo già scritto (e che si trovano nel download allegato all'articolo). Oltre a quelli di descrizione, è presente un file di constraint (*crono_example.ucf*) che consente di associare ogni segnale di uscita del nostro sistema ai pin fisicamente connessi alla scheda di sviluppo; le informazioni a riguardo possono essere ricavate dal manuale utente o direttamente dallo schema elettrico.
L'ultima finestra riporta un riassunto delle impostazioni che abbiamo selezionato. Confermando, se non abbiamo commesso errori, viene finalmente creato il progetto. ISE ritorna alla finestra principale che appare come mostrato nella **Fig. 8**. Selezionando *Implement top module* dal menu *Process*, viene avviato il processo di sintesi e place&route. Grazie ad una serie di script predefiniti, il flusso procede automaticamente e se, come dovrebbe, non si verificano errori, il processo termina con la generazione del bitstream. Accanto ad alcune delle icone che rappresentano le diverse fasi, potrebbero evidenziarsi dei punti esclamativi che richiamano l'attenzione del progettista segnalando la presenza di warning. Nel nostro caso possiamo trascurare queste indicazioni.

PROGRAMMAZIONE DEL DISPOSITIVO E TEST

L'ultimo passo è la programmazione del dispositivo: connettiamo il kit di sviluppo al PC mediante il cavo USB a corredo, come mostrato nella **Fig. 9** e alimentiamo quindi la scheda. La prima volta ci verrà chiesto di installare dei driver per poter gestire la programmazione; sul kit di sviluppo è infatti presente un dispositivo con interfaccia USB che svolge tali funzionalità. È necessario consentire l'installazione per poter proseguire; nella maggior parte dei casi la richiesta di installazione viene ripetuta tre volte prima di restituire sulla barra degli strumenti di Windows l'indicazione che il nuovo hardware è stato installato correttamente ed è pronto per l'uso; ciò è abbastanza normale.
A questo punto selezioniamo in ISE l'opzione *Configure Target Device*; facendo clic sul pulsante destro del mouse, richiamiamo il menu di pop-up e quindi selezioniamo *run*. Viene aperta una nuova finestra nel riquadro a destra per eseguire l'applicativo *Impact*; per default dovrebbe essere selezionata la voce *Configure Device using Boundary-Scan (JTAG)*. Confermate per procedere oltre. Tramite la connessione USB, il software esegue la scansione dei dispositivi presenti sulla scheda e riconosce l'esistenza di una FPGA Spartan-3AN. Richiederà quindi di associare al dispositivo il file di programmazione; selezionate, allora, *crono_example.bit*, presente all'interno della directory del progetto ISE. Nel caso in cui la procedura non venisse eseguita automa-



le memorie non volatili del kit sono già programmate con dati necessari all'esecuzione della demo che abbiamo descritto nella precedente puntata; riscrivendo una di queste difficilmente sarà possibile eseguirla nuovamente a meno di non ripristinare correttamente il contenuto delle memorie sovrascritte. Già nella precedente puntata avevamo sottolineato questo aspetto ed indicato come, nel manuale utente e sul sito, siano riportate tutte le indicazioni ed i file necessari a tale procedura di ripristino.

ticamente, selezionate l'icona della FPGA che appare nella finestra, aprite il menu di pop-up cliccando con il tasto destro sul menu ed eseguite *Assign New Configuration File*. Infine, per iniziare la procedura di programmazione della FPGA, nel riquadro *Processes* presente sulla sinistra della finestra selezionate la voce *Program FPGA Only*. Se tutto è stato eseguito correttamente, siamo finalmente in grado di provare il nostro circuito.
Premendo il pulsante EAST sulla scheda, si inizia il conteggio; premendolo nuovamente lo si arresta. In questa condizione, premendo il pulsante WEST il cronometro viene azzerato. Sul display a carattere dovremmo vedere aggiornato costantemente, una volta al secondo, il tempo misurato.
Tenete conto che la procedura descritta prevede la programmazione della sola FPGA che, come chiarito nelle precedenti puntate, è un dispositivo volatile; quindi spegnendo la scheda di sviluppo la configurazione del dispositivo si perde e, per poter eseguire nuovamente il nostro esempio di riferimento, sarà necessario ripetere la procedura precedente.
In alternativa potete programmare una delle memorie non volatili presenti sulla scheda (compresa ad esempio la flash interna della FPGA) e configurare il kit affinché all'accensione l'FPGA carichi automaticamente da questa il bitstream di configurazione.
Nel manuale utente del kit sono spiegate in dettaglio le modalità su come procedere in questi casi. Fate solo attenzione al fatto che

CONCLUSIONI

In questa puntata abbiamo mostrato come è possibile realizzare un semplice progetto con FPGA partendo da zero usando il nostro kit di sviluppo. Chiaramente ci sono molti aspetti del problema che non sono stati affrontati ed altri sono solo stati accennati ma riteniamo che con le poche indicazione fornite e seguendo l'esempio di riferimento proposto, ci siano ampi margini per poter iniziare a lavorare realizzando una propria applicazione. Alcune cose saranno più chiare solo con l'uso; su altre avremo modo di tornare in futuro.
Come anticipato in precedenza, una delle caratteristiche più interessanti delle moderne FPGA, è la disponibilità di una collezione di blocchi funzionali (per i quali si usa il termine di IP – Intellectual Property) già sviluppati e che possono essere inseriti all'interno del proprio progetto. Alcuni sono disponibili gratuitamente, altri sono proprietari e distribuiti direttamente dai produttori di logiche programmabili o da terze parti. Tra questi moduli, vi sono ad esempio core di processori, UART, interfacce PCI, MAC Ethernet, moduli DMA, eccetera. Si può così realizzare all'interno della FPGA un vero e proprio sistema complesso, più o meno corrispondente a quello che qualche anno fa si faceva con una scheda intera! Nella prossima puntata vedremo un esempio di ciò basandoci sempre sul nostro kit ed usando l'ambiente di sviluppo Embedded Development Kit di Xilin che supporta il microprocessore proprietario MicroBlaze.

