M.M.Electronics - http://www.mmetft.it

Michele Marino - michele.marino@mmetft.it
Fabiano Boccolini - boccolinifabiano@gmail.com

Sistemi embedded x86 e processamento DSP tramite FPGA

V 0.1

Giugno 2008

INFORMATIVA

AVVERTENZE

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.
Gli autori declinano ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.
Si fa inoltre presente che quanto riportato viene fornito cosi com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.
Gli autori ringraziano anticipatamente per la segnalazione di ogni errore.

Michele Marino
Fabiano Boccolini

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

I sistemi embedded stanno diventando sempre più di largo uso in tutti i settori applicativi. I controlli industriali, le centraline delle automobili, i controlli di accesso si basano tutti sull'utilizzo di sistemi embedded. In questo articolo verrà descritto un sistema basato su architettura Intel x86 distrbuito dalla MESA Electronics. A tale sistema verrà interfacciata una FPGA (Field Programmable Gate Array) la quale permette di processare digitalmente segnali provenienti dal mondo esterno (DSP - Digital Signal Programming). La comunicazione tra la scheda CPU (Central Processing Unit) e la scheda che equipaggia l'FPGA avviene attraverso il bus PC/104, ovvero il bus ISA (Industry Standard Architecture) a 16 bit. In primo luogo verranno descritti tutti i componenti in dettaglio con particolare riferimento all'architettura del bus PC/104. Si farà riferimento anche al codice VHDL (VLSI Hardware Description Language - Very Large Scale Integration) necessario alla FPGA per la gestione dei dati sul bus nonchè per una applicazione di esempio per la gestione di segnali esterni.

2 Il sistema embedded x86-PC/104

In questa sezione si farà particolare riferimento alla scheda MESA 4C61A le cui caratteristiche sono di seguito elencate.

- sistema basato su architettura Intel 5x86
- controller video SVGA (Super Video Graphic Adapter) integrato
- 64MB di memoria RAM (Random Access Memory)
- memoria flash da 64MB integrata
- interfaccia IDE a 32 bit
- controller Ethernet 100baseT
- due porte seriali RS-232
- una porta seriale RS-485
- una porta parallela EPP/ECP/IEEE1284
- una porta PS2
- controller tastiera
- due porte standard USB

L'utilizzo del sistema necessita quindi di una tastiera standard a 101 tasti, un monitor e un cavo per il collegamento ad un PC host per il trasferimento del sistema operativo e dati nella memoria flash integrata. Per tale link viene utilizzato un cavo parallelo di tipo laplink a 4 bit insieme al software FastLynx v3.2. Nella figura 1 viene riportata la piedinatura del connettore a 25 poli (DB25) per la connessione tramite interfaccia parallela.

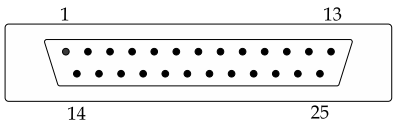

Figura 1: Piedinatura del connettore a 25 poli - DB25

La tabella 1 mostra la corrispondenza della piedinatura tra il connettore del PC host e il sistema embedded 4C61A.

Per chi non intendesse perdere tempo nella costruzione del cavo è bene sapere che è possibile acquistarlo già bello e

Connettore lato PC		Connettore lato sistema embedded	
segnale DB25	pin	pin	segnale DB25
Data Bit 0	2	15	Error
Data Bit 1	3	13	Select
Data Bit 2	4	12	Paper Out
Data Bit 3	5	10	Acknowledge
Data Bit 4	6	11	Busy
Acknowledge	10	5	Data Bit 3
Busy	11	6	Data Bit 4
Paper Out	12	4	Data Bit 2
Select	13	3	Data Bit 1
Error	15	2	Data Bit 0
Reset	16	16	Reset
Signal Ground	25	25	Signal Ground

Tabella 1: Connessione cavo laplink

pronto in qualsiasi negozio di elettronica. La figura 2 mostra invece la schermata principale del programma FastLynx dove sulla sinistra è presente la struttura del file system del PC host, mentre sulla destra è possibile selezionare il tipo di connessione con il sistema embedded. Una volta selezionato il tipo di connessione (nel nostro caso LPTX dove la X contrassegna il numero della porta), è necessario abilitare la connessione cliccando su "Connect as client". A questo punto sulla destra comparirà la struttura del file system del sistema embedded connesso al PC host. Da questo momento in poi è possibile sfruttare tutte le funzioni del file system standard ovvero, copia, eliminazione, taglio ecc. Per esempio, volendo trasferire un file presente sul PC host, basta cliccarci col tasto destro e successivamente su "copy" col tasto sinistro.

La scheda CPU comunica con la scheda DSP (FPGA) attraverso il BUS PC/104 la cui piedinatura viene riportata nella figura 3.

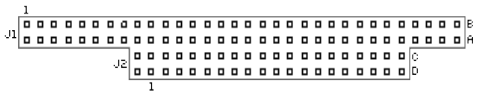

Figura 3: Piedinatura del connettore PC/104

Il BUS PC/104 funziona con un clock di sistema a 8MHz ed esiste sia nella versione a 8 bit che a 16. Le due versioni si distinguono per il fatto che in quella a 8 bit è presente solo il connettore J1 (figura 3). E' evidente che il microprocessore presente a bordo della scheda è in grado di fare varie operazioni tra le quali la lettura da un dispositivo esterno, la scrittura su un dispositivo esterno e l'esecuzione di operazioni interne che non implicano trasferimenti di dati con l'esterno. Affinché questo sia possibile è necessario avere a disposizione un set di segnali. I segnali del BUS possono essere raggruppati in tre gruppi:

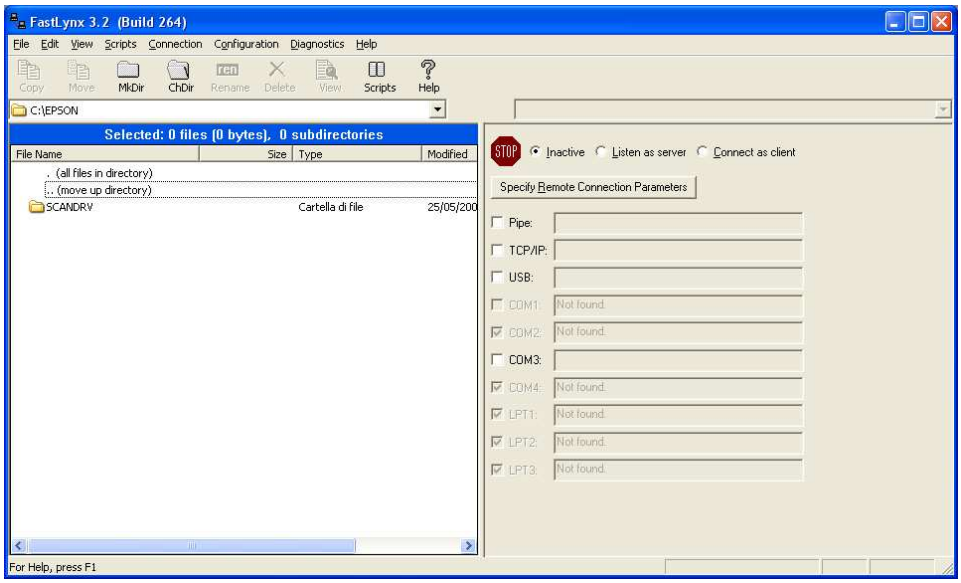

Figura 2: Schermata principale del software per il trasferimento dati

- segnali di indirizzamento
- segnali di controllo
- segnali per il trasferimento di dati

Il microprocessore utilizza il bus di indirizzamento per identificare il dispositivo esterno col quale comunicare. Le linee di indirizzamento sono 24 e vengono settate ogniqualvolta la CPU vuole comunicare con un dispositivo. Ogni dispositivo è dotato di un decodificatore di indirizzo che legge i livelli logici dal bus indirizzi e stabilisce se la CPU vuole comunicare con il dispositivo in esame e meno.

I segnali relativi al controllo del BUS consentono essenzialmente, di identificare il tipo di transazione (lettura, scrittura, ecc.) e, di sincronizzare i trasferimenti di dati con i dispositivi che lavorano e frequenze di clock diverse da quelle del clock di sistema.

