



Università degli Studi di L'Aquila

D.I.I.I.E. Dipartimento di Ingegneria Industriale dell'Informazione e di Economia

D.I.S.I.M. Dipartimento di Ingegneria e Scienze dell'informazione e Matematica

Centro di Eccellenza DEWS

Tesina del corso:

EMBEDDED SYSTEMS

Docente:

Dott. Ric. Ing. L. Pomante

Studente: **Angelo Marinacci**

Matr. 207497

L.M. Ing. Elettronica

a.a. 2012/2013

Indice

Parte 1) Introduzione

Parte 2) Micro Controllore Intel 8051

Caratteristiche tecniche

Instruction-Set Architecture (I.S.A.)

Parte 3) Flussi di Simulazione

Software

Hardware

Parte 4) Programmabilità e Configurabilità

Parte 5) Conclusioni



Introduzione

Introduzione

Obiettivo del lavoro è:

La sperimentazione di un ISS (C++) e di un modello HDL sintetizzabile (VHDL) relativi al microcontrollore Intel 8051 (leggermente semplificato).

In dettaglio:

- i) Effettuare la simulazione del uC che esegue diversi programmi “noti” come se fossero caricati in ROM*
- ii) Simularne il funzionamento tramite un ISS e valutarne le prestazioni*
- iii) Simularne il funzionamento tramite i modelli HDL e valutarne le prestazioni*
- iv) Effettuare il confronto quantità/qualità delle informazioni e velocità in entrambi i tipi di simulazione.*

Micro Controllore Intel 8051

Micro Controllore Intel 8051

Caratteristiche tecniche

intel® 8051

HARVARD
ARCHITECTURE

Cpu ad 8-bit

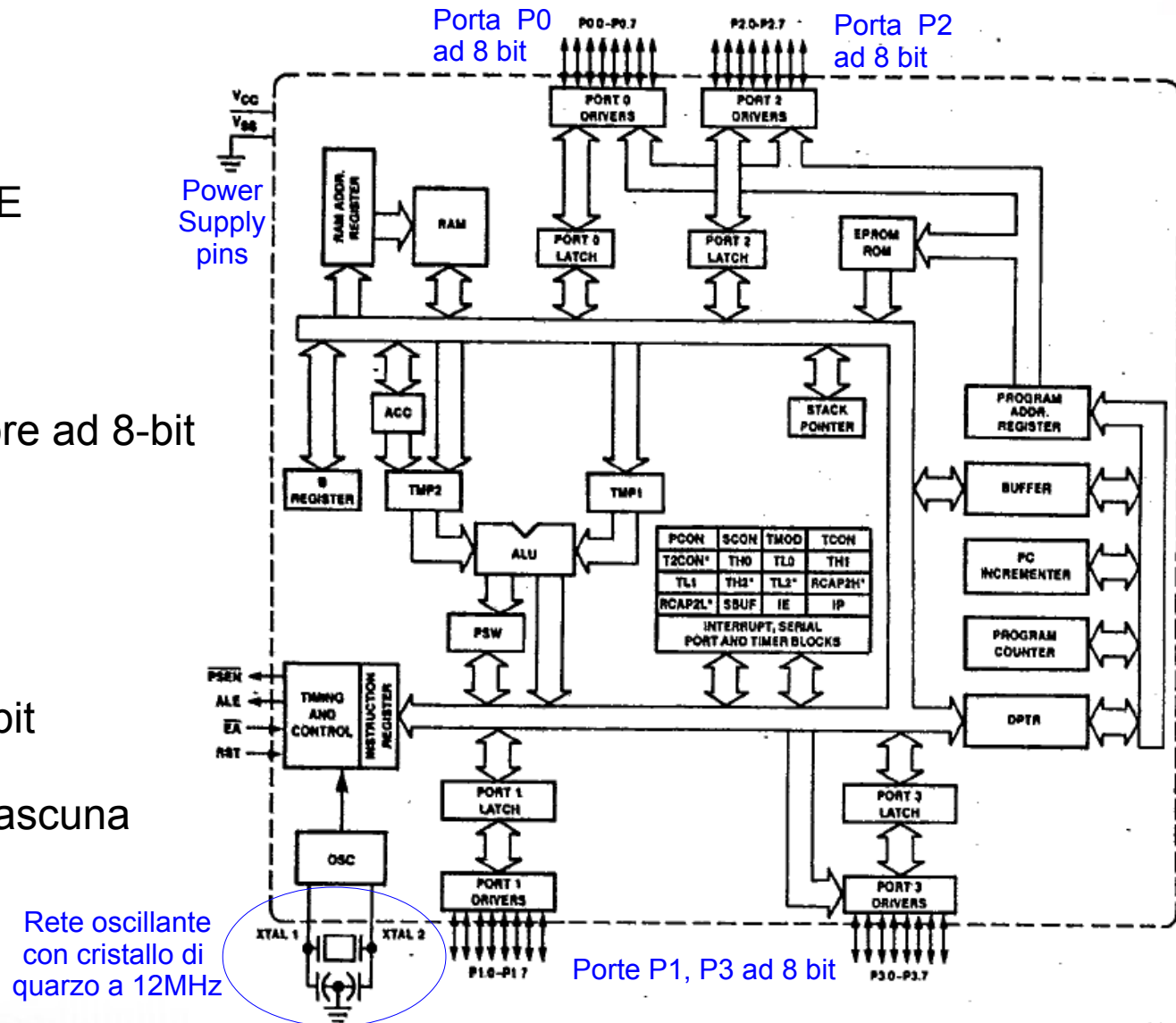
Registri e Accumulatore ad 8-bit

ALU ad 8-bit

DATA Bus ad 8-bit

ADDRESS Bus a 16-bit

4 Porte I/O da 8-bit ciascuna



Micro Controllore Intel 8051

Instruction Set Architecture (I.S.A.)

Di seguito sono riportati i codici mnemonici delle 111 istruzioni del Instruction Set Architecture del Intel 8051

ACALL,	ADD1,	ADD2,	ADD3,	ADD4,	ADDC1,	ADDC2,	ADDC3,	ADDC4,
AJMP,	ANL1,	ANL2,	ANL3,	ANL4,	ANL5,	ANL6,	ANL7,	ANL8,
CJNE1,	CJNE2,	CJNE3,	CJNE4,	CLR1,	CLR2,	CLR3,	CPL1,	CPL2,
CPL3,	DA,	DEC1,	DEC2,	DEC3,	DEC4,	DIV,	DJNZ1,	DJNZ2,
INC1,	INC2,	INC3,	INC4,	INC5,	JB,	JBC,	JC,	JMP,
JNB,	JNC,	JNZ,	JZ,	LCALL,	LJMP,	MOV1,	MOV2,	MOV3,
MOV4,	MOV5,	MOV6,	MOV7,	MOV8,	MOV9,	MOV10,	MOV11,	MOV12,
MOV13,	MOV14,	MOV15,	MOV16,	MOV17,	MOV18,	MOVC1,	MOVC2,	MOVX1,
MOVX2,	MOVX3,	MOVX4,	MUL,	NOP,	ORL1,	ORL2,	ORL3,	ORL4,
ORL5,	ORL6,	ORL7,	ORL8,	POP,	PUSH,	RET,	RETI,	RL,
RLC,	RR,	RRC,	SETB1,	SETB2,	SJMP,	SUBB1,	SUBB2,	SUBB3,
SUBB4,	SWAP,	XCH1,	XCH2,	XCH3,	XCHD,	XRL1,	XRL2,	XRL3,
XRL4,	XRL5,	XRL6,						

Si possono raggruppare nei seguenti gruppi:

Aritmetiche; di Controllo; Logiche; Operazioni sul singolo bit; di Trasferimento dati

Flusso di Simulazione Software

Flusso di Simulazione SW (1)

Si dispone (*) di:

Instruction-Set Simulator (I.S.S.) costituito dai seguenti file:

Main.cc
i8051.cc
i8051.h

Una serie di **files sorgente .c** di test, per ognuno dei quali esiste il corrispondente file **.hex** ottenuto, a partire dal sorgente, con compilatore Intel per 8051

Non volendo utilizzare una board con 8051, si procederà alla simulazione dell'esecuzione di un file **.hex** per 8051 mediante I.S.S. del uC

L'I.S.S. deve essere necessariamente compilato per l'ambiente in cui verrà lanciato, Windows in tal caso.