Corso FPGA

Realizziamo un SoC utilizzando il microcontrollore Microblaze della Xilinx e l'ambiente di sviluppo EDK.

di
MARIANO SEVERI

Fino ad alcuni anni fa le FPGA erano prevalentemente utilizzate per realizzare logica di interconnessione ("glue logic") tra altri dispositivi (ad esempio, per implementare la decodifica di indirizzo del bus di un microprocessore in accesso alle periferiche), in moduli di acquisizione dati (ad esempio, per controllare convertitori analogico/digitali esterni) o in sistemi di comunicazione a più elevato data-rate. Col tempo, però, la crescente complessità dei dispositivi ne ha favorito la diffusione in ambiti di applicazione diversi, dall'elaborazione numerica dei segnali, ai sistemi di trasmissione dati, agli apparati di modulazione e demodulazione. Grazie alla elevata capacità logica, infatti, si è riusciti ad integrare sempre maggiori funzionalità all'interno dei dispositivi fino a realizzare dei veri e propri System-On-Chip (SoC).
Adottando la definizione proposta dalla Dataquest nell'ormai lontano 1995, un SoC, come mostrato nella **Fig. 1**, è un dispositivo che integra un'unità di elaborazione dati (sia essa un microprocessore, un microcontrollore o un DSP), un'unità di me-

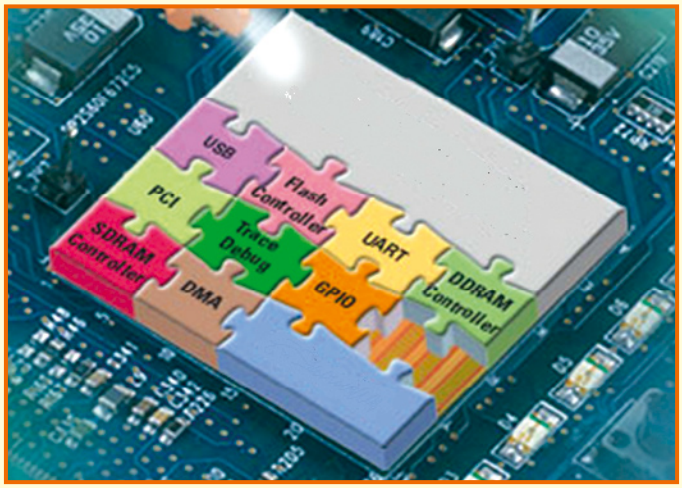


Fig. 1 - Un intero sistema in un singolo componente: i System-On-Chip.

moria on-chip e fino a 100.000 gate di logica configurabile in funzione dell'esigenze dell'utente. Le FPGA sono quindi piattaforme ideali per la realizzazione di SoC, grazie anche alla sempre maggiore diffusione di core IP.
Il termine IP è acronimo per Intellectual Property, ovvero proprietà intellettuale.
Un IP core è un circuito logico – descritto sotto forma di netlist o mediante linguaggi di descrizione hardware come il VHDL – a sé stante ed in grado di implementare una particolare funzione (ad esempio una porta di comunicazione UART) che, in alternativa, sarebbe realizzata da un componente dedicato. All'interno di una FPGA possono essere istanziati e connessi diversi IP, ognuno con diverse funzionalità, allo stesso modo in cui fino ad ora si sono integrati su uno stesso circuito stampato diversi componenti. Esistono dunque IP core per funzioni di vario tipo, che vanno da semplici interfacce di comunicazione (come UART, I²C, SPI), a controller di rete (Gigabit Ethernet), fino a moduli per la compressione delle immagini (encoder H264, ad esempio) o più in generale per l'elaborazioni numerica dei segnali. E vi sono anche core IP di microcontrollori più o meno complessi. Alcuni sono disponibili gratuitamente sul web, mentre altri sono soggetti a licenza. Actel, ad esempio, rilascia gratuitamente una versione del microcontrollore 8051 per le proprie FPGA e distribuisce una versione delle nuove ProAsic3 in cui una parte del dispositivo è già programmata per implementare una CPU ARM Cortex-M1. Xilinx ed Altera hanno invece sviluppato soluzioni proprietarie, i microcontrollori MicroBlaze e Nios-II. Ed è proprio del MicroBlaze che ci occuperemo in questa puntata, presentando una breve introduzione alla architettura del processore ed un esempio di System-on-Chip basato su di esso. Utilizzeremo per questo la scheda di sviluppo per FPGA Spartan-3AN che abbiamo descritto nelle precedenti puntate del corso.