I segnali del BUS dati servono appunto per consentire il trasferimento di dati tra la CPU e i dispositivi indirizzati. Nel sistema sotto analisi il BUS dati è a 16 bit con le linee raggruppate e gruppi da 8. Il BUS è di tipo bi-direzionale half-duplex ovvero, i trasferimenti in entrambi i versi (lettura e scrittura) avvengono in istanti di tempo diversi. La tabella 2 mostra la piedinatura del connettore con i nomi dei relativi segnali associati.

In genere viene fornita soltanto l'alimentazione a +5V per l'intera logica del sistema mentre le linee +12V, -12V, -5V vengono lasciate flottanti.

- Il segnale AEN (Address Enable) viene asserito quando è presente un controllo di tipo DMA (Direct Memory Access). Questo evita che un qualsiasi dispositivo, durante un trasferimento con accesso diretto alla memoria (DMA), risponda ai comandi inviati dal controllore DMA. Quando il pin AEN viene asserito basso il controllore DMA ha il pie-

Pin	J1/P1	J1/P1	J2/P2	J2/P2
Numero	riga A	riga B	riga C	riga D
0	–	–	0V	0V
1	$\sharp IOCHCHK$	0V	$\sharp SBHE$	$\sharp MEMCS16$
2	SD7	RESETDRV	LA23	$\sharp IOCS16$
3	SD6	+5V	LA22	IRQ10
4	SD5	IRQ9	LA21	IRQ11
5	SD4	-5V	LA20	IRQ12
6	SD3	DRQ2	LA19	IRQ15
7	SD2	-12V	LA18	IRQ14
8	SD1	$\sharp ENDXFR$	LA17	$\sharp DACK0$
9	SD0	+12V	$\sharp MEMR$	DRQ0
10	IOCHRDY	(KEY)2	$\sharp MEMW$	$DACK5$
11	AEN	$\sharp SMEMW$	SD8	DRQ5
12	SA19	$\sharp SMEMR$	SD9	$\sharp DACK6$
13	SA18	$\sharp IOW$	SD10	DRQ6
14	SA17	$\sharp IOR$	SD11	$\sharp DACK7$
15	SA16	$\sharp DACK3$	SD12	DRQ7
16	SA15	DRQ3	SD13	+5V
17	SA14	$\sharp DACK1$	SD14	$\sharp MASTER$
18	SA13	DRQ1	SD15	0V
19	SA12	$\sharp REFRESH$	–	(KEY)2 0V
20	SA11	SYSCLK	–	–
21	SA10	IRQ7	–	–
22	SA9	IRQ6	–	–
23	SA8	IRQ5	–	–
24	SA7	IRQ4	–	–
25	SA6	IRQ3	–	–
26	SA5	$\sharp DACK2$	–	–
27	SA4	TC	–	–
28	SA3	BALE	–	–
29	SA2	+5V	–	–
30	SA1	OSC	–	–
31	SA0	0V	–	–
32	0V	0V	–	–

Tabella 2: Piedinatura connettore BUS PC/104

no controllo del bus indirizzi e dei comandi di lettura/scrittura.

- Il segnale BALE (BUS Address Latch Enable) consente di campionare in dei latch l'indirizzo presente sul BUS indirizzi. Il campionamento avviene sul fronte di salita di BALE. L'indirizzo sul BUS di indirizzamento (SA[0..19]) sarà valido a partire dal fronte di discesa del segnale BALE fino alla fine del ciclo di BUS. Tale segnale viene forzato al livello logico alto durante i trasferimenti DMA.
- BCLK (BUS Clock) rappresenta il segnale di clock del BUS con duty cycle al 33%. La frequenza in genere varia tra 4.77MHz e 8MHz.
- DACKx (DMA Ackowledge) viene portato al livello logico basso per consentire una precedente richiesta di traferimento DMA sulle linee DRQ (DMA Request). Le linee vanno da 0 a 3 e da 5 a 7.
- DRQx (DMA Request) rappresentano dei canali asincroni per la richiesta, da parte dei dispositivi di I/O, di un trasferimento DMA. I canali da 0 a 3 servono per la richiesta di trasferimenti a 8 bit, mentre quelli da 5 a 7 per i trasferimenti a 16 bit. Il canale 4 viene utilizzato internamente dal sistema. La priorità di richiesta parte dal canale 0 e, a seguire, fino al canale 7.
- IOCS16 (I/O Chip Select 16 bit) asserito basso da un dispositivo slave quando viene indirizzato dal BUS master, indica un trasferimento I/O a 16 bit con uscite a collettore aperto.
- I/OCHCK (I/O Channel Check) viene asserito basso quando è presente un errore di parità su un dispositivo connesso al BUS di comunicazione.
- I/OCHRDY (I/O Channel Ready) viene asserito basso da un dispositivo di memorizzazione o un dispositivo di I/O per prolungare il ciclo di lettura/scrittura sul BUS. Esso dovrebbe essere tenuto basso per almeno 2.5ms.
- IOR (I/O Ready) viene asserito basso dal BUS master per indicare al dispositivo di I/O indirizzato di scrivere il proprio dato sul BUS dati (SD0..SD15).
- IOW (I/O Write) viene asserito basso dal BUS master per indicare al dispositivo di I/O indirizzato di leggere il valore presente sul BUS dati (SD0..SD15).
- IRQx (Interrupt Request) indicano una richiesta di interrupt. La priorità è stabilita come segue: IRQ 9(2),10,11,12,13,14,3,4,5,6,7.
- LAxx (Latchable Address) sono linee che si combinano con il BUS di indirizzamento basso al fine di avere una espansione dello spazio di indirizzamento a 24 bit (16MB di memoria indirizzabile).
- MASTER viene asserito basso dal dispositivo che aveva richiesto un trasferimento DMA (DRQx) e ne ha ottenuto il consenso attraverso il canale DACKx. In queste condizioni il controllore DMA gestisce il BUS di indirizzamento, il BUS dati e i segnali di controllo.

- MEMCS16 (Memory Chip Select 16) quando asserito basso indica un trasferimento di memoria a 16 bit.
- MEMR (Memory Read) quando asserito basso indica al controller di memoria di portare i dati sulle linee SD0..SD15. Il segnale rimane attivo basso per tutto il ciclo di lettura.
- MEMW (Memory Write) quando asserito basso indica al controller di memoria di memorizzare i dati presenti sulle linee SD0..SD15. Anche in questo caso il segnale rimane attivo basso per l'intero ciclo di scrittura.
- NOWS (No Wait State) viene utilizzato per ridurre il numero di cicli di attesa generati per default dal timer di sistema. In pratica, tale segnale serve per troncare in anticipo il ciclo di BUS evitando di inserire cicli di attesa.
- OSC (Oscillator) rappresenta un clock a 14.31818MHz con duty cycle al 50% per il refresh delle memorie dinamiche.
- REFRESH viene asserito basso quando la logica di refresh della memoria prende il controllo del BUS e diventa quindi il BUS master.
- SA0..SA19 (System Address) rappresentano le linee di indirizzamento del BUS che vengono latchate sul fronte di discesa di BALE.
- SBHE (System BUS High Enable) indica un trasferimento a 16 bit e in particolare che la parte alta del dato (byte alto) è presente sulle linee SD8..SD15. Questo segnale viene utilizzato anche per indicare un trasferimento a 8 bit utilizzando per il traferimento la parte alta del BUS (SD8..SD15).
- SD0..SD15 (System Data) rappresentano le linee per il trasferimento dei dati tra la CPU, la memoria e i dispositivi di I/O.

In definitiva, quando la CPU esegue un'operazione di lettura o scrittura inizia una sequenza di eventi identificata come ciclo di BUS. Durante il ciclo di BUS, la CPU pone l'indirizzo sul BUS di indirizzamento, setta le linee di controllo per indicare il tipo di transazione e trasferisce i dati tra il dispositivo selezionato e se stessa. Tutto questo avviene rispettando un ordine ben preciso, dove ogni passo avviene ad un certo istante di tempo all'interno del proprio slot temporale. Tutti i cicli di BUS sono costituiti almeno da due stati: il tempo di indirizzamento e il tempo dati. E' evidente che alcuni dispositivi sono ababstanza lenti tali da rendere necessario il prolungamento del ciclo di BUS e questo avviene attraverso il segnale READY.