La **compilazione del I.S.S. in Windows** è effettuata mediante ambiente di sviluppo integrato, I.D.E., Microsoft Visual Studio 2010 Express (gratuito) creando un apposito progetto

(*) il materiale è reperibile gratuitamente in rete al sito <<http://www.cs.ucr.edu/~dalton/i8051/>>

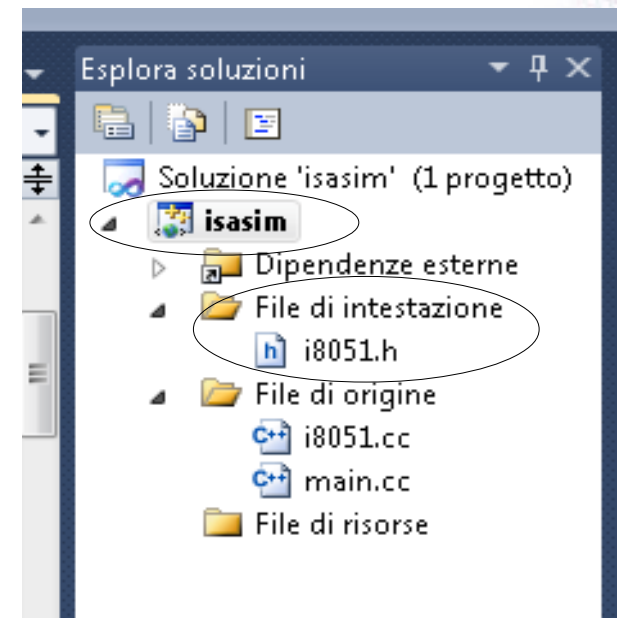
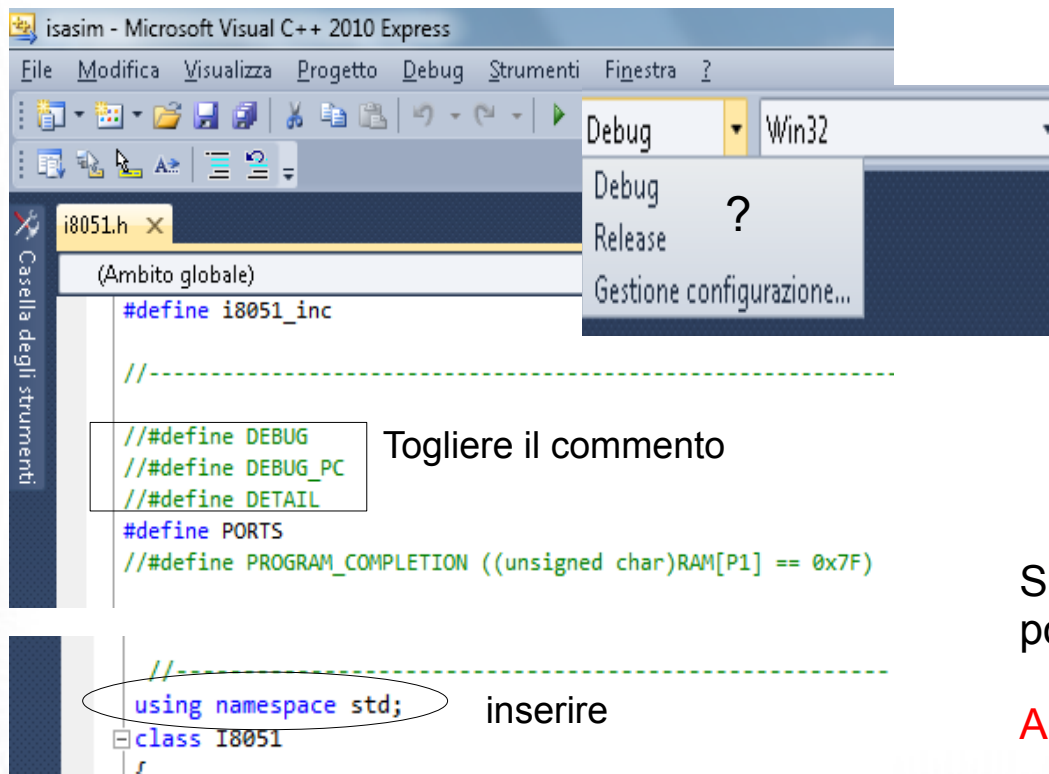
Flusso di Simulazione SW (2)

Avviato l'I.D.E. si crea un nuovo “Empty Project”, chiamato **isasim** in tal caso, importandovi i file precedentemente citati i quali verranno automaticamente riconosciuti dall'ambiente.

Nota: è preferibile creare nella cartella di lavoro dell'I.D.E. una sottocartella per ogni singolo progetto.

Successivamente si apre l'header file e:

- si rimuove il commento alle **#define...** il cui significato sarà descritto in seguito
- si inserisce il comando **using namespace std;** sopra la riga “**class I8051**”
(il prog. è stato scritto quando i namespace non esistevano)



Si effettua il salvataggio del file editato e poi dell'intero progetto.

A questo punto è possibile compilare

Flusso di Simulazione SW (3)

Gli esempi riportati nelle slides 12...22 fanno riferimento alla **compilazione** del progetto **isasim** effettuata in modalità **DEBUG**, come normalmente avviene durante la fase di sviluppo di un programma.

Tuttavia, il codice meglio se compilato in modalità **RELEASE**, poichè già “senza bug”:

- tale processo è maggiormente orientato all'esecuzione finale;
- l'applicazione generata, in quanto ottimizzata, richiederà al pc meno lavoro.

Nelle slides 23, 24 e 25 si fa riferimento alla compilazione dello stesso in modalità **RELEASE**, come avviene al suo rilascio, facendo il confronto con la modalità precedente **ma nei caso in cui le #define... sono commentate!**

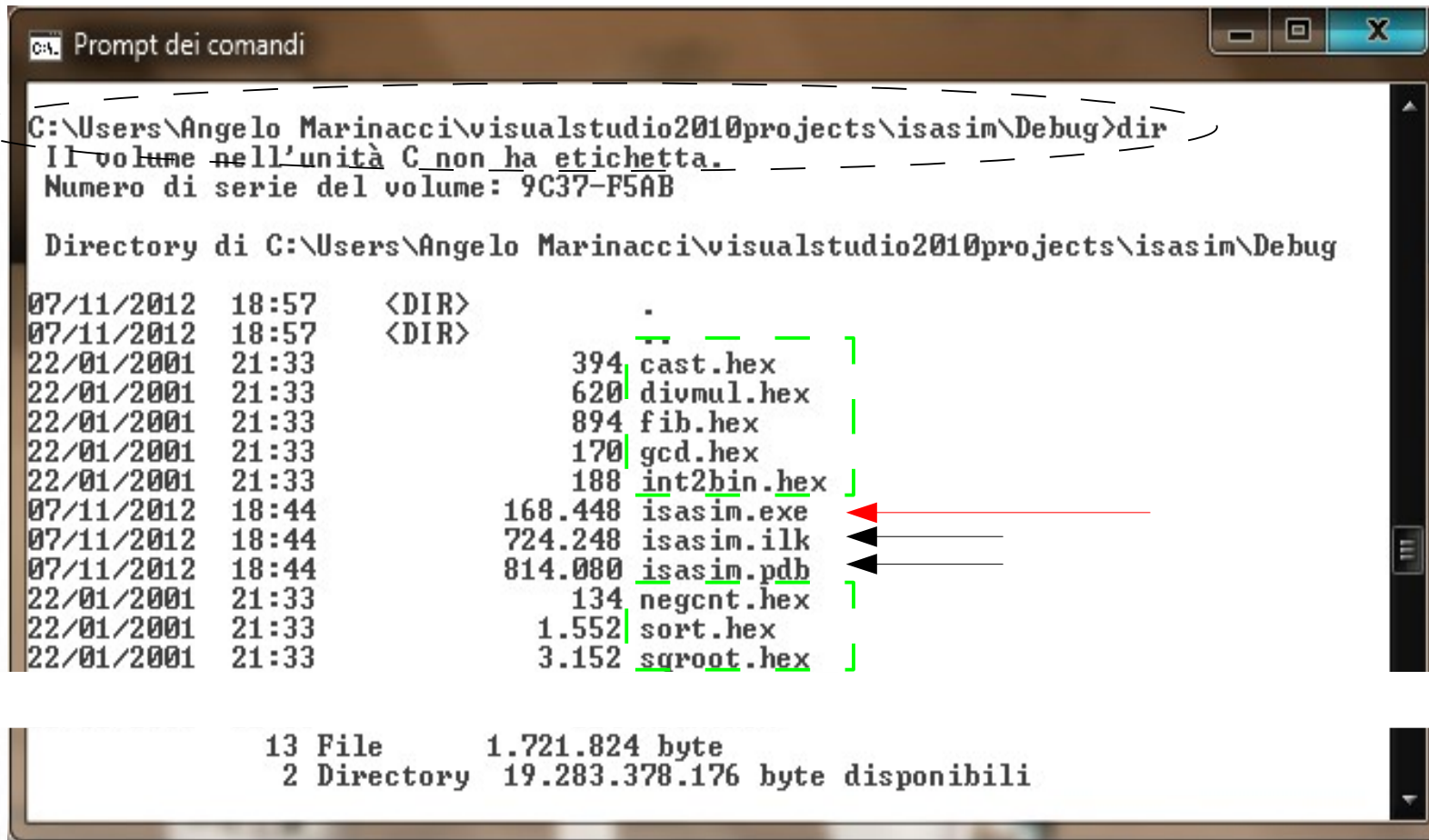
Infatti, tra le due modalità ma con le istruzioni #define... non commentate la differenza è minima poichè il pc è occupato a generare il rapporto con notevole quantità di informazioni.