IL MICROCONTROLLORE MICROBLAZE DI XILINX

La **Fig. 2** mostra uno schema di principio dell'architettura del processore MicroBlaze, un sistema RISC (Reduced Instruction Set CPU) a 32 bit con architettura Harvard caratterizzata da spazi di indirizzamento separati per le memorie dati e programma. Ogni area ha una dimensione massima di 4 GByte; il processore vi supporta accessi a word, half-word e byte. Le due aree possono essere sovrapposte all'interno di una stessa memoria, così da consentire il debug del software applicativo.
Il Register File consiste in 32 registri a 32 bit di tipo "general purpose"; sono poi previsti fino a 18 registri speciali.
L'intera architettura del processore può essere configurata in fase di sintesi mediante parametri generici, scegliendo le funzionalità necessarie alla propria applicazione e considerando quelle che sono le esigenze dell'utente in termini di prestazioni ed utilizzo di risorse interne; possono essere create fino a 70 diverse configurazioni!
La CPU, ad esempio, ha una architettura pipelined il cui numero di stadi può essere scelto tra 3 e 5. La più semplice configurazione in 3 stadi prevede le fasi di Fetch, Decode, ed Execute e assicura una capacità di calcolo dell'ordine di 0,95 DMIPs/MHz; è usata tipicamente per le FPGA di più piccole dimensioni, come i dispositivi della serie Spartan 3A DSP, dove raggiunge una massima frequenza di clock di 106 MHz. L'architettura in 5 stadi di pipeline utilizza invece i passi di Fetch, Decode, Execute, Access Memory e Writeback; nei dispositivi della serie Virtex-5 FXT raggiunge una frequenza di clock di 235 MHz. La capacità di calcolo è di 1,19 DMIPs/MHz
La maggior parte delle istruzioni termina l'esecuzione in un singolo ciclo di clock; nel caso di istruzioni multi-ciclo la pipeline entra in stallo, come ad esempio durante l'accesso a memorie lente. In questo caso, per ridurre la penalità il processore implementa un buffer di prefetch delle istruzioni. Alla ripresa da una condizione di stallo, la successiva istruzione

da eseguire viene letta da tale buffer.
Il MicroBlaze ha un insieme di istruzioni ortogonale, nel senso che ogni istruzione è in grado di operare su dati accessibili mediante uno qualunque dei modi di indirizzamento supportati; supporta istruzioni di tipo A, con due operandi sorgente ed uno destinazione di tipo registro, e di tipo B con un operando sorgente di tipo registro, l'altro di tipo immediato ed un singolo operando destinazione di tipo registro. L'insieme di istruzioni include operazioni di tipo aritmetico, logico, salto e load/store, oltre ad istruzioni speciali. Non sono previste, invece, istruzioni di Input/Ouput; infatti il processore usa uno schema di I/O di tipo memory-mapped. Dispone, come mostrato in **Fig. 2**, di tre interfacce dedicate per l'accesso alla memoria :

- Local Memory Bus (LMB), che è un bus sincrono per l'accesso alle memorie on-chip disponibili all'interno della FPGA con latenza di un singolo ciclo di clock;
- Processor Local Bus (PLB), ossia un bus sincrono non multiplexato, multimaster, con protocollo di tipo pipelined, come definito all'interno dell'architettura CoreConnect di IBM ed impiegato per la connessione al processore di periferiche veloci ed a bassa latenza;
- Xilinx Cache Link (XCL), che è un bus sincrono punto-punto di tipo single-master single-slave impiegato per la connessione a controller di dispositivi di memoria esterna (SDRAM, DDR, DDR2). Tale interfaccia è tipicamente impiegata per gestire i trasferimenti tra la memoria esterna e la cache interna del processore (ove questa sia stata abilitata in fase di configurazione del sistema).

Il processore supporta modalità di accesso in memoria di tipo reale o virtuale. In modalità reale è usato l'indirizzo effettivo per accedere direttamente alla memoria; in modalità virtuale invece, l'indirizzo effettivo viene traslato in un indirizzo fisico mediante l'unità MMU (Memory Management Unit). La modalità virtuale consente di rilocare dinamicamente in sistemi multi-task programmi e dati (il che permette di eseguire, ad esempio, programmi che richiedano quantità di memoria maggiore dello spazio indirizzabile dal processore) oltre a proteggere l'accesso alla memoria creando regioni private. La MMU del processore MicroBlaze (anche questa, qualora non sia necessaria, può essere disabilitata durante la fase di configurazione per ridurre l'occupazione di risorse del sistema) supporta dimensioni di pagina fino a 16 MByte e controllo indipendente per la traslazione e la protezione degli indirizzi per le memorie istruzioni e dati; la strategia di sostituzione della pagina è definita dal software di gestione ed è quindi dipendente dalla particolare applicazione.
Il MicroBlaze dispone inoltre di memoria cache separata per istruzioni e dati; la dimensione di tali memorie può essere configurata, al solito, mediante parametri generici. L'utilizzo di una cache istruzioni consente di migliorare le prestazioni nell'esecuzione di software applicativi residenti nello spazio di memoria esterno alla regione accessibile mediante bus LMB. La cache istruzioni è di tipo direct mapped (1-way associativa), con 4 o 8 word per linea. Cache con dimensioni di linea maggiori migliorano tipicamente le prestazioni nel caso di accessi sequenziali in memoria in quanto, in questo caso, nella cache viene precaricato