I dispositivi di I/O vengono gestiti attraverso le seguenti porte:

- porta di comando: i dati scritti su questa porta corrispondono appunto, ai comandi impartiti al dispositivo. Ogni bit del dato scritto su tale porta controlla differenti aspetti operativi del dispositivo selezionato.
- porta di stato: i dati letti dalla porta in questione consentono di avere informazioni riguardanti lo stato del dispositivo di I/O associato. Anche in questo caso, ogni bit rappresenta diversi aspetti dello stato del dispositivo.

- porta dati: tale porta viene letta quando il dispositivo deve trasferire dei dati alla CPU. Viceversa, quando la CPU deve trasmettere dei dati ad un dispositivo utilizza la porta dati per la scrittura.

3 La scheda DSP 4i34M

L'FPGA presente sulla scheda 4i34M è una tipica FPGA volatile, che fa uso di una RAM statica (SRAM) per memorizzare la configurazione, ovvero le connessioni interne. Nel momento in cui viene fornita l'alimentazione e/o dopo un reset, l'FPGA deve essere nuovamente riconfigurata per eseguire il task richiesto. Tale configurazione è inviata tramite il bus PC/104 attraverso una sequenza di bytes. La locazione di I/O (o base address) dove questi bytes sono inviati è selezionata da due jumper presenti sulla scheda 4i34 stessa. La programmazione dell'FPGA avviene attraverso un PLD ATF20V8BQ (Programmable Logic Device) il quale riceve l'indirizzo selezionato dallo switch su due linee di I/O.

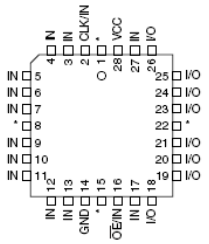

Figura 4: Pinout del PLD

La tabella 3 mostra le connessioni tra il PLD , il BUS PC/104 e l'FPGA.

L'I/O di configurazione usa due locazioni contigue partendo dall'indirizzo base (base address). Inoltre, il byte basso (SD0...SD7) del data bus (SD0...SD15) del PC/104 è bufferizzato sulla scheda e il buffer è controllato da due segnali:

- LBE (Low Byte Enable): segnale attivo basso che abilita il buffer;
- LBDIR (Low Byte Direction): determina la direzione del byte ovvero basso per i dati in uscita e alto per i dati in ingresso alla FPGA (tale segnale può essere pilotato dall'FPGA con segnali come IORD o SMEMR).

Il buffer è rappresentato dall'integrato 74HCT245 costituito da otto buffer tristate bidirezionali (figura 5).

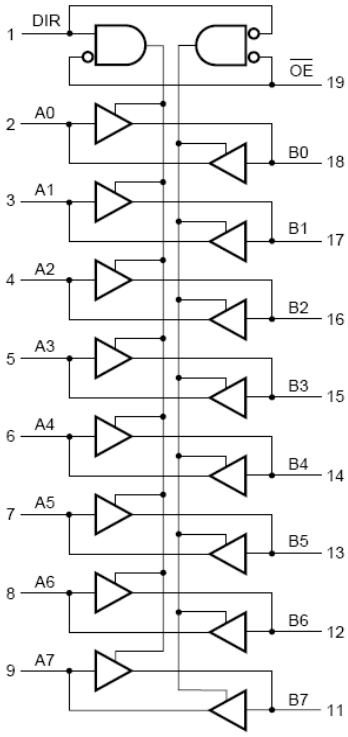

Figura 5: Configurazione logica del buffer tristate 74HC245

Come si può notare dalla figura 5, un livello logico basso sulla linea $\overline{OE}$ (/LBE)

μC		FPGA	PC/104		
tipo segnale	pin			tipo segnale	pin
NC	1			NC	–
CLK/IN	2		√	SA1	30
IN	3		√	SA2	29
IN	4		√	SA3	28
IN	5		√	SA4	27
IN	6		√	SA5	26
IN	7		√	SA6	25
NC	8			NC	–
IN	9		√	SA7	24
IN	10		√	SA8	23
IN	11		√	SA9	22
IN	12		√	AEN	11
IN	13		√	IOW	13
IN	14	√	√	GND	–
NC	15			NC	–
CE/IN	16	√		NC	101
IN	17	√		DONE/RESET	104
I/O	18			NC	–
I/O	19	√	√	jumper W5 (GND)	–
I/O	20	√	√	jumper W4 (GND)	–
IN	21	√		CCLK	155
NC	22			NC	–
I/O	23			NC	–
I/O	24			NC	–
I/O	25	√		NC	112
I/O	26		√	SD0	9
IN	27		√	SA0	31
VCC	28		√	+5V	29

Tabella 3: Piedinatura connettore BUS PC/104

μC		FPGA	PC/104		
tipo segnale	pin			tipo segnale	pin
DIR	1	$\surd$		LBDIR	84
DATA(0)	2	$\surd$		LD0	153
DATA(1)	3	$\surd$		LD1	146
DATA(2)	4	$\surd$		LD2	142
DATA(3)	5	$\surd$		LD3	135
DATA(4)	6	$\surd$		LD4	126
DATA(5)	7	$\surd$		LD5	119
DATA(6)	8	$\surd$		LD6	115
DATA(7)	9	$\surd$		LD7	108
GND	10	$\surd$	$\surd$	GND	–
DATA(0)	11		$\surd$	SD0	9
DATA(1)	12		$\surd$	SD1	8
DATA(2)	13		$\surd$	SD2	7
DATA(3)	14		$\surd$	SD3	6
DATA(4)	15		$\surd$	SD4	5
DATA(5)	16		$\surd$	SD5	4
DATA(6)	17		$\surd$	SD6	3
DATA(7)	18		$\surd$	SD7	2
$\overline{OE}$	19	$\surd$		/LBE	83
VCC	20	$\surd$	$\surd$	VCC	–

Tabella 4: Connessioni buffer 74HCT245

attiva le porte AND dopodiché il segnale DIR (LBDIR) consente di selezionare la direzione del trasferimento dei dati. La connessione del buffer è riportata nella tabella 4.

La programmazione dell'FPGA sulla scheda 4i34 viene eseguita tramite un file eseguibile DOS (SC4I34.EXE, il cui sorgente è disponibile sia in linguaggio C che in Pascal) che provvede all'invio della configurazione e la cui sintassi è la seguente:

SC4I34 FPGAFILE.BIN 220

dove 220 rappresenta l'indirizzo in esadecimale del dispositivo di I/O (4i34M). Tale comando usa un file di configurazione binario (estensione .bin) che può essere sia uno standard Xilinx BIT file sia un PROM format file derivabili dal tool ISE della Xilinx. La fase di configurazione così avviata prevede la scrittura di una serie di byte dal file al data register di configurazione della scheda, non prima di aver settato i bit dell'apposito registro di controllo (figura 6).

4 L'FPGA Xilinx

L'FPGA (Field Progammable Gate Array) presente sulla scheda di sviluppo 4i34M è una Xilinx XC2S200 da 200000 gate realizzata con processo a $0.18\mu m$. La tensione di alimentazione è di 2.5V con una frequenza massima di funzionamento di 200MHz. Il package è di tipo PQ208 con 140 linee di I/O. La programmazione dell'FPGA avviene trasferendo il file binario in una memoria statica. Il trasferimento può avvenire in modalità seriale o parallela. Nel caso in esame il trasferimento avviene serialmente mediante l'arbitraggio da parte del PLD presente sulla scheda 4i34M. L'inizializzazione della modalità di programmazione può avvenire in due modi:

- quando l'FPGA viene alimentata
- quando il pin $\overline{PROGRAM}$ viene asserito basso.