Flusso di Simulazione SW (4)

Nella cartella di lavoro dell'I.D.E. è presente la cartella *isasim* che contiene l'intero progetto e del quale porta il nome. All'interno di quest'ultima c'è la sottocartella *Debug* contenente l'applicazione generata al termine della compilazione: *isasim.exe*.

Si copiano i file di test e li si incollano all'interno della cartella *Debug*.

Si avvia il *Prompt dei comandi* e si esplora il path fino a "raggiungerla"; si digita *dir* per accertarsi del contenuto: gli 8 *.hex* e la terna *.exe*, *.ilk*, *.pdb* generata dal compilatore.



```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>dir
Il volume nell'unità C non ha etichetta.
Numero di serie del volume: 9C37-F5AB

Directory di C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug
07/11/2012  18:57    <DIR>          .
07/11/2012  18:57    <DIR>          ..
22/01/2001  21:33          394 cast.hex
22/01/2001  21:33          620 diumul.hex
22/01/2001  21:33          894 fib.hex
22/01/2001  21:33          170 gcd.hex
22/01/2001  21:33          188 int2bin.hex
07/11/2012  18:44       168.448 isasim.exe
07/11/2012  18:44       724.248 isasim.ilk
07/11/2012  18:44       814.080 isasim.pdb
22/01/2001  21:33          134 negcnt.hex
22/01/2001  21:33       1.552 sort.hex
22/01/2001  21:33          3.152 sqroot.hex

13 File             1.721.824 byte
 2 Directory        19.283.378.176 byte disponibili
```


Flusso di Simulazione SW (5)

I due, degli otto totali, file sorgente i cui relativi .hex verranno simulati sono:

sqroot.c il uC calcola la radice quadrata della somma di due quadrati

divmul.c il uC effettua la divisione intera tra due numeri

sqroot.hex e **divmul.hex** differiscono per la percentuale di “copertura” dell'I.S.A. del 8051: utilizzano rispettivamente 54/111 e 26/111 tipi diversi di istruzioni dell'intero set

L'assembler per 8051 genera i file nello standard Intel HEX.

Il .hex è il file utilizzato per indicare come scrivere la ROM interna caricandovi il programma.

Di seguito è mostrato una parte di file nello standard Intel HEX:

```
: 10 0003 00 7F2F7E0BEF6E6015EFD39E4008C3EF9E EC
: 10 0013 00 FFF58080EFC3EE9FFEF59080E78FA080 11
: 02 0023 00 FE22 BB
: 03 0000 00 020025 D6
: 0C 0025 00 787FE4F6D8FD758107020003 27
: 00 0000 01 FF
```

Posizione 1	simbolo : individua l'inizio del record
Posizioni 2-3	lunghezza del record
Posizioni 4to7	indirizzo dei dati da 0000h a FFFFh (tutto lo spazio indirizzabile)
Posizioni 8-9	tipo di dato
Posizioni 10to?	campo dati, ogni dato da 1 byte è rappresentato da due cifre esadecimali
Ultime due	CHECKSUM

Flusso di Simulazione SW (6)

sqroot.c

```
#include <reg51.h>
#include <math.h>

void main() {

    float x = 3.0;
    float y = 4.0;
    float xx, yy, xx_yy, sqrt_xx_yy;

    xx = x * x;
    P0 = (unsigned char)xx;

    yy = y * y;
    P1 = (unsigned char)yy;

    xx_yy = xx + yy;
    P2 = (unsigned char)xx_yy;

    sqrt_xx_yy = sqrt(xx_yy);
    P3 = (unsigned char)sqrt_xx_yy;

    while(1);
}
```

*Header file necessario nel
reale ambiente di sviluppo
per 8051 usato per la
compilazione*

divmul.c

```
#include <reg51.h>

void main() {

    unsigned x = 134;
    unsigned y = 1;
    unsigned q, r, p, i;

    for(i=0; i<12; i++) {

        y++;

        q = x / y;
        r = x % y;
        p = q * y + r;

        P0 = q;
        P0 = r;
        P0 = p;

        while(1);
    }
}
```

*Il programma sarà terminato tramite ridefinizione
di ctrl+c. Esiste un modo più pulito ma fare
modifiche implica avere un compilatore per 8051*

Flusso di Simulazione SW (7)

Dal *Prompt dei comandi*, all'interno della cartella **Debug** o della cartella **Release** (...\\...\\Debug>_) o (...\\...\\Release>_):

- la simulazione può essere avviata mediante il seguente comando:

```
nome-applicazione    <hex-file>    <output-file>
```

*Nota: l'output-file è un **file di testo** che fornisce informazioni in merito all'esecuzione appena terminata. Non è pre-esistente ed il nome assegnatogli può essere arbitrario.*

Il rapporto è generato, al termine, solo previa abilitazione di alcune #define... dalle quali dipende la quantità di informazioni visualizzabili nello stesso. In caso contrario non verrà creato alcun .txt

- i due programmi di test si avviano rispettivamente nel seguente modo:

```
isasim    sqroot.hex    sqrootreport.txt  
isasim    divmul.hex    divmulreport.txt
```

- lanciata la simulazione per **interromperne l'esecuzione** l'utente dovrà digitare da tastiera **ctrl+c**, essendo la condizione while(1) nel sorgente sempre verificata.

- in seguito al ctrl+c, digitando il nome del file testuale seguito dall'estensione, seguito da invio, si apre il “blocco note” e lo si visualizza.

ISASIM - Debug mode (1)

simulazione sqroot.hex

Durante la simulazione è visualizzato lo stato delle porte durante l'avanzamento del programma

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim sqroot.hex  
x sqrootreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x09	0xFF	0xFF	0xFF
0x09	0x10	0xFF	0xFF
0x09	0x10	0x19	0xFF
0x09	0x10	0x19	0x05

ctrl+c

```
-----  
Instructions Executed:          77114  
Execution Time(seconds):      2.344  
Average Instructions/second:  32898.5  
  
Clock Cycles Required for 8051: 1811772  
Execution Time for 8051(12 MHz)(seconds): 0.150981  
Average Instructions/second for 8051: 510753
```

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>_
```

I dati sono forniti in uscita mediante le 4 porte I/O a 8 bit:
i risultati parziali in P0, P1 e P2;
il risultato finale in P3.

A video sono rappresentati in esadecimale previa conversione in carattere del rispettivo float a 32bit

09h = 0000 1001bcd = 9d = 3^2
10h = 0001 0000bcd = 16d = 4^2
19h = 0001 1001bcd = 25d = $3^2 + 4^2$
05h = 0000 0101bcd = 5d = $\sqrt{(3^2 + 4^2)}$

Digitando **ctrl+c** termina l'esecuzione e vengono fornite info statistiche riguardanti le prestazioni temporali

ISASIM - Debug mode (2)

simulazione sqroot.hex

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim sqroot.hex  
x sqrootreport.txt
```

INVIO

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x09	0xFF	0xFF	0xFF
0x09	0x10	0xFF	0xFF
0x09	0x10	0x19	0xFF
0x09	0x10	0x19	0x05

CTRL+C

```
-----  
Instructions Executed:                77114  
Execution Time(seconds):             2.344  
Average Instructions/second:         32898.5  
-----
```

```
-----  
Clock Cycles Required for 8051:      1811772  
Execution Time for 8051(12 MHz)(seconds): 0.150981  
Average Instructions/second for 8051: 510753  
-----
```

```
#define DEBUG  
#define DEBUG_PC  
#define DETAIL  
#define PORTS  
//#define PROGRAM_
```