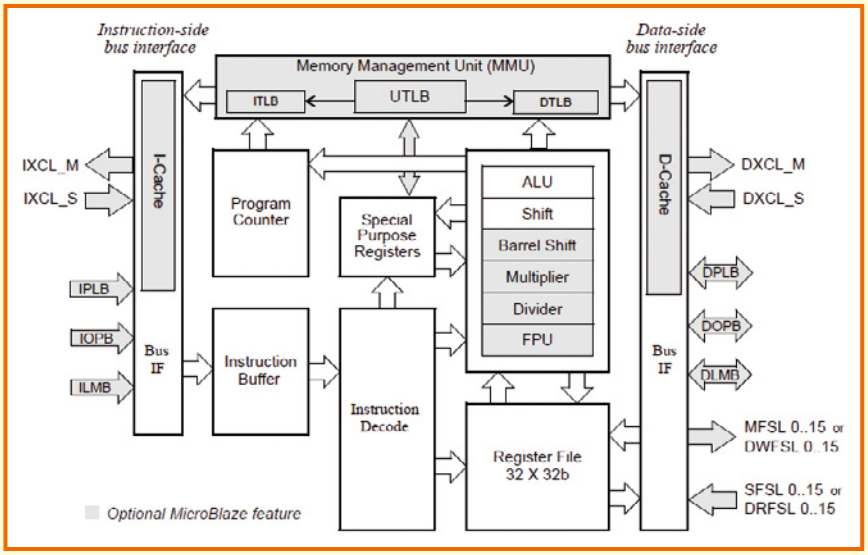


***Fig. 2** - Architettura del processore MicroBlaze.*

un numero maggiore di locazioni. Nel caso invece di applicativi che implementino uno schema piuttosto casuale di accesso alla memoria, l'impiego di cache con dimensioni di linea maggiore determina una frequenza di cache "miss" maggiore e quindi, probabilmente, un degrado della velocità di esecuzione del programma. È prevista la possibilità di abilitare dinamicamente la cache mediante accesso ad uno dei registri di configurazione della CPU, oltre alla disponibilità di istruzioni WIC per invalidare linee della memoria. La cache dati, analogamente a quella istruzioni, è di tipo direct-mapped (1-way associativa), 4 o 8 word per linea, di tipo write-through o write-back; possono essere associate alla cache solo aree di memoria non appartenenti allo spazio di indirizzamento riservato al bus LMB. Configurabili sono pure un Barrel Shifter ed un moltiplicatore in grado di eseguire le operazioni in un singolo periodo di clock, un divisore tra interi con latenza di 32 cicli ed una FPU (Floating Point Unit) per i calcoli in virgola mobile. Tale FPU è basata sullo standard IEEE 754, usa un formato in singola precisione (incluse le definizione di infinito, not-a-number e zero) ed implementa la modalità round-to-nearest; supporta operazioni di addizione, sottrazione, moltiplicazione, divisione, confronto, conversione e radice quadrata. La catena di compilazione inclusa nell'ambiente di sviluppo EDK descritto in seguito, supporta nativamente tali operazioni in singola precisione, che sono quindi tradotte in istruzioni FPU; nella realizzazione di filtri FIR, ad esempio, si ottiene una accelerazione dell'esecuzione dei programmi di un fattore fino a 42. Le operazioni in virgola mobile in doppia precisione sono invece emulate via software. Oltre ai bus per l'accesso in memoria descritti in precedenza il processore MicroBlaze dispone di fino a 16 porte FSL (Fast Simplex Link), che possono essere abilitate in fase di sintesi. Si tratta di interfacce punto-punto unidirezionali per comunicazioni di tipo data streaming. Il processore accede a tali interfacce mediante le semplici operazioni put e get, che istruiscono il trasferimento dati da un registro sorgente alla porta stessa e viceversa. Le istruzioni possono essere eseguite nelle diverse modalità blocking data, non-blocking data, blocking control, and non-blocking control. Le porte FSL sono tipicamente impiegate per connettere acceleratori hardware dedicati al processore.