A questo punto l'FPGA pone i pin DONE e $\overline{INT}$ al livello logico basso indicando che il processo di cancellazione della memoria ha inizio. La fine dell'operazione viene indicata da un livello logico alto sul pin $\overline{INT}$. A questo punto è possibile trasferire lo stream di dati di configurazione. La figura 7 mostra l'andamento temporale dei segnali di programmazione. La fine della programmazione viene indicata da un livello logico alto sul pin DONE che rimane basso durante tutta l'operazione di programmazione. Quando DONE diventa alto l'FPGA diventa operativa ed è pronta per svolgere la funzione per la quale è stata programmata.

5 VHDL vs PC/104

Il VHDL necessario all'FPGA per la gestione dei dati sul BUS prevede:

- una fase di decodifica del base address (es.:220h) con AEN asserito a livello logico BASSO (non si lascia il pieno controllo del bus indirizzi e dei comandi di lettura/scrittura al DMA);
- l'asserimento a livello logico BASSO dell' IOCS16 per il trasferimento a 16 bit;
- il pilotaggio del buffer attraverso i segnali LBE e LBDIR (quest'ultimo controllato dal segnale IORD);

BASE ADDRESS CONFIGURATION DATA REGISTER

BIT	D7	D6	D5	D4	D3	D2	D1	D0
READ	CFGD7	CFGD6	CFGD5	CFGD4	CFGD3	CFGD2	CFGD1	CFGD0
WRITE	CFGD7	CFGD6	CFGD5	CFGD4	CFGD3	CFGD2	CFGD1	CFGD0

BASE ADDRESS+1 CONFIGURATION STATUS & CONTROL REGISTER

BIT	D7	D6	D5	D4	D3	D2	D1	D0
READ	XX	XX	XX	XX	XX	XX	XX	DONE
WRITE	XX	XX	XX	XX	LED	/WRITE	/PROG	/CS

Figura 6: Registri di configurazione

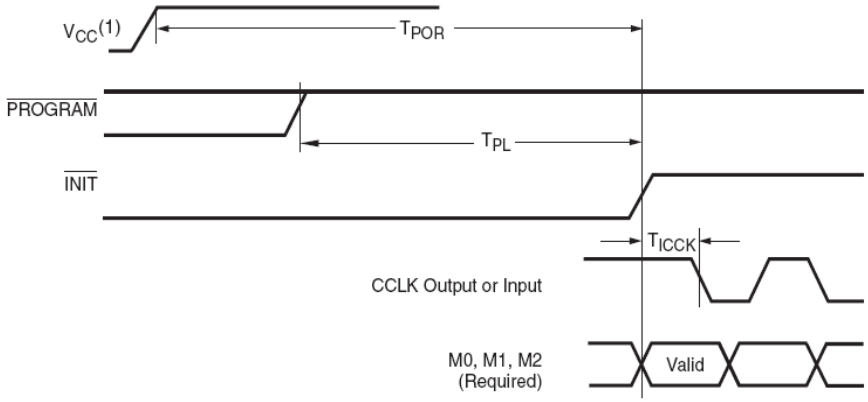

Figura 7: Temporizzazione dei segnali di programmazione

- il controllo dei segnali di IORD o IOWR rispettivamente per la scrittura o lettura dell'FPGA sul o dal bus;

oltre alla gestione vera e propria della lettura o scrittura a 16 bit dal bus.

Quindi l'entity relativa prevede essenzialmente:

- come input: il pin di reset, il clock, l'Addess Enable, l'I/O Ready, l'I/O Write, l'I/O Chip Select a 16 bit, il BUS Address;
- come input-output: il BUS Data;
- come output: il Low Byte Enable, il Low Byte Direction;

più tutti i segnali necessari al task richiesto.

Un'applicazione di esempio, riportata in seguito, potrebbe essere:

- la lettura, da parte dell'FPGA, di una word (16 bit) inviata sul bus PC/104 (SD0...SD15) dall'host CPU - in seguito di una scrittura tramite interfaccia in C - e mappatura del byte più (o quello meno) significativo sugli 8 leds predisposti sulla scheda 4i34M;
- la scrittura, da parte dell'FPGA, di una word (16 bit) sul bus PC/104 (SD0...SD15) e lettura da parte dell'host della word stessa presente sul bus.

Inoltre,è possibile:

- implementare la generazione di diverse frequenze da andare a rilevare sui pin dei due connettori di I/O presenti sulla scheda 4i34M (P2 e P3);
- implementare un dispositivo che si basa sul conteggio degli impulsi per eseguire una misura di frequenza (o di tempo), sulla base del clock di sistema (base dei tempi) e che riproduca sui leds la parte alta (o la parte bassa) del numero di impulsi (in formato binario) rilevati.

6 Dal codice sorgente all'implementazione

Il linguaggio di descrizione utilizzato è il VHDL e di seguito viene schematizzato, in maniera semplificata, l'iter progettuale da seguire per lo sviluppo del dispositivo digitale (8).

- Scrittura del codice sorgente: stesura del codice VHDL che descrive il dispositivo digitale da realizzare
- Simulazione (testbench): verifica comportamentale del dispositivo digitale descritto
- Sintesi: prende in ingresso una descrizione comportamentale (o RTL o data-flow) e da in uscita una descrizione strutturale (o rete logica o gate-level o netlist) per la mappatura nei logic blocks dell'FPGA
- Mappatura dei pin: gestione dei pin del package dell'FPGA
- Translate: prepara la netlist per il layout
- Mapping: assegna gli elementi logici agli elementi fisici che attualmente implementano le funzioni logiche del dispositivo (ovvero conversione della lista dei componenti base in una lista di componenti logici)

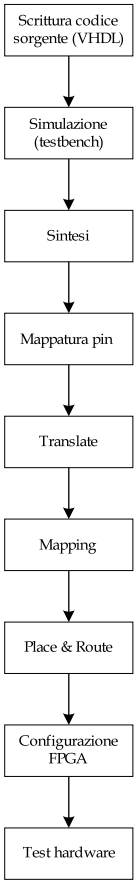

Figura 8: Flusso di progetto del sistema embedded

- Place&Route: assegna le celle logiche ad una specifica locazione dell'FPGA e realizza le interconnessioni (ovvero il layout per la programmazione il chip)

7 Breve introduzione a ISE

In questa sezione vengono riportati i passi necessari alla compilazione del codice VHDL sull'FPGA prescelta al fine di ottenere il file binario attraverso il quale programmare l'FPGA. La trattazione si riferisce all'ambiente di sviluppo ISE versione 9.2i della Xilinx. E' possibile scaricare la versione di prova di 60 giorni direttamente dal sito della Xilix [1]. Per creare un nuovo progetto è necessario eseguire i seguenti passi:

1. selezionare *File* > *New Project* ... (apparirà il *New Project Wizard* - figura 9):
   - editare il nome del progetto nel campo *Project Name* e cambiare, se necessario, il path in cui si intende salvare il progetto stesso (la subdirectory con il nome del progetto verrà creata automaticamente). Nella lista *Top-Level Source Type* selezionare *HDL*
   - selezionare *Next*
2. riempire le proprietà nella tabella che appare secondo quanto segue in figura 10:
   - selezionare la famiglia alla quale apaprtiene l'FPGA
   - L'FPGA utilizzata
   - il package del dispositivo

[1] www.xilinx.com

- la velocità
- il tipo di sintetizzatore utilizzato
- il tipo di simulatore utilizzato
- il tipo di sorgente

3. selezionare *Next* per passare alla schermata *Create New Source* nel *New Project Wizard* (figura 11)

4. selezionare ancora *Next* per passare alla schermata *Add Source* dove poter richiamare i file .vhd a disposizione

5. selezionando *Next* apparirà il *Project Summary*

6. selezionando nuovamente *Next* apparirà la lista dei file .vhd inseriti, quindi premere *OK*

7. selezionare *Synthesis /implementation* dalla drop-down list nella finestra *Sorces* (figura 12):
   - cliccare sul segno + affianco al gruppo di processi *Synthesize-XST*
   - doppio click su *Check Syntax* per eseguire un controllo della sintassi del codice VHDL
   - doppio click su *Synthesize-XST* per eseguire la sintesi

8. se tutto è andato a buon fine ci si troverà dinanzi la situazione di figura 13 (Il punto esclamativo con sfondo giallo indica la presenza di warnings - *tranquilli! Fanno parte del gioco*)