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>
```

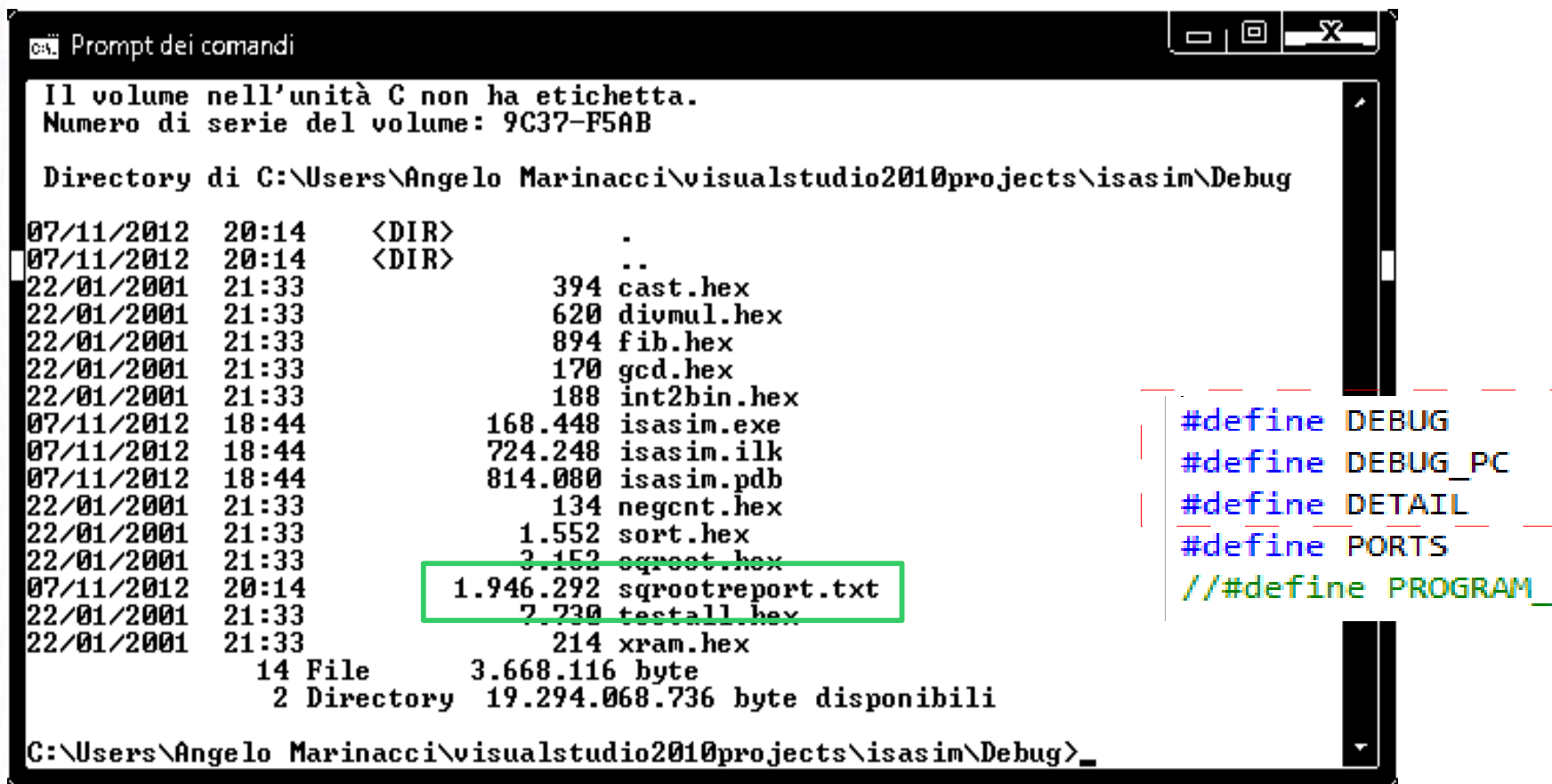
La simulazione è in run per $\approx 2.35s$; il tempo è calcolato tra le digitazioni di invio e ctrl+c. L'I.S.S. simula ben 77114 istruzioni con una media di 32898 istruzioni/s.

Il microcontrollore clockato a 12MHz sarebbe in run per $1.811772 \cdot 10^6$ cicli di clock ovvero per appena 0.15s reali circa.

Il uC così cadenzato ha la capacità di eseguire mediamente 510753 istruzioni al secondo ed infatti $510753 \cdot 0.15 = 77113.99$ pari al n. di istruzioni totali eseguite.

ISASIM - Debug mode (3)

simulazione sqroot.hex



```
C:\> Prompt dei comandi

Il volume nell'unità C non ha etichetta.
Numero di serie del volume: 9C37-F5AB

Directory di C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug
07/11/2012  20:14    <DIR>          .
07/11/2012  20:14    <DIR>          ..
22/01/2001  21:33             394 cast.hex
22/01/2001  21:33             620 divmul.hex
22/01/2001  21:33             894 fib.hex
22/01/2001  21:33             170 gcd.hex
22/01/2001  21:33             188 int2bin.hex
07/11/2012  18:44        168.448 isasim.exe
07/11/2012  18:44        724.248 isasim.ilc
07/11/2012  18:44        814.080 isasim.pdb
22/01/2001  21:33             134 negcnt.hex
22/01/2001  21:33             1.552 sort.hex
22/01/2001  21:33             3.152 sqroot.hex
07/11/2012  20:14    1.946.292 sqrootreport.txt
22/01/2001  21:33             7.730 testall.hex
22/01/2001  21:33             214 xram.hex
          14 File           3.668.116 byte
          2 Directory      19.294.068.736 byte disponibili

C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>
```

```
#define DEBUG
#define DEBUG_PC
#define DETAIL
#define PORTS
// #define PROGRAM
```

Visualizzando il contenuto della directory *Debug* si nota la presenza del file testuale di uscita *sqrootreport.txt* generato a fine simulazione ovviamente solo per lo specifico programma eseguito.

E' illustrato di seguito

ISASIM - Debug mode (4)

simulazione sqroot.hex

```
sqrootreport - Blocco note
File  Modifica  Formato  Visualizza  ?
00803 - ADD 4
        A <- A + 0x81
        A = 0x7F
        B = 0x81
        A + B = 0x00
1 1 1|
00805 - XCH 1
        A <-> R0
00806 - JNC
        if( !C ) JMP 815
00808 - CLR 2
        C <- 0
00809 - SUBB 1
        A <- A - R0 - C
        A = 0x7F
        B = 0x00
        C = 0x00
00810 - JNC
        if( !C ) JMP 818
00818 - MOV 8
        RAM(130) <- A
00820 - MOV 1
        A <- R1
00821 - ADD 1
        A <- A + R1
        A = 0xA0
        B = 0xA0
        A + B = 0x40
0 0 1
00822 - ORL 1
        A <- A | R3
00823 - ORL 1
        A <- A | R2
00824 - JNZ
        if( A != 0 ) JMP 831
00831 - MOV 12
```

File di testo ottenuto a fine simulazione

```
#define DEBUG
#define DEBUG_PC
#define DETAIL
#define PORTS
// #define PROGRAM_COMPLETION
```

Sezione di codice sorgente relativa alle info da visualizzare nel rapporto
Il livello di dettaglio è il massimo ottenibile poichè il simbolo di commento è rimosso a monte di tutte e tre le define.

La 5a, `#define PROGRAM_COMPLETION`, è ignorata poichè il programma è terminato dall'utente.

ISASIM - Debug mode (5)

simulazione sqroot.hex

```
sqrootreport - Blocco note
File Modifica Formato Visualizza ?
MOV 5      R6 <- A
MOV 2      A <- RAM(240)
LJMP      LJMP611
MOV 5      R5 <- A
RLC       RLC A
POP       POP RAM(224)
MOV 5      R3 <- A
JNC       if( !C ) JMP 625
INC 2      R3++
CJNE 3     if( R3 != 0x00 ) JMP 637
           if( R3 < 0x00 ) C <- 1
           else C <- 0
LJMP      LJMP718
MOV 1      A <- R4
JNB       if( !RAM(352).7 ) JMP 738
MOV 16     C <- RAM(336).5
MOV 1      A <- R3
RRC       RRC A
MOV 5      R4 <- A
```

```
#define DEBUG
//#define DEBUG_PC
#define DETAIL
#define PORTS
//#define PROGRAM_COM
```

Il livello di dettaglio è minore in questi altri due casi.

Alcune righe di codice sono commentate rispetto al caso precedente.

Di conseguenza alcune sezioni di codice de I sorgente i8051.cc verranno ignorate

```
sqrootr
File Mod
MOV 1
RLC
MOV 5
MOV 17
MOV 1
SETB 2
XCH 1
RLC
MOV 1
RLC
MOV 5
JNC
RET
MOV 10
ANL 1
INC 1
JNZ
MOV 1
INC 1
JNZ
MOV 17
MOV 1
INC 1
JZ
CLR 1
XCH 1
PUSH
CLR 2
SUBB 1
MOV 5
JZ
JNB
MOV 1
ADD 1
MOV 5
MOV 1
```

```
#define DEBUG
//#define DEBUG_PC
//#define DETAIL
#define PORTS
//#define PROGRAM_COM
```


ISASIM - Debug mode (6)

simulazione sqroot.hex

Nel caso di un diverso livello di dettaglio del rapporto:

- la simulazione è in run per $\approx 1.37s$ (un secondo in meno rispetto al caso precedente) ma questa volta sono state eseguite ben ≈ 14.84 milioni di istruzioni.
- in poco meno di un secondo e mezzo sono stati simulati quasi 30s di esecuzione all'interno del 8051 che equivalgono a ben 356.117556 M-cicli di clock a 12MHz

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim sqroot.hex  
x sqrootreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x09	0xFF	0xFF	0xFF
0x09	0x10	0xFF	0xFF
0x09	0x10	0x19	0xFF
0x09	0x10	0x19	0x05

```
#define DEBUG  
// #define DEBUG_PC  
// #define DETAIL  
#define PORTS  
// #define PROGRAM_COM
```