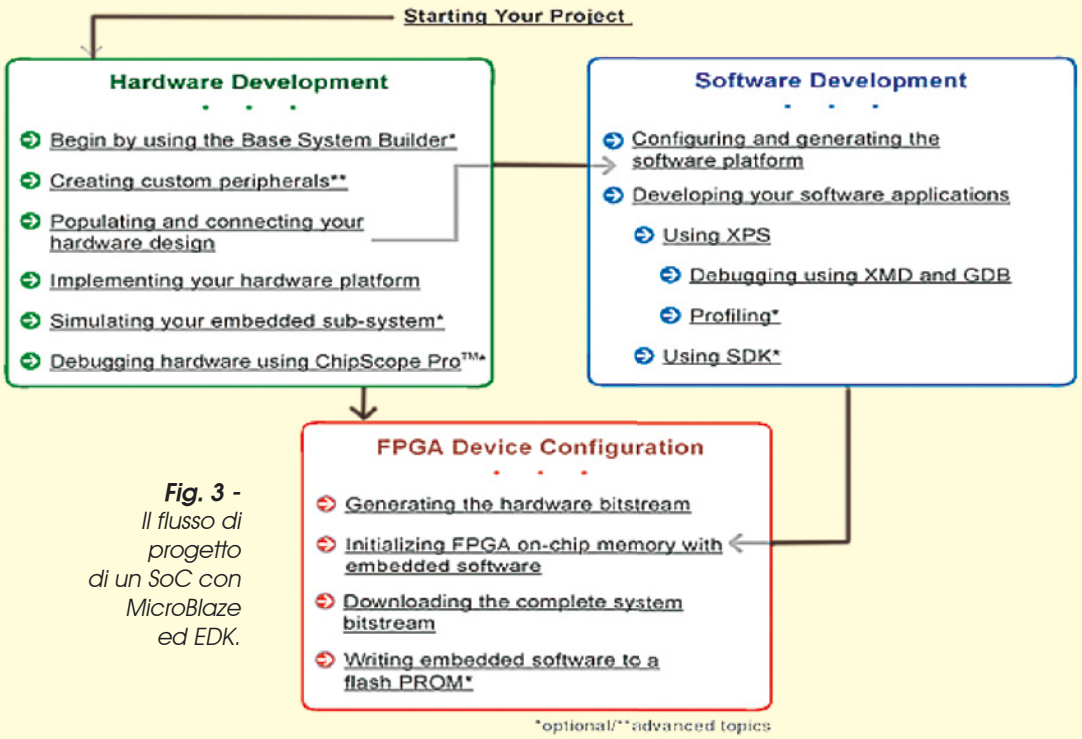


Fig. 3 - *Il flusso di progetto di un SoC con MicroBlaze ed EDK.*

Sono inoltre disponibili interfacce dedicate per scopi di debug e trace. L'interfaccia di debug, in particolare, si connette alla periferica MDM (MicroBlaze Debug Module) che consente l'accesso alla porta JTAG del dispositivo FPGA per la comunicazione con un comune tool BDM (Background Debug Module) residente su una macchina host; la periferica MDM è in grado di supportare architetture multi-processore. Tra le capacità di debug rese disponibili vi sono la configurazione di un numero limitato di breakpoint e watchpoint hardware e di un numero illimitato di breakpoint software, il controllo step-by-step dell'esecuzione del processore, l'accesso in lettura e scrittura alla memoria, alla memoria cache, ai registri generici e speciali (seppure con alcune eccezioni). L'interfaccia di Trace consente invece di esportare una serie di segnali di stato interni per scopi di monitoring ed analisi delle prestazioni.

L'AMBIENTE DI SVILUPPO EDK

La **Fig. 3** mostra uno schema di principio del flusso di progetto necessario per creare un SoC basato su microprocessore MicroBlaze. EDK (Embedded Development Kit) è l'ambiente di sviluppo fornito per gestire tale flusso. È contenuto nella distribuzione ISE Design Suite Embedded Edition; una versione di valutazione completamente funzionante per un periodo di prova di 30 giorni può essere scaricata previa registrazione dal seguente sito web: *http://www.xilinx.com/ise_eval/index.htm* . L'EDK contiene:

- XPS (Xilinx Platform Studio), l'ambiente grafico che permette di generare la configurazione hardware del nostro SoC selezionando da una ampia libreria di core IP le funzionalità necessarie alla propria applicazione;
- SDK (Software Development Kit), l'ambiente di sviluppo integrato con funzionalità avanzate e basato sulla piattaforma Eclipse per la creazione, la compilazione ed il debug del software applicativo.

Come al solito, il modo migliore per imparare ad usare un nuovo ambiente di sviluppo è forse quello di metterlo alla prova; senza indugi, vediamo quindi come realizzare la più classica delle applicazioni "Hello World!". Useremo come banco di prova la piattaforma di sviluppo per FPGA Spartan-3AN che abbiamo

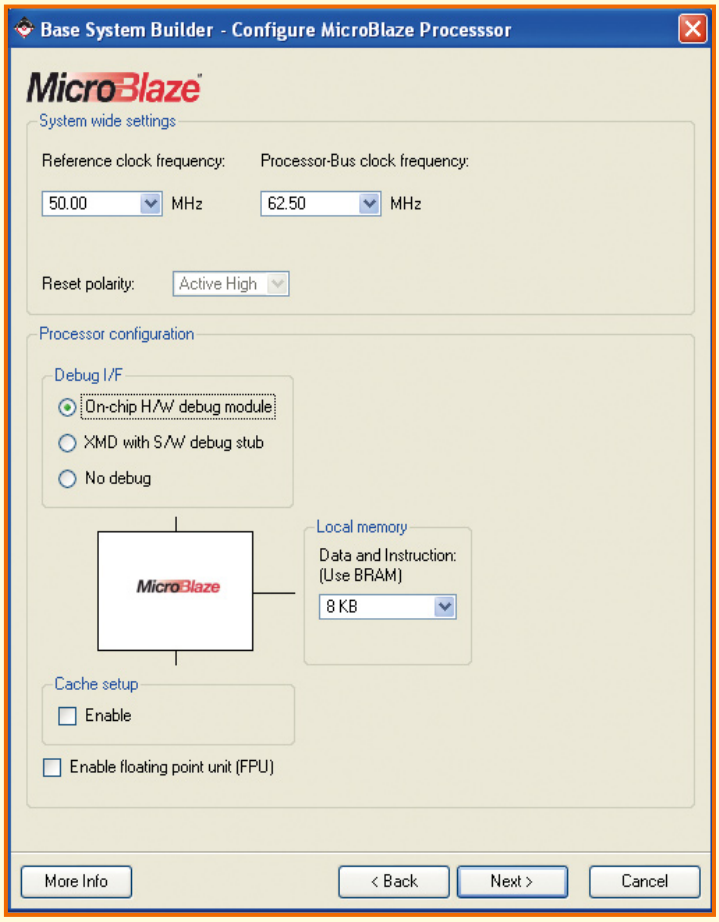


Fig. 4 *- La finestra del Base System Builder per la configurazione del MicroBlaze.*