9. cliccare sul segno + affianco al gruppo di processi *User Constraints* (figura 14):
   - doppio click sul processo *Create Timing Constraints*: verrà eseguito il processo *Translate* presente tra i processi del gruppo *Implement Design* (viene anche eseguito il processo di sintesi che però viene saltato in quanto già eseguito nel passo precedente); verrà creato uno *User Constraints File* (UCF) e verrà visualizzato il messaggio di figura 15
   - scegliere *Yes* per aggiunger il file UCF al progetto

10. comparirà il *Constraints Editor*:
    - selezionare il tab *Global* e inserire *20ns* nel campo *Period* (50MHz è la frequenza di clock a cui deve operare l'FPGA)
    - cliccare *OK* (i constrantis inseriti saranno visualizzati nel tab *Constraints (read-write)*

11. selezionare *File>Save* e chiudere l'editor

12. doppio click sul processo *Assign Package Pins* per aprire lo *Xilinx Pinout e Area Contraints Editor* (PACE)

13. selezionare il tab *Package View*

14. nella finestra *Design Object List* eseguire la mappatura dei pin di I/O del sistema con i pin di I/O dell'FPGA (il file UCF verrà aggiornato con tale mappatura)

15. selezionare *File>Save*

16. selezionare *XSR Default <>* e cliccare su *OK*

17. chiudere *PACE*[2]

18. doppio click sul processo *Implement Design* nel tab *Processes*. Notare che, completata l'implementazione, i processi del gruppo *Implementation* hanno un check-mark verde indicante che essi sono stati portati a termine con successo senza errori (o warnings, molto raro! - figura 16)

19. cliccare sul segno + affianco al gruppo di processi *Generate Programming File* e selezionare le proprietà del gruppo *Generate Programming File* (figura 17):

    - porre come *Float* il valore degli *Unused IOB Pins* tra le *Configuration Options*

20. doppio click su *Generate Programming File*

21. caricare il tool *iMPACT* con un doppio click su *Generate PROM, ACE, or JTAG File*

22. selezionare *Prepare a PROM File*, quindi *Next*:

    - selezionare *Xilinx PROM* e BIN (tra i *PROM File Format*)
    - inserire un nome per il *PROM File Name* (p.e.: fpga)
    - cliccare su *Next*

23. in *Specify Xilinx PROM Device* selezionare *Auto Select PROM*, quindi *Next*

24. cliccare su *Finish in File Generation Summary*

25. cliccando su *OK* sulla dialog *Add Device* verrà data la possibilità di selezionare il file .bit contente il device in questione

26. una volta selezionato verrà richiesto se si desidera introdurre un altro device per il *Data Stream* (dipende dall'applicazione): noi scegliamo *No*!

27. nella tabella dei processi comparirà *Generate File...* (figura 18) sul quale un doppio click darà l'avvio alla generazione del PROM file .bin da caricare nell'FPGA indicando la catena di flusso PROM-FPGA[3]

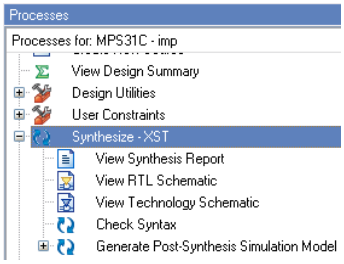

Figura 12: Sintesi del progetto e controllo sintassi

8 Il programma di arbitraggio in linguaggio C

Il programma per la gestione dei dati sul BUS è scritto in linguaggio C. Il prgramma viene avviato ponendo come parametro l'indirizzo del dispositivo da gestire, ovvero 220h. L'indirizzo viene memorizzato nella variabile *PortBase* necessaria

[2] Il file UCF può anche essere editato manualmente in modalità testuale, tramite il processo *Edit Constraints*.

[3] Il flusso di inserimento delle varie opzioni per la generazione del PROM file può essere evitato salvando il configuration project file all'atto della chiusura del tool iMPACT.