```
Instructions Executed:  
Execution Time(seconds):  
Average Instructions/second:
```

```
14839855  
1.375  
1.07926e+007
```

```
Clock Cycles Required for 8051:  
Execution Time for 8051(12 MHz)(seconds):  
Average Instructions/second for 8051:
```

```
356117556  
29.6765  
500055
```

- in meno tempo è stata simulata una esecuzione 200 volte più duratura dei soli 0.15s del caso precedente.

- è da notare però che la simulazione dal punto di vista computazionale è meno corposa

- il tutto dipende anche dal carico del PC che si utilizza per eseguire l'I.S.S.!

Maggiore è il carico computazionale per il calcolatore (livello di dettaglio del .txt) e minore sarà, a parità della durata della simulazione, il tempo di esecuzione del uC simulato.

ISASIM - Debug mode (7)

simulazione divmul.hex

La procedura descritta è ora applicata al file divmul.hex ed il rapporto è chiamato divmulreport.txt

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim divmul.hex  
x divmulreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x0A	0xFF	0xFF	0xFF
0x04	0xFF	0xFF	0xFF
0x86	0xFF	0xFF	0xFF

P0 evolve durante l'esecuzione

P1, P2, P3 Inattive (a 0xFF)

```
Instructions Executed:  
Execution Time(seconds):  
Average Instructions/second:
```

52829
2.328
22692.9

```
Clock Cycles Required for 8051:  
Execution Time for 8051(12 MHz)(seconds):  
Average Instructions/second for 8051:
```

1265448
0.105454
500967

Evolve soltanto lo stato della porta P0; le altre sono in idle

La porta mostra in sequenza il valore esadecimale del quoziente, del resto e del dividendo:

0Ah = 0000 1010 bcd = 10d
04h = 0000 0100 bcd = 4d
86h = 1000 0110 bcd = 134d

Le 3 #define... sono tutte abilitate

La simulazione è in run per $\approx 2.33s$; il tempo è calcolato sempre tra le digitazioni di invio e ctrl+c. All'I.S.S. sono richieste ben 52829 istruzioni con una media di 22692 istruzioni/s.

Il microcontrollore clockato a 12MHz sarebbe in run per $1.265448 \cdot 10^6$ cicli di clock ovvero per appena $\approx 0.10s$ reali circa.

Il uC così cadenzato ha la capacità di eseguire mediamente 510753 istruzioni al secondo ed infatti $500967 \cdot 0.10 = 52828.97$ pari al n. di istruzioni totali eseguite.

ISASIM - Release mode (vs. Debug) (1)

simulazione sqroot.hex

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Release>isasim sqroot.  
hex sqrootreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x09	0xFF	0xFF	0xFF
0x09	0x10	0xFF	0xFF
0x09	0x10	0x19	0xFF
0x09	0x10	0x19	0x05

```
Instructions Executed:  
Execution Time(seconds):  
Average Instructions/second:
```

```
Clock Cycles Required for 8051:  
Execution Time for 8051(12 MHz)(seconds):  
Average Instructions/second for 8051:
```

> 250 M-istruzioni

251473435
2.531
9.93573e+007

1740356180
145.03
1.73394e+006

```
//#define DEBUG  
//#define DEBUG_PC  
//#define DETAIL  
#define PORTS  
//#define PROGRAM_COM
```

Notare come nello stesso tempo di simulazione, $\approx 2.5s$, sono stati simulati ben 145s di funzionamento reale del uC compilando in mod. RELEASE rispetto ai "soliti" $\approx 56s$ compilando in mod. DEBUG!

In fase di rilascio il programma è ottimizzato dunque maggiormente orientato all'esecuzione

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim sqroot.  
hex sqrootreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x09	0xFF	0xFF	0xFF
0x09	0x10	0xFF	0xFF
0x09	0x10	0x19	0xFF
0x09	0x10	0x19	0x05

```
Instructions Executed:  
Execution Time(seconds):  
Average Instructions/second:
```

```
Clock Cycles Required for 8051:  
Execution Time for 8051(12 MHz)(seconds):  
Average Instructions/second for 8051:
```

≈ 28 M-istruzioni

27953447
2.563
1.09065e+007

670843764
55.9036
500029

```
//#define DEBUG  
//#define DEBUG_PC  
//#define DETAIL  
#define PORTS  
//#define PROGRAM_COM
```

La durata della simulazione è simile ai casi precedenti, $\approx 2.5s$ anziché $\approx 2.3s$, in cui almeno una delle 3 #define... è abilitata: si noti a parità di durata, la sostanziale differenza nelle istruzioni eseguite in ambedue le modalità di compilazione.

Ciò ad ulteriore testimonianza del fatto che molto dipende dal quanto il pc è occupato.

ISASIM - Release mode (vs. Debug) (2)

simulazione divmul.hex

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Release>isasim divmul.
```

```
hex divmulreport.hex
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x0A	0xFF	0xFF	0xFF
0x04	0xFF	0xFF	0xFF
0x86	0xFF	0xFF	0xFF

Instructions Executed:	281272167
Execution Time(seconds):	2.781
Average Instructions/second:	1.01141e+008

Clock Cycles Required for 8051:	2455562264
Execution Time for 8051(12 MHz)(seconds):	204.63
Average Instructions/second for 8051:	1.37454e+006

```

#define DEBUG
#define DEBUG_PC
#define DETAIL
#define PORTS
#define PROGRAM_COM

```

Dinuoovo quasi nello stesso tempo di simulazione, <3s, sono stati simulati ben 205s di funzionamento reale del uC compilando in mod. RELEASE rispetto ai "soli" ≈51s compilando in mod. DEBUG!

In fase di rilascio il programma è ottimizzato dunque maggiormente orientato all'esecuzione

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>
```

```
C:\Users\Angelo Marinacci\visualstudio2010projects\isasim\Debug>isasim divmul.he
```

```
x divmulreport.txt
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x0A	0xFF	0xFF	0xFF
0x04	0xFF	0xFF	0xFF
0x86	0xFF	0xFF	0xFF

Instructions Executed:	25977026
Execution Time(seconds):	2.36
Average Instructions/second:	1.10072e+007

Clock Cycles Required for 8051:	623446176
Execution Time for 8051(12 MHz)(seconds):	51.9538
Average Instructions/second for 8051:	500002

```

#define DEBUG
#define DEBUG_PC
#define DETAIL
#define PORTS
#define PROGRAM_COM

```

La durata della simulazione è simile ai casi precedenti, sempre minore di 3s, in cui almeno una delle 3 #define... è abilitata: si noti a parità di durata, la sostanziale differenza nelle istruzioni eseguite in ambedue le modalità di compilazione.

Ciò ad ulteriore testimonianza del fatto che molto dipende dal quanto il pc è occupato.

ISASIM - Release mode (vs. Debug) (3)

simulazione divmul.hex

```
C:\Users\Angelo Marinacci\visualstudio2010\projects\isasim\Release>isasim divmul.  
hex divmulreport.hex
```

P0	P1	P2	P3
0xFF	0xFF	0xFF	0xFF
0x0A	0xFF	0xFF	0xFF
0x04	0xFF	0xFF	0xFF
0x86	0xFF	0xFF	0xFF

Instructions Executed:	373077260
Execution Time(seconds):	3.703
Average Instructions/second:	1.0075e+008

Clock Cycles Required for 8051:	363917200
Execution Time for 8051(12 MHz)(seconds):	30.3264
Average Instructions/second for 8051:	1.2302e+007

Addirittura in quest'altra situazione, in più tempo, quasi 4s di durata della simulazione, sono stati simulati appena 30s di funzionamento del 8051.

Pur essendo in modalità RELEASE e con le #define... commentate.