presentato nelle precedenti puntate del corso con la versione 10.1 dell'EDK. Per brevità non potremo ovviamente vedere in dettaglio tutti gli aspetti del problema, ma le indicazioni riportate dovrebbero essere sufficienti a portare a termine in pochi passi il nostro progetto. Per una descrizione più dettagliata potrete consultare i manuali forniti unitamente all'ambiente di sviluppo, oltre alle diverse Application Notes che sono rese disponibili sul sito web di Xilinx (*www.xilinx.com*).

UN SEMPLICE ESEMPIO: HELLO WORLD!

Configurare un SoC per FPGA Xilinx basato su processore MicroBlaze e creare un applicativo di esempio è estremamente semplice ed immediato se si usa la funzionalità di Base System Builder. Dopo aver lanciato l'ambiente di sviluppo EDK, compare un pannello iniziale che ci richiede se vogliamo creare un progetto nuovo o aprirne uno esistente. Selezioniamo la voce Base System Builder Wizard (recommended) quindi indichiamo il nome

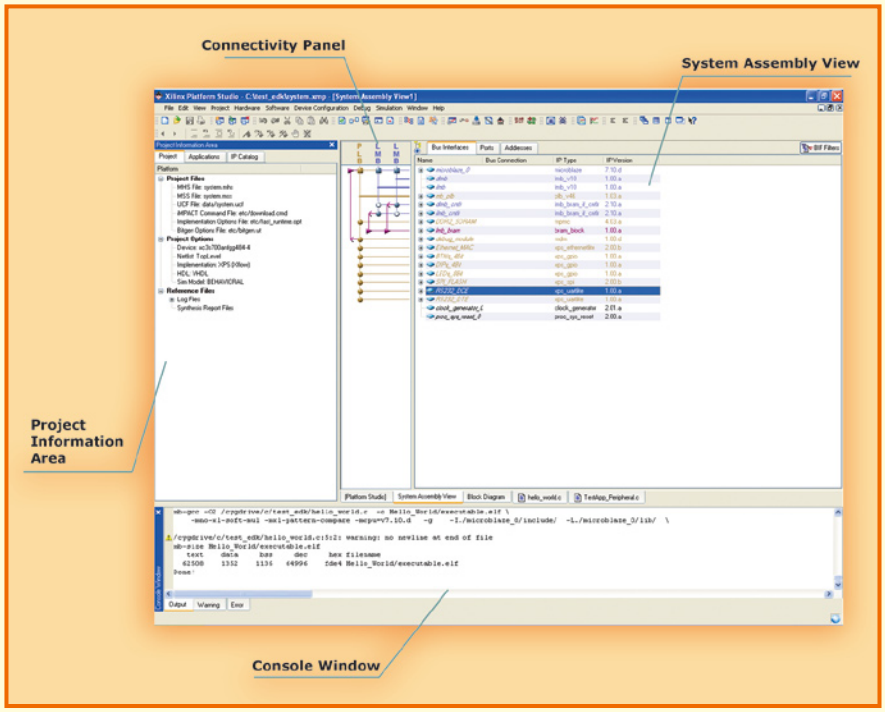


Fig. 5 - Il pannello principale dell'ambiente XPS.

del progetto ed eventualmente la directory di destinazione sul disco rigido del nostro computer. Si aprirà una finestra di benvenuto; richiediamo ancora una volta la creazione di un nuovo progetto. Quindi selezioniamo la nostra scheda scegliendo:

- Board Vendor : Xilinx;
- Board Name : Spartan-3AN Kit;
- Board Revision : D.

Dopo aver confermato la scelta, si aprirà una serie di finestre (del tipo mostrato in **Fig. 4**) che ci permettono di volta in volta di configurare le caratteristiche della nostra CPU e delle periferiche ad essa associate. È possibile abilitare, ad esempio, la cache del processore o la generazione della FPU; possiamo aggiungere al nostro SoC fino a due porte di comunicazione UART, un controller ethernet o una porta SPI per l'accesso alla memoria flash seriale presente sulla scheda di sviluppo. Nel caso del nostro semplice esempio, basterà in realtà lasciare inalterate le selezioni predefinite che ci vengono proposte. Dopo un po', dovremmo finalmente ritrovarci all'interno dell'ambiente XPS (Xilinx Platform Studio) come mostrato nella **Fig. 5**, con il nostro progetto correttamente creato e configurato. Nell'area System Assemby View, sulla destra del pannello, appare la lista dei moduli e delle periferiche che sono presenti nel nostro SoC; l'area immediatamente alla sua sinistra, indicata come Connectivity Panel, mostra invece i bus di connessione che sono stati previsti. Si noti, ad esempio, la connessione delle porte UART indicate come RS232DCE ed RS232DTE al bus PLB del processore. Cliccando sulla tab Addresses della System Assemby View, il contenuto cambia; viene mostrato lo spazio di indirizzamento associato ad ogni periferica. Nella Project Information Area presente sulla sinistra del pannello di XPS, sono invece riportate le informazioni di progetto. Cliccando ad esempio sulla tab IP catalog, comparirà la lista di tutti i core IP che sono disponibili e che potrebbero essere aggiunti al nostro progetto, laddove ve ne fosse la necessità. Per fare questo è sufficiente selezionare la periferica desiderata con il mouse e trascinarla sull'area System Assemby View. Sarà quindi necessario connetterla al bus del processore (si può fare facilmente utilizzando il menu a tendina che si apre cliccando sull'istanza del modulo all'interno della System Assemby View) e configurarne le caratteristiche (mediante l'apposita finestra che si apre facendo doppio click su questa stessa istanza). Una volta definita la configurazione hardware del nostro SoC, dobbiamo generare il bit-stream di configurazione della FPGA e creare l'applicativo. Selezioniamo quindi Hardware -> Generate Bitstream. Se il processo termina correttamente vedremo un messaggio **Done!** nella Console Window che si trova in basso all'interno del nostro ambiente. Tenete presente che questa fase di sintesi e place&route può durare anche qualche decina di minuti, a seconda della complessità del sistema che abbiamo configurato e delle prestazioni del nostro PC; abbiate quindi pazienza. Una volta creata la piattaforma hardware dobbiamo passare a creare il nostro applicativo. Per farlo possiamo usare le utility fornite all'interno dello stesso EDK oppure esportare il progetto verso SDK, l'ambiente di sviluppo basato su Eclipse cui si è accennato in precedenza. Per semplicità e facilità di uso adotteremo la prima soluzione. EDK include