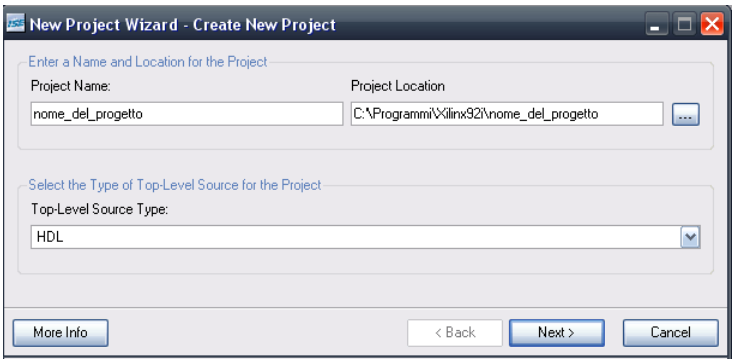

Figura 9: Finestra di impostazione progetto

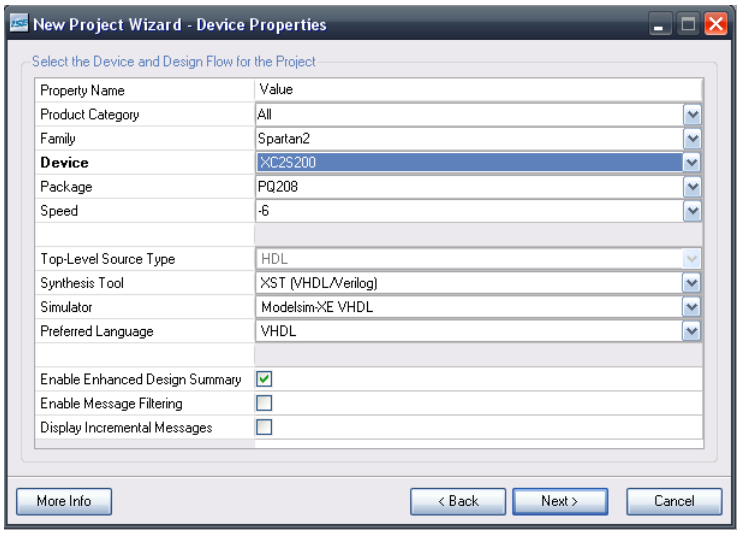

Figura 10: Selezione delle proprietà del dispositivo

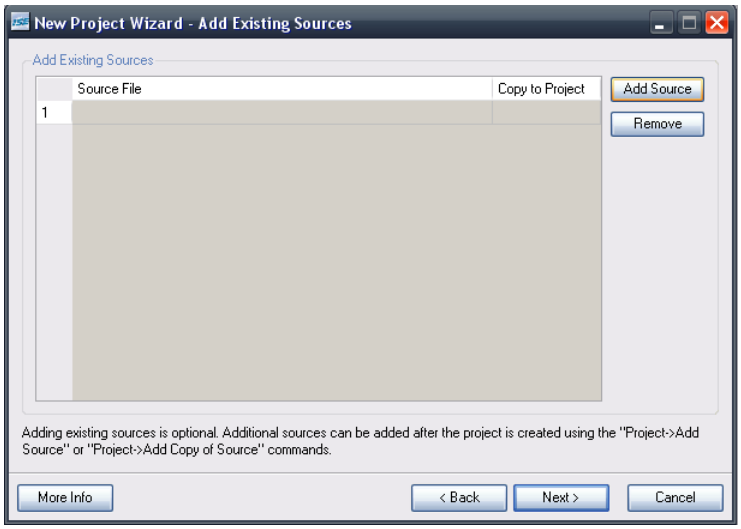

Figura 11: Finestra di selezione di sorgenti esterne

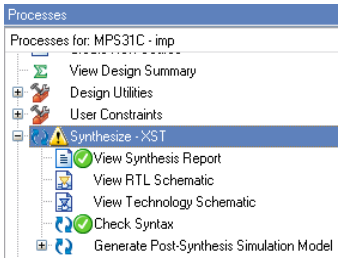

Figura 13: Sintesi terminata con successo

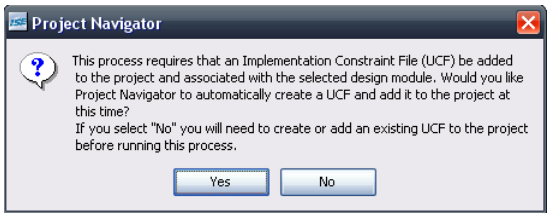

Figura 15: Messaggio di dialogo User Constraints

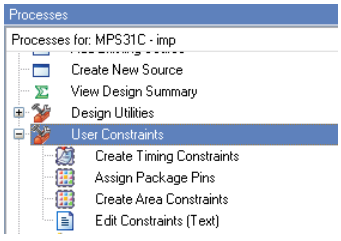

Figura 14: Finestra per il settaggio dei limiti temporali

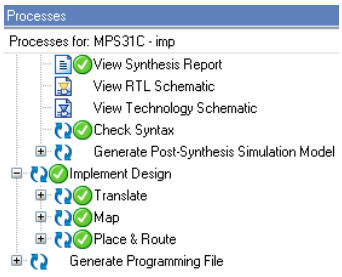

Figura 16: Schermata di fine implementazione

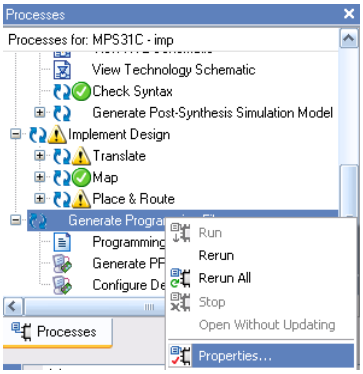

Figura 17: Schermata di generazione file di programmazione

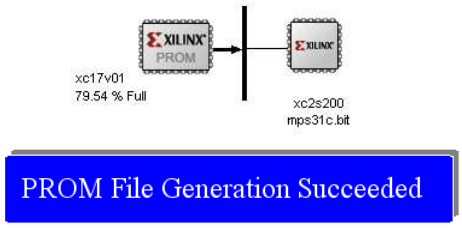

Figura 18: Schermata di fine generazione file binario

per la gestione del BUS. L'indirizzo base viene utilizzato per pilotare il registro di trasferimento dati mentre l'indirizzo base+1 viene utilizzato per la gestione del registro di controllo/status. Tutto questo avviene mediante la definizione di tre costanti:
♯ define R_4I34DATA 0
♯ define R_4I34CONTROL 1
♯ define R_4I34STATUS 1

Le istruzioni per la scrittura dei dati sono le seguenti:

1. outportb(PortBase+R_4I34DATA, byte)
2. outport (PortBase+R_4I34DATA, word)

dove con la prima istruzione è possibile scrivere un byte per volta quindi con una trasmissione a 8 bit, mentre con la seconda una parola da 16 bit. Per la lettura dei dati dal BUS valgono le stesse considerazioni per quanto riguarda la trasmissione a 16 oppure 8 bit e, sono le seguenti:

1. inportb(PortBase+R_4I34DATA, byte)
2. inport (PortBase+R_4I34DATA, word)

Per quanto riguarda il controllo del BUS vengono definite le seguenti costanti:
♯define B_4I34CFGCS 0
♯define M_4I34CFGCSON
(0x0 ≪ B_4I34CFGCS)
♯define M_4I34CFGCSOFF
(0x1 ≪ B_4I34CFGCS)

♯define B_4I34CFGINIT 1
♯define M_4I34CFGINITASSERT
(0x0 ≪ B_4I34CFGINIT)
♯define M_4I34CFGINITDEASSERT

(0x1 ≪ B_4I34CFGINIT)

♯define B_4I34CFGWRITE 2
♯define M_4I34CFGWRITEENABLE
(0x0 ≪ B_4I34CFGWRITE)
♯define M_4I34CFGWRITEDISABLE
(0x1 ≪ B_4I34CFGWRITE)

♯define B_4I34LED 3
♯define M_4I34LEDON
(0x1 ≪ B_4I34LED)
♯define M_4I34LEDOFF
(0x0 ≪ B_4I34LED)

Con riferimento alla prima definizione, la costante B_4I34CFGCS viene posta uguale a zero. L'istruzione successiva associa alla costante M_4I34CFGCSON il valore zero (0x0) shiftato a sinistra di un numero di posizioni pari a B_4I34CFGCS ovvero, nessuna posizione (B_4I34CFGCS=0). Analogamente, alla costante M_4I34CFGCSOFF viene associato il valore uno (0x1) shiftato a sinistra di zero posizioni (B_4I34CFGCS=0) e così via per tutte le altre definizioni. I valori delle costanti sono quindi:

M_4I34CFGCSON =%0000
M_4I34CFGCSOFF =%0001
M_4I34CFGINITASSERT =%0000
M_4I34CFGINITDEASSERT =%0010
M_4I34CFGWRITEENABLE =%0000
M_4I34CFGWRITEDISABLE =%0100
M_4I34LEDON =%0000
M_4I34LEDOFF =%1000

Il significato delle costanti è il seguente:

1. M_4I34CFGCSON: abilitazione accesso lettura/scrittura alla FPGA tramite il registro R_4I34DATA
2. M_4I34CFGCSOFF: disabilitazione accesso lettura/scrittura alla FPGA tramite il registro R_4I34DATA
3. M_4I34CFGINITASSERT: abilita modalità programmazione FPGA
4. M_4I34CFGINITDEASSERT: modalità attesa dati di programmazione
5. M_4I34CFGWRITEENABLE: direzione trasferimento dati CPU → R_4I34DATA
6. M_4I34CFGWRITEDISABLE: direzione trasferimento dati R_4I34DATA → CPU
7. M_4I34LEDON: LED rosso sulla scheda acceso
8. M_4I34LEDOFF: LED rosso sulla scheda spento

Un esempio di controllo è il seguente:

outportb ((PortBase + R_4I34CONTROL), (M_4I34CFGCSON | M_4I34CFGINITDEASSERT | M_4I34CFGWRITEENABLE | M_4I34LEDON));

dove il simbolo | rappresenta una operazione di OR e quindi il valore passato al registro di controllo è pari a %0010. Acquisite le basi di cui sopra, il codice sorgente presente on line risulta essere di facile intuizione per cui ne viene lasciata al lettore l'analisi, nonché un approfondimento e un'eventuale personalizzazione per l'applicazione specifica.

9 Un esempio pratico di implementazione

Nel seguito viene presentato un esempio di implementazione pratica, ovvero un un dispositivo che si basa sul conteggio degli impulsi per eseguire una misura di frequenza (o di tempo), sulla base del clock di sistema (base dei tempi) e che fornisce in uscita, sulla base del numero di impulsi (in formato binario) rilevati, una forma d'onda quadra. L'architettura realizzata è quella mostrata nella figura 19, i cui moduli presentano le seguenti funzionalità:

- cnt_trigg: contatore di impulsi del segnale a frequenza base (clock di sistema 50MHz) negli N periodi del segnale da rilevare);
- div_bin: divisore binario a 20 bit;
- wave_generator: generatore della forma d'onda d'uscita;
- pc104_manager: gestore della comunicazione di I/O con il bus PC104.

Il contatore di impulsi implementa quanto riportato nella figura 20, nella quale la porta individua una finestra temporale (gli N periodi del segnale da rilevare) durante la quale vengono conteggiati gli impulsi del segnale a frequenza base fc (base dei tempi). Particolare attenzione è stata posta nella realizzazione della finestra start-stop il cui inizio e fine sono stabiliti sulla base della transizione dal livello logico BASSO al livello logico ALTO del segnale da rilevare campionandone, alla frequenza di sistema, il livello assunto da tale segnale. Nelle figure che seguono vengono mostrati i segnali all'inizio del conteggio e alla fine (i valori sono stati indicati in formato unsigned per semplicità di rappresentazione - figg. 21 e 22).

L'algoritmo utilizzato per la divisione binaria - per dividere il numero di impulsi rilevati per il numero N di periodi - consiste nello shiftare il dividendo in un registro inizializzato a zero. Ogni qualvolta il dividendo eguaglia o supera il divisore quest'ultimo viene sottratto dal dividendo e viene inserito un 1 in una posizione appropriata del quoziente (in questo modo si rende la divisione binaria molto simile a quella aritmetica - figura 23).