```
//#define DEBUG  
//#define DEBUG_PC  
//#define DETAIL  
#define PORTS  
//#define PROGRAM_COM
```

E' da precisare che tutte le simulazioni sono state eseguite utilizzando il S.O. Windows 7 ma istallato su macchina virtuale.

Il software di virtualizzazione è configurato in modo da ottimizzare le prestazioni della macchina HOST, avente Mac OS X 10.6.8, e non la performance del S.O. GUEST sul quale si stanno effettuando i test con l'I.S.S.

Dunque la capacità computazionale in ambiente Windows è influenzata anche da questo fattore!

Flusso di Simulazione Hardware

Flusso di Simulazione HW (1)

Si simulerà il uC in esecuzione degli stessi programmi di test utilizzando stavolta un modello HDL (e uno strumento EDA per FPGA) procedendo successivamente al confronto con i risultati ottenuti nel corso delle simulazioni mediante I.S.S.

Ric.: La FPGA (Field Programmable Gate Array) è un dispositivo logico ri-configurabile: ciò che vi si scarica andrà ad implementare un circuito digitale ovvero dell'hardware. Non è presente nulla all'interno del chip quando non è configurato: soltanto la “matrice” di porte logiche le quali in attesa di essere combinate in una qualche maniera.

La sintesi, qualora attuata, ricreerà l'8051 in esecuzione di sqroot.hex oppure di divmul.hex così come verrebbe implementato all'interno del chip. La differenza tra i due casi è nella ROM, scritta in maniera diversa. Tuttavia in questa trattazione ci si limiterà alla sola simulazione.

Il sintetizzatore opera in maniera diversa a seconda della casa costruttrice del chip, in questo caso la nipponica **Xilinx**

(il modo in cui viene realizzato lo stesso circuito differisce tra i costruttori: Aldec, Altera, Synopsys, Xilinx,...)

L'ambiente software sviluppato dalla stessa si chiama ISE (quì nella v. 12.2) ed il simulatore in dotazione è ISIM.

Flusso di Simulazione HW (2)

Di seguito sono indicati i file sorgente, scritti in VHDL (*Very high speed integrated circuits Hardware Description Language*), ciascuno dei quali:

- **descrivere** una particolare area del uC8051
- consentirebbe di **configurare** il gate array con la medesima area

i8051_all.vhd	i8051_alu.vhd	
	i8051_dec.vhd	i8051_tsb.vhd
	i8051_ctr.vhd	i8051_lib.vhd
	i8051_ram.vhd	i8051_dbg.vhd
	i8051_rom.vhd*	
	i8051_xrm.vhd	

(*) nota:

Un discorso a parte deve essere fatto per il modulo di memoria ROM.
Essa differisce, per contenuto, a seconda se vi si è caricato sqroot.hex oppure divmul.hex.

A tal proposito si dispone del file sorgente **i8051_mkr.c**

In MS Visual Studio si crea un nuovo empty project chiamato, non a caso, **hex2rom** importandovi il su indicato file sorgente; al termine della compilazione dello stesso è generata l'applicazione **hex2rom.exe** (all'interno della corrispondente cartella *Release*)

Flusso di Simulazione HW (3)

Dal Prompt dei comandi si entra nella cartella hex2rom, "sede" del progetto:

```
C:\Users\Angelo Marinacci\visualstudio2010projects\hex2rom>dir
Il volume nell'unità C non ha etichetta.
Numero di serie del volume: 9C37-F5AB
```

```
Directory di C:\Users\Angelo Marinacci\visualstudio2010projects\hex2rom

12/11/2012  22:03    <DIR>
12/11/2012  22:03    <DIR>
12/11/2012  22:00    <DIR>
10/11/2012  13:59    <DIR>
12/11/2012  22:03      1.789.952 hex2rom.sdf
10/11/2012  13:12      893 hex2rom.sln
10/11/2012  13:59    <DIR>
                ipch
                2 File      1.790.845 byte
                5 Directory 18.145.460.224 byte disponibili

C:\Users\Angelo Marinacci\visualstudio2010projects\hex2rom>
```

Release

hex2rom contiene la sotto cartella
Release

Successivamente si copiano i files
.hex e li si incollano all'interno di
quest'ultima che conterrà:

```
12/11/2012  22:00    <DIR>
12/11/2012  22:00    <DIR>
22/01/2001  21:33      620 divmul.hex
10/11/2012  15:11      37.376 hex2rom.exe
10/11/2012  15:11     366.996 hex2rom.ilc
10/11/2012  15:11     379.904 hex2rom.pdb
22/01/2001  21:33      3.152 sqroot.hex
                5 File      788.048 byte
                2 Directory 18.145.984.512 byte disponibili
```

L'applicazione:
hex2rom.exe

i due file in linguaggio assembler:
sqroot.hex
divmul.hex

altri due file generati al termine della
compilazione:
hex2rom.ilc
hex2rom.pdb

Flusso di Simulazione HW (4)

Dal *Prompt dei comandi*, all'interno della sottocartella *Release*

con il comando

hex2rom sqroot.hex

si ottiene il file

i8051_rom.vhd (per sqroot.hex)

**Questo file consente di creare la descrizione del modulo di memoria ROM del uC.
Il modulo ottenuto rappresenta lo stato della memoria quando vi è caricato sqroot.hex**

mentre, con il comando

hex2rom divmul.hex

si ottiene il file

i8051_rom.vhd (per divmul.hex)

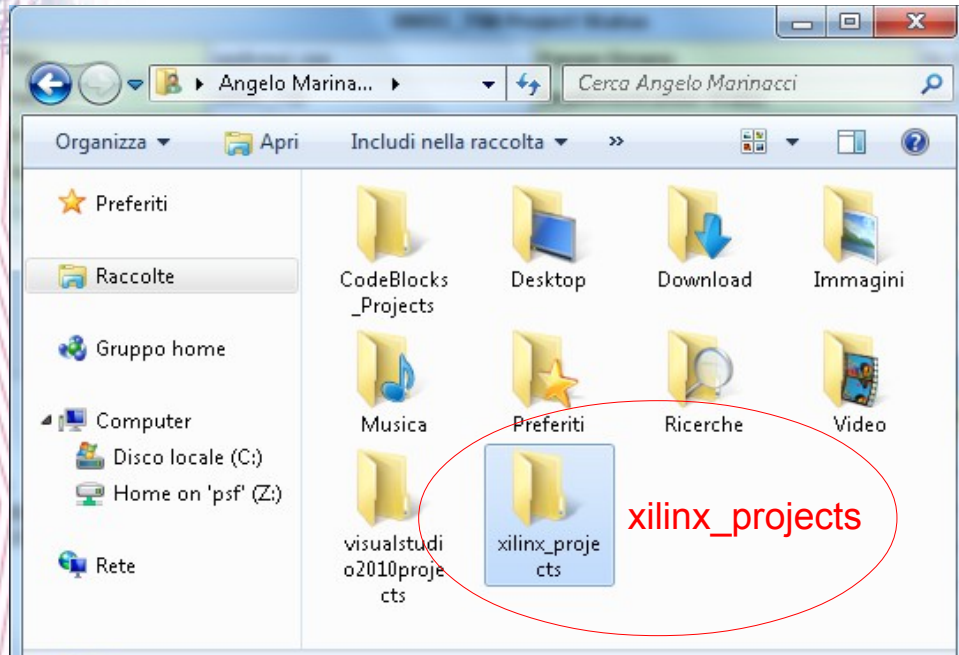
**Questo file consente di creare la descrizione del modulo di memoria ROM del uC.
Il modulo ottenuto rappresenta lo stato della memoria quando vi è caricato divmul.hex**

Nota:

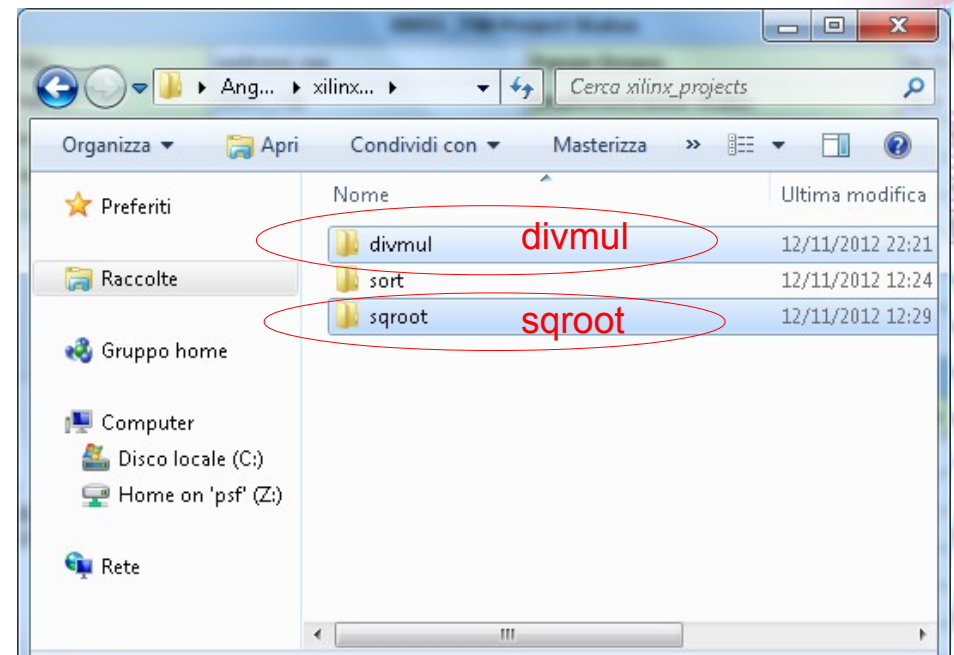
il file di uscita, i8051_rom.vhd, ha sempre lo stesso nome indipendentemente dal .hex che lo origina. E' da evitare la sovrascrittura rendendoli distinguibili.

Flusso di Simulazione HW (5)

Nella directory di lavoro del sw Xilinx, quì **xilinx_projects**



creare due sottocartelle, quì **divmul** e **sqroot**



Ora copiare i files sorgente .vhd ed incollarli in entrambe la sottocartelle.

Prestare attenzione al i8051_rom.vhd: deve essere quello appropriato

i8051_all	22/01/2001 21
i8051_alu	25/07/2003 16
i8051_ctr	22/01/2001 21
i8051_dbg	22/01/2001 21
i8051_dec	22/01/2001 21
i8051_lib	22/01/2001 21
i8051_ram	22/01/2001 21
i8051_rom	12/11/2012 22
i8051_tsb	22/01/2001 21
i8051_xrm	22/01/2001 21

A questo punto, se tutto è andato a buon fine, il contenuto di divmul e sqroot è quello illustrato di fianco.

Notare la data di ultima modifica di i8051_rom.vhd: è notevolmente differente

Flusso di Simulazione HW (6)

File > New Project...

Avviare il tool
Xilinx ISE Design Suite



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location:

Working Directory:

Description:

Select the type of top-level source for the project

Top-level source type:

More Info

Next

Cancel

Location e Working Directory: utilizzare le stesse cartelle create al p.to 2 della slide precedente

Inserire nome progetto: qui chiamati rispettivamente isedivmul e isesqroot

Non modificare questo campo

Next

Nota: è disabilitato se non si assegna il nome e il path

Flusso di Simulazione HW (7)

Al termine della creazione guidata di ciascuno dei due progetti, il contenuto delle due cartelle è il seguente:

Nota: tali scelte sono editabili in qualsiasi momento!

New Project Wizard

Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Product Category	General Purpose
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

More Info

Next

Cancel

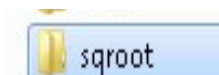
Sceita della FPGA non necessaria per la sola simulazione

Sceita di:
strumento di sintesi (non modificabile)
strumento di simulazione (ISIM)
linguaggio (VHDL)
Non modificare le restanti voci

Al termine per confermare "Next" e dopo "Finish".



- isedivmul
- i8051_all