un ambiente di compilazione per il processore MicroBlaze basato sulla catena GNU e che supporta codice scritto in linguaggio C e C++; include le librerie standard C, Math, GCC e C++. Selezioniamo dapprima Software -> Generate Libraries and BSPs per generare le librerie per la gestione delle periferiche ed il Board Support Package del nostro SoC. Quindi nella Project Information Area selezioniamo la tab Application. Troveremo già alcuni progetti software creati automaticamente dal Base System Builder per verificare il corretto funzionamento delle memorie del processore (TestApp_Memory) o delle periferiche accessibili (TestApp_Peripheral). Per creare una nostra applicazione è sufficiente selezionare Add Software Application Project. Si aprirà un popup dove inseriremo il nome del nostro progetto. Dopo aver confermato, il progetto apparirà tra gli altri all'interno della lista. Cliccando con il pulsante destro del mouse sulla voce **Sources**, aggiungiamo un nuovo file al progetto. Espandiamo quindi il menu a tendina sotto la stessa voce e clicchiamo due volte per aprire questo nuovo file. Editiamo il testo aggiungendo le poche righe di codice di seguito riportate:

```
#include "stdio.h"

int main (void) {
  printf("Hello world");
}
```

Infine salviamo il file. Quindi, dal menu di popup che si apre cliccando con il pulsante destro del mouse sul nome del nostro progetto all'interno della Project Information Area, selezioniamo "Mark to Initialize BRAMs". Infatti il MicroBlaze inizia ad eseguire dalla locazione 0x00 dopo il reset e nella configurazione creata automaticamente dal Base System Builder tale area è associata alla memoria embedded del processore. Selezionando "Mark to Initialize BRAMs" istruiamo il compilatore ad allocare in tale memoria le diverse sezione del software compilato. Dallo stesso popup menu, selezioniamo infine la voce **Build Project** per compilare il nostro applicativo e generare il file *elf*. Se non ci sono errori, nella finestra di Output comparirà il messaggio **Done!** con le indicazioni proprio delle dimensioni delle diverse sezioni (text, data, bss, dec) risultato della compilazione. Nel nostro caso non si pone alcun problema; nel caso di applicativi più complessi potremmo invece ricevere un messaggio di errore che ci indica che la memoria interna non è sufficiente a contenere tutte le sezioni. Tenerlo presente nel caso si sviluppino applicazioni proprie!
Avendo scelto di far eseguire il codice dalla memoria interna del processore, dobbiamo modificare infine il bitstream di configurazione della FPGA in modo che proprio questa memoria sia correttamente inizializzata. Selezioniamo dalla barra dei menu principali di XPS la voce Device Configuration -> Update Bitstream. A questo punto non ci resta che provare che il tutto funzioni. Connettiamo il nostro kit di sviluppo al PC mediante il cavo USB ed un cavo seriale; alimentiamo la scheda e apriamo sul PC un qualunque programma di gestione della porta seriale come Hyperterminal. Usiamo la seguente configurazione per la comunicazione : 9600 8-N-1. Da XPS selezioniamo infine dal menu principale Device Configuration -> Download Bitstream. L'FPGA viene configurata attraverso la porta USB. Se non si verificano errori, al termine del processo il nostro sistema inizierà a funzionare e dovremmo vedere sul nostro terminale apparire la scritta "Hello world!".