Il waves_generator produce una forma d'onda quadra in uscita, il cui periodo T viene impostato dall'esterno (tramite bus PC-104) e il livello ALTO è pari alla somma di un tempo T1 (fornito sempre tramite bus) e di un tempo T2 (pari al numero di impulsi medio rilevato sugli N periodi - figg. 24, 25 e 26).

Il pc104_manager si occupa della gestione dell'I/O sul bus PC104 (controllando i segnali IORD, IOWR, IOCS16 e AEN - figura 27) per:

- introdurre il numero N di periodi per il conteggio degli impulsi;
- introdurre l'intervallo di tempo T1;
- introdurre il periodo T della forma d'onda d'uscita;
- acquisire e rendere disponibile all'esterno il numero totale di impulsi rilevato sugli N periodi.

10 Implementazione e misure

In questa sezione verrà riportata la lista dei comandi necessari per l'impostazione della forma d'onda d'uscita generata. La

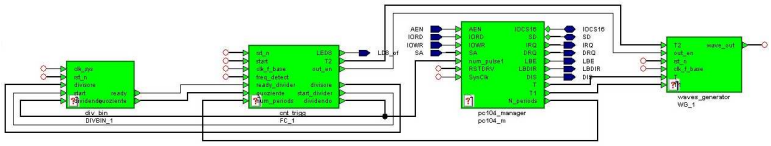

Figura 19: Schema a blocchi del sistema implementato

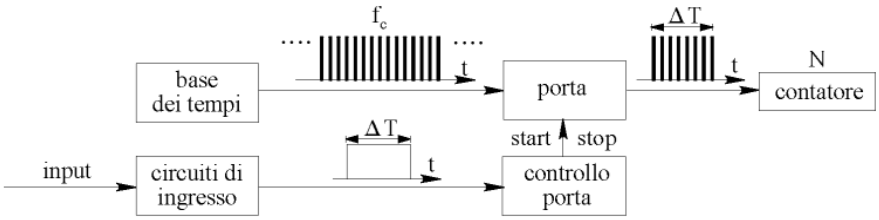

Figura 20: Schema di principio del contatore di impulsi

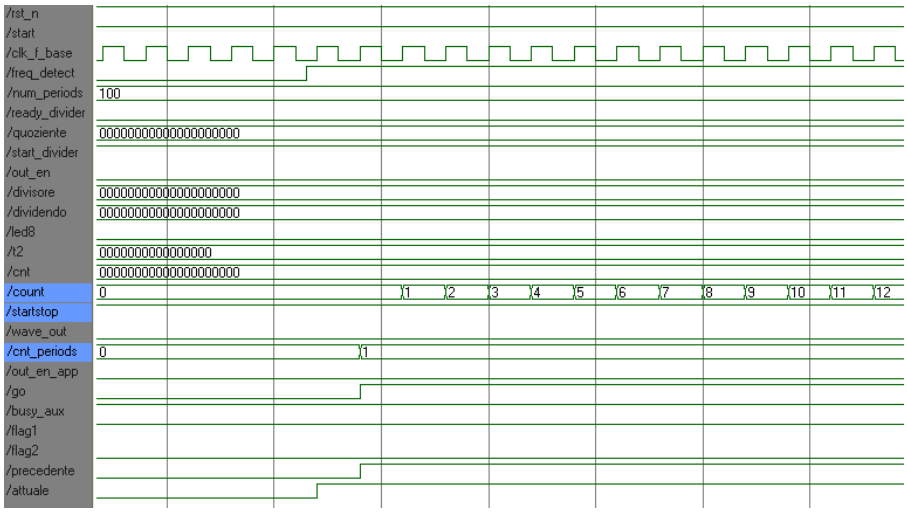

Figura 21: Simulazione temporale: inizio conteggio impulsi

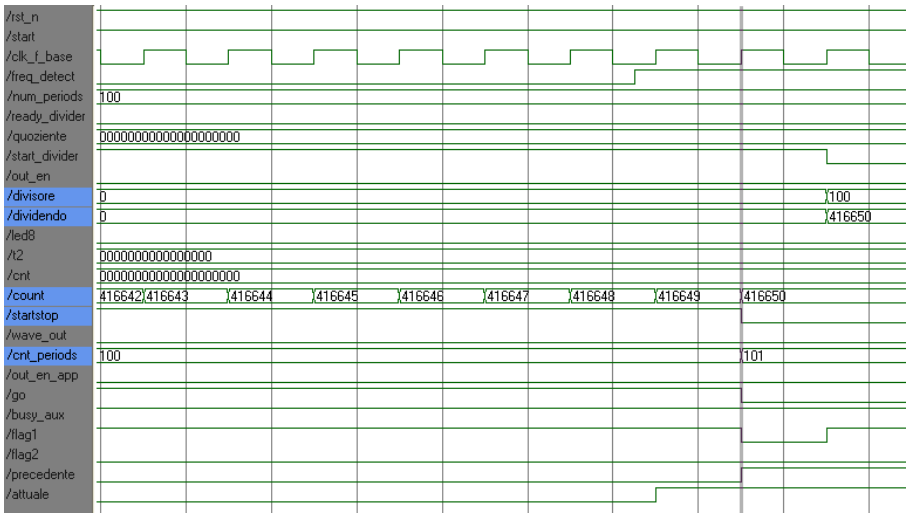

Figura 22: Simulazione temporale: fine conteggio impulsi

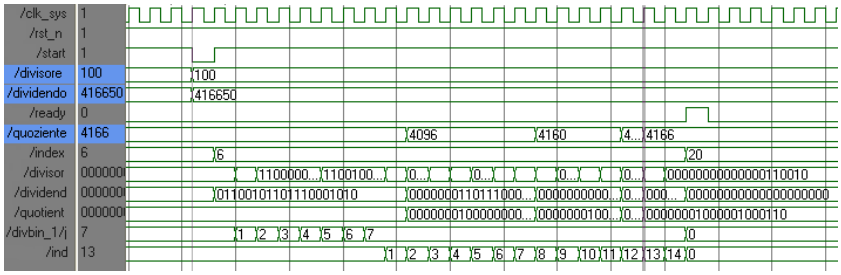

Figura 23: Simulazione temporale divisore

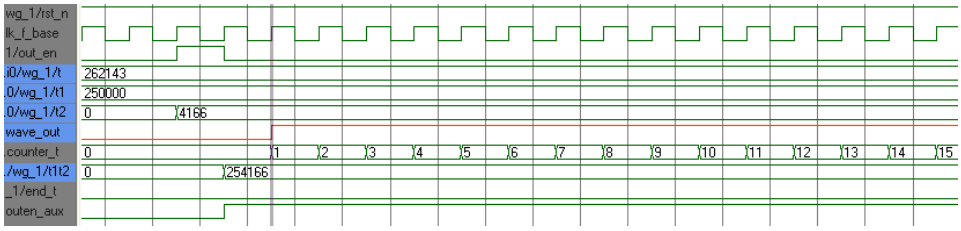

Figura 24: Simulazione temporale: inizio generazione forma d'onda di uscita

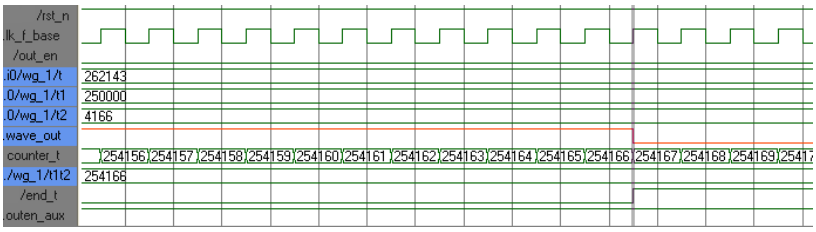

Figura 25: Simulazione temporale: fine livello logico alto - duty cycle

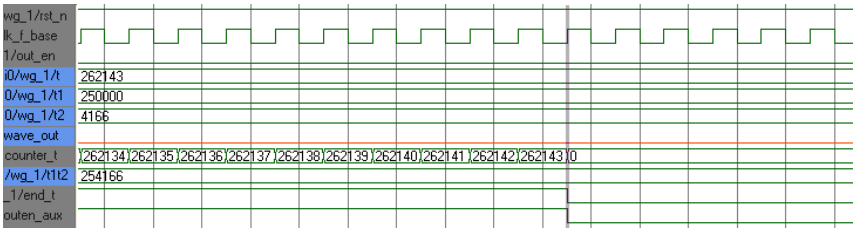

Figura 26: Simulazione temporale: fine periodo forma d'onda di uscita

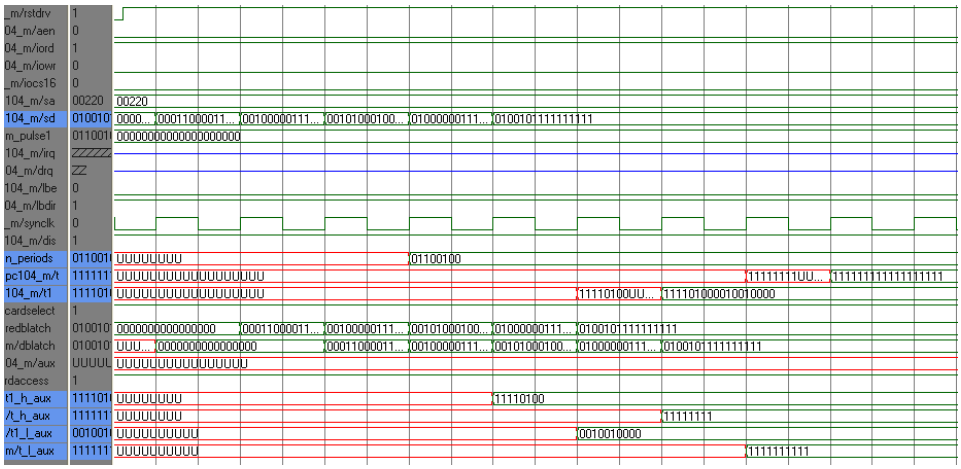

Figura 27: Simulazione temporale dell'acquisizione tramite il bus PC-104

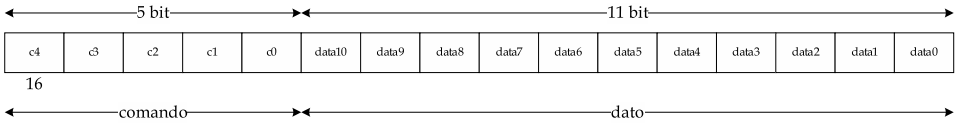

Figura 28: Struttura dei comandi/word a 16 bit

struttura di ogni singola word inviata sul BUS PC-104 è riportata in figura 28.

I primi cinque bit identificano il comando mentre i restanti 11 bit servono per la trasmissione dati. I comandi relativi al menu di scrittura dalla CPU alla FPGA, sono riportati nella tabella 5.

La tabella 6 invece mostra alcuni comandi di esempio utilizzati per fare le misure e i test del prototipo. Il valore di T1 è pari a 250000 che corrisponde ad un livello positivo dell'onda d'uscita pari a:

$$T1(t) = 250000 \cdot 20^{-9} = 5ms \qquad (1)$$

dove il fattore moltiplicativo di 20ns rappresenta il periodo della frequenza base, ovvero 50MHz.