IL DEBUG DELL'APPLICATIVO

L'esempio che abbiamo visto è estremamente semplice. In una applicazione reale il nostro applicativo sarà più complesso e non è detto che funzioni correttamente da subito. Abbiamo bisogno quindi di poter fare il debug del codice che abbiamo scritto. La **Fig. 6** chiarisce le diverse configurazioni di debug che sono supportate. EDK include al suo interno il software XMD (Xilinx Microprocessor Debugger). Come si può vedere dalla figura, XMD controlla il processore target in modalità "hardware" oppure in modalità "software". Nel primo caso comunica con la periferica MDM descritta in precedenza e che si connette direttamente alla porta di debug del MicroBlaze; nel secondo, invece, è richiesto che il processore esegua il programma dedicato xmdstub che si può trovare nella lista della applicazioni importate automaticamente in ogni nuovo progetto. In entrambi i casi la

comunicazione avviene su linea seriale o cavo JTAG. XMD rende poi disponibile una interfaccia TCL (Tool Command Language) che può essere usata per controllare il debug da riga di comando eventualmente mediante script. In alternativa supporta il controllo da remoto mediante il protocollo TCP definito da GBD, il debugger GNU. In questo secondo caso, per i debugger può essere usata un'interfaccia grafica. Il debugger può risiedere sulla stessa macchina dove è eseguito XMD (questa macchina è fisicamente connessa alla scheda target) oppure su una macchina remota. Nel nostro esempio, il Base System Builder, come possiamo vedere dalla finestra di System Assemby View del progetto, ha creato una configurazione del sistema che impiega la periferica MDM; per il debug useremo quindi la modalità hardware. Selezioniamo dalla barra dei menu il comando Debug -> XMD Debug Option… e verifichiamo che sia selezionato il tipo di connessione corrispondente. Nella Project Information Area, all'interno della sezione Applications, con il pulsante destro del mouse selezioniamo l'applicazione microblaze_0_bootloop e scegliamo l'opzione Mark to Initializa BRAMs. Quindi eseguiamo i comandi Device Configuration -> Update Bitstream e (avendo verificato che la scheda target sia accesa e connessa al PC mediante il cavo di configurazione USB) Device Configuration -> Download Bitstream. In questo modo l'FPGA verrà configurata ed il processore eseguirà al power-up un programma che lo forza in uno stato busy, evitando che si verifichino eccezioni che potrebbero precludere la corretta comunicazione con il debugger. Lanciamo quindi XMD eseguendo il comando Debug -> Launch XMD… Si aprirà una finestra DOS con la shell di XMD; controlliamo che non siano riportati messaggi di errore che indichino l'impossibilità di stabilire la connessione con il target. Come debugger useremo quello integrato in EDK. Eseguiamo il comando Debug -> Launch Software Debug e selezioniamo la nostra applicazione dal menu a tendina che ci verrà presentato. Dopo aver confermato la selezione si aprirà infine la finestra principale. Dalla barra dei menu selezioniamo Run -> Connect to target e quindi Run -> Download. Il debugger si connetterà ad XMD e scaricherà il nostro applicativo nella memoria esterna del processore. Nella finestra principale sarà inoltre visualizzato il codice; come in ogni classico debugger è possibile introdurre dei breakpoint software in corrispondenza della esecuzione di una particolare istruzione. È possibile controllare quindi l'esecuzione del nostro applicativo passo-passo e visualizzare il contenuto dello spazio di memoria del processore e dei registri interni.

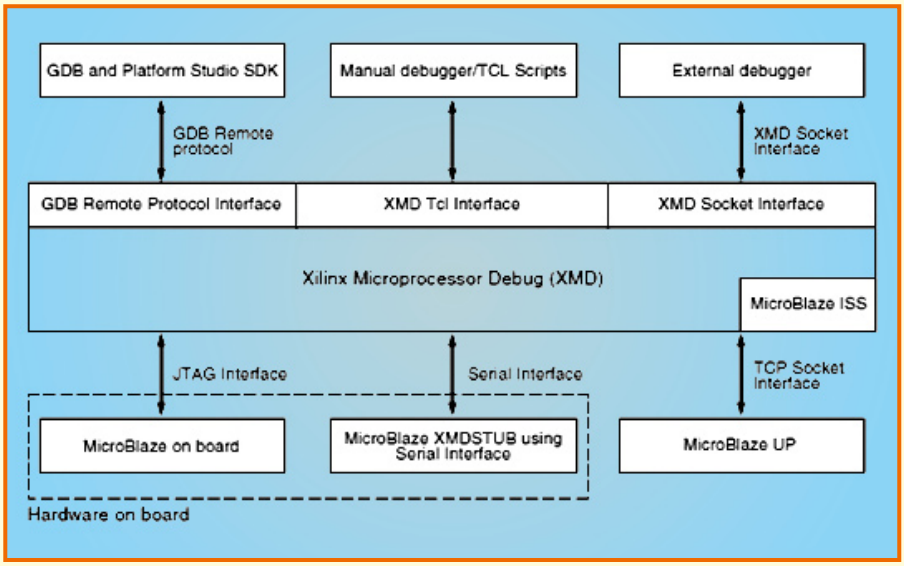


Fig. 6 - *Diverse configurazioni di debug per processore MicroBlaze.*

CONCLUSIONI

In questa puntata abbiamo mostrato come realizzare un SoC usando FPGA Xilinx. Utilizzando l'ambiente di sviluppo EDK e la libreria di core IP fornita con esso possiamo creare il microcontrollore che più si adatta alle nostre esigenze. La soluzione è corredata da un ambiente di sviluppo e debug del software applicativo del tutto autoconsistente.
Si chiude così il nostro corso introduttivo alle logiche programmabili. Ovviamente non abbiamo potuto trattare tutti gli aspetti in dettaglio ma speriamo che le informazioni fornite e gli esempi proposti abbiamo stimolato la curiosità del lettore verso un nuovo mondo dalla pressoché infinite possibilità.