Il primo test è stato fatto inviando solo il numero di periodi (100) e il periodo T dell'onda quadra. La figura 29 mostra la misura dell'onda quadra di uscita. Il segnale di ingresso è stato prelevato da un generatore di segnali ed ha una frequenza pari a 10KHz. Il numero di impulsi a frequenza base (50MHz) contenuti nel segnale di ingresso è pari:

$$np = \frac{50^6}{10^3} = 5000 \qquad (2)$$

Quindi in ogni periodo del segnale di ingresso sono contenuti 5000 impulsi a frequenza base. Poichè vengono conteggiati 100 periodi, il numero totale di impulsi è 500000. Tale valore viene posto in un registro che viene decrementato alla frequenza base per l'ampiezza dell'onda di uscita è pari a:

$$PWT2 = 5^5 \cdot 20^{-9} = 100\mu s \qquad (3)$$

La figura 30 mostra invece, la forma d'onda di uscita quando viene inviato

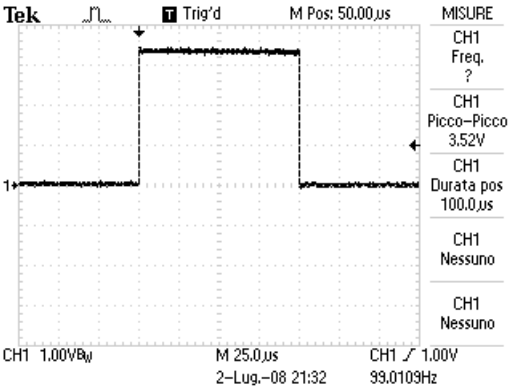

Figura 29: Forma d'onda d'uscita con n e T impostato

anche il valore di T1 che vale:

$$PWT1 = 25000 \cdot 20^{-9} = 5ms \qquad (4)$$

In questo caso l'ampiezza dell'impulso è:

$$PW = 5ms + 100\mu s = 5.1ms \qquad (5)$$

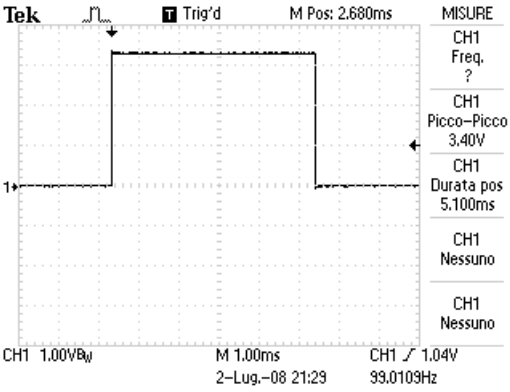

Figura 30: Forma d'onda d'uscita con n, T1 e T impostato

In definitiva, l'onda generata in uscita ha la struttura riportata in figura 31.

codice comando	significato	bit di dati
00011	numero di periodi	data7...data0
00100	parte alta T1	data7...data0
00101	parte bassa T1	data9...data0
00110	parte alta T	data7...data0
00111	parte bassa T	data9...data0

Tabella 5: Comandi relativi al menu di scrittura dalla CPU sull'FPGA

valore binario	valore esadecimale	significato
0001100001100100	1864	numero di periodi (100)
0010000011110100	20F4	parte alta T1 (250000)
0010100010010000	2890	parte bassa T1 (250000)
0100000011111111	40FF	parte alta T (262143)
0100101111111111	4BFF	parte bassa T (262143)

Tabella 6: Comandi di esempio per le misure e i test del prototipo

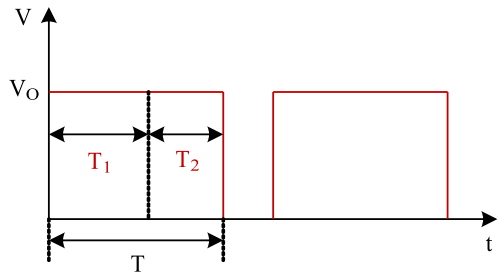

Figura 31: Struttura della forma d'onda di uscita

Riferimenti bibliografici

[1] Tom Shanley, Donanderson, *'ISA System Architecture Third Edition'*, Addison-Wesley Publishing Company.

[2] Marco Coppelli, Bruno Stortoni, *'Microprocessori Sistemi Teoria e Progetto, terza edizione'*, La Sovrana Editrice - Fermo.

[3] Xilinx, *'Spartan-II 2.5V FPGA Family: Complete Data Sheet'*.

[4] MESA, *'4C61A CPU Board datasheet'*, http://www.mesanet.com

[5] MESA, *'4i34M FPGA Board datasheet'*, http://www.mesanet.com

[6] Atmel, *'ATF20V8BQ datasheet'*.

[7] PC-104 Specification, *'http://www.pc104.org/specifications.php'*.

[8] Fairchild Semiconductor, *'74HCT245 datasheet'*.