- i8051_alu
- i8051_ctr
- i8051_dbg
- i8051_dec
- i8051_lib
- i8051_ram
- i8051_rom
- i8051_tsb
- i8051_xrm

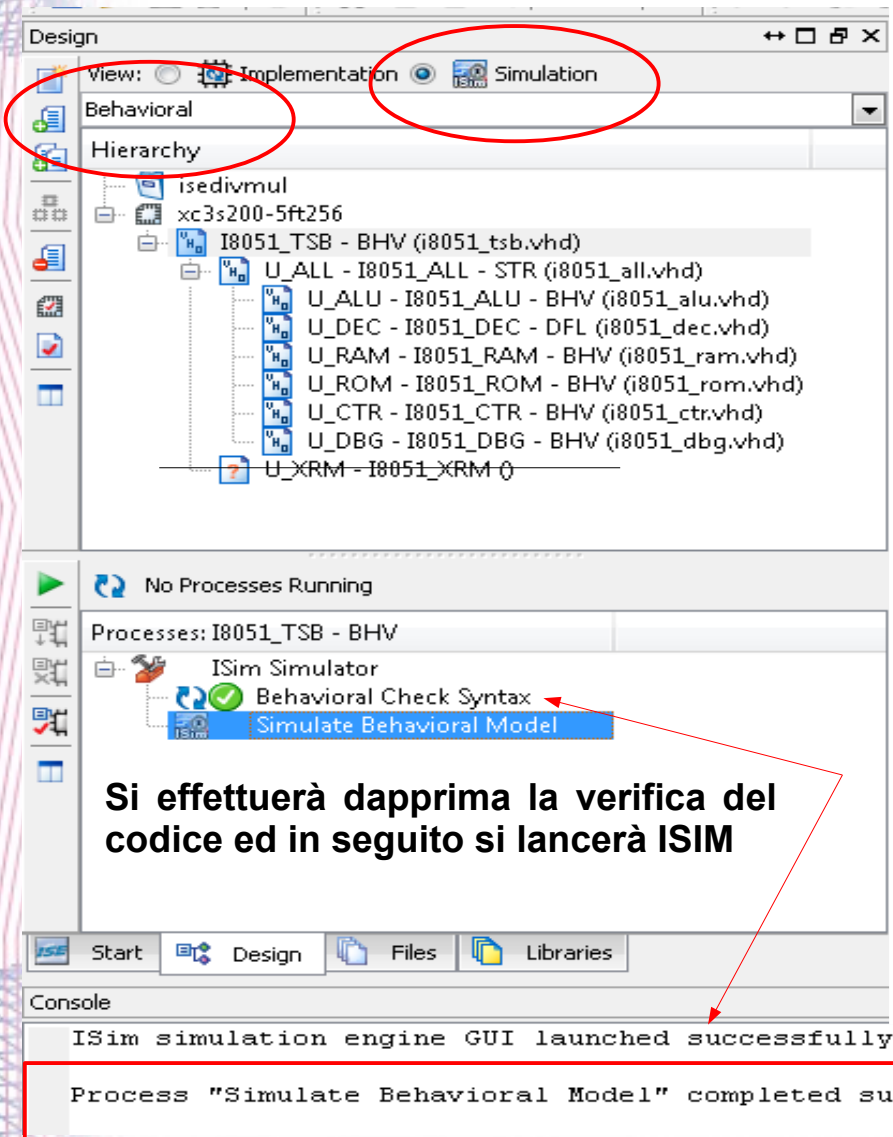


- isesqroot
- i8051_all
- i8051_alu
- i8051_ctr
- i8051_dbg
- i8051_dec
- i8051_lib
- i8051_ram
- i8051_rom
- i8051_tsb
- i8051_xrm

Flusso di Simulazione HW (8)

Behavioral Simulation (Simulazione "Comportamentale")

Con il progetto aperto > Project > Add Source > inserire i files sorgente .vhd quindi la situazione sarà la seguente



I files vengono riconosciuti in automatico così come la gerarchia tra essi:

I8051_TSB... (test-bench)

U_ALL... richiama tutti i moduli costituenti il uC

la ALU

il DECODER istruzioni

La RAM

la ROM (specifica al caso in esame!!)

la CONTROL UNIT (il processore del uC)

U_DBG... fornisce la traccia delle singole istruzioni eseguite

U_XRM... si rimuove poichè non è presente la RAM esterna

Al termine, il simulatore è avviato in automatico dall'ambiente di sviluppo e l'interfaccia grafica è illustrata di seguito

Flusso di Simulazione HW (9)

ISIM G.U.I.

The screenshot displays the ISIM G.U.I. interface with the following components:

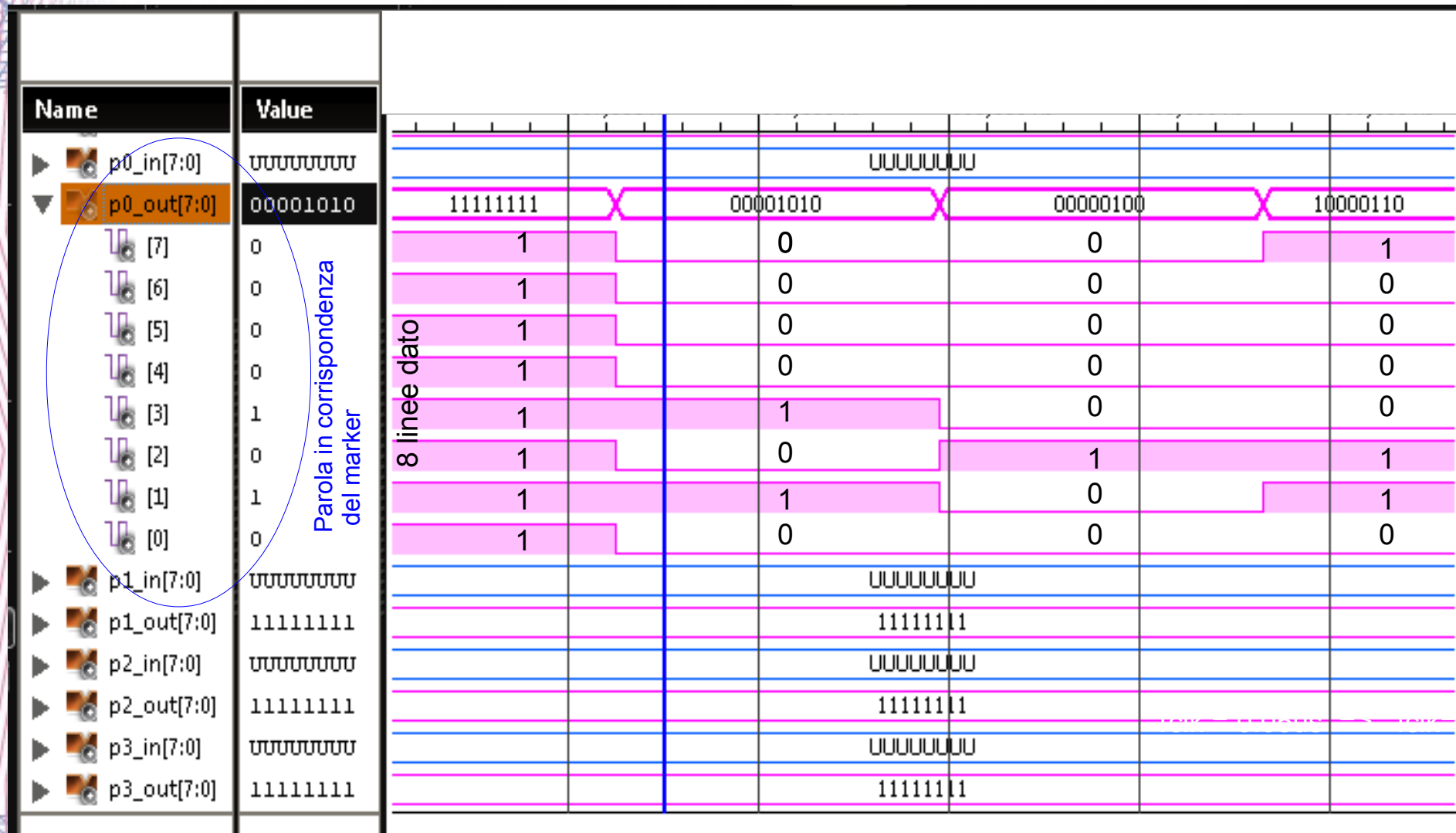
- Instances and Processes:** A tree view on the left showing the hierarchy of components, including `i8051_tsb`, `U_ALL`, `U_ALU`, `U_DEC`, `U_RAM`, `U_ROM`, `U_CTR`, `:114`, `:5201`, `U_DBG`, `:72`, `:73`, `std_logic_1164`, `std_logic_arith`, `textio`, and `i8051_lib`.
- Objects:** A table in the center-left showing simulation objects for `:114`. It lists object names and their values, such as `v8[7:0]` with value `UUUUUUUU`, `rst` with value `1`, and `clk` with value `0`.
- Waveform:** A large area on the right showing a signal waveform. The time axis ranges from `0 us` to `5 us`. A yellow vertical line marks the start of the simulation at `0.000000 us`. Red text indicates the area where signals are visualized.
- Console:** A window at the bottom showing the simulation log. It includes commands like `ISim> # restart` and `ISim> # run all`, along with status messages and timestamps.

Annotations in red text highlight the simulation controls and the signal visualization area:

- Pulsanti di gestione del diagramma temporale** (Buttons for managing the timing diagram)
- Pulsanti di avvio e arresto della simulazione** (Buttons for starting and stopping the simulation)
- Area in cui sono visualizzati i segnali** (Area where signals are visualized)

Flusso di Simulazione HW (10)

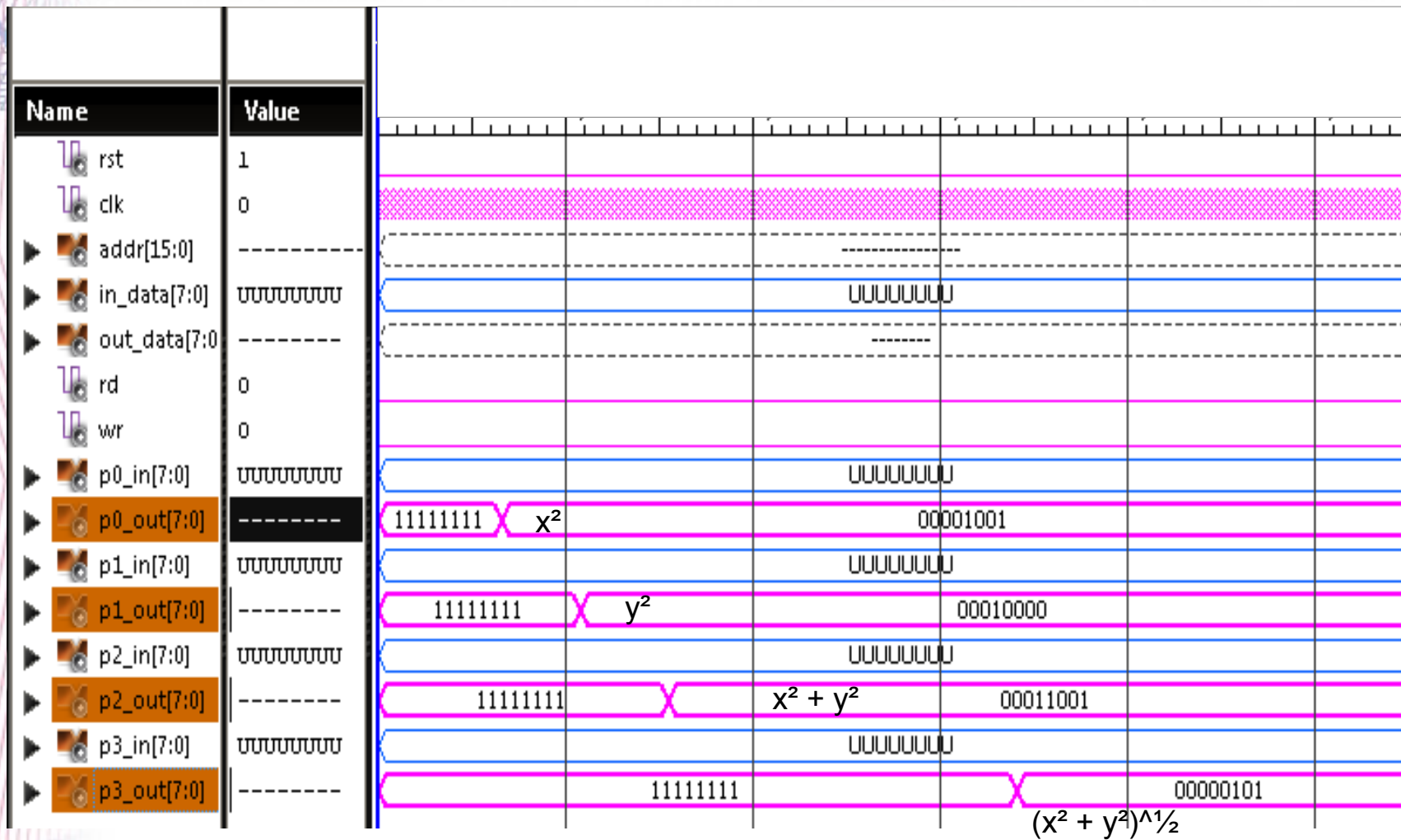
Simulazione del progetto "isedivmul"



Evolve soltanto lo stato della porta P0_OUT; le altre sono in idle (8bit ad 1 se out-port oppure 8bit a 0 se in-port)
 La porta P0_OUT mostra in sequenza il valore esadecimale del quoziente, del resto e del dividendo:
 0Ah = 0000 1010 bcd = 10d; 04h = 0000 0100 bcd = 4d; 86h = 1000 0110 bcd = 134d

Flusso di Simulazione HW (11)

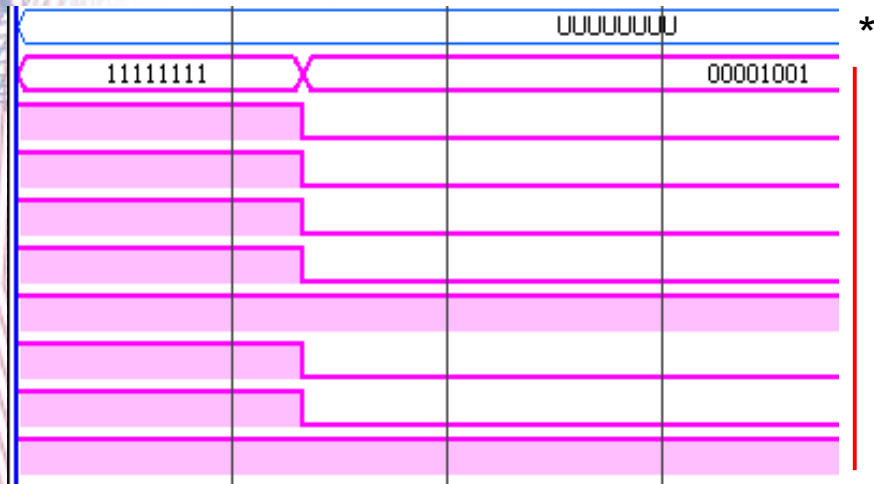
Simulazione del progetto "isesgroot"



Evolgono le 4 porte di uscita in maniera sequenziale da P0_OUT a P4_OUT: sono forniti x^2 , y^2 , la loro somma e la radice quadrata di quest'ultima. A partire dagli istanti temporali in cui si verificano le transizioni di stato dei bus ad 8 bit sulle rispettive porte il dato è "pronto". La presenza di rising&falling edges sta ad indicare il bus e non la singola linea.

Flusso di Simulazione HW (12)

Simulazione del progetto "isesqroot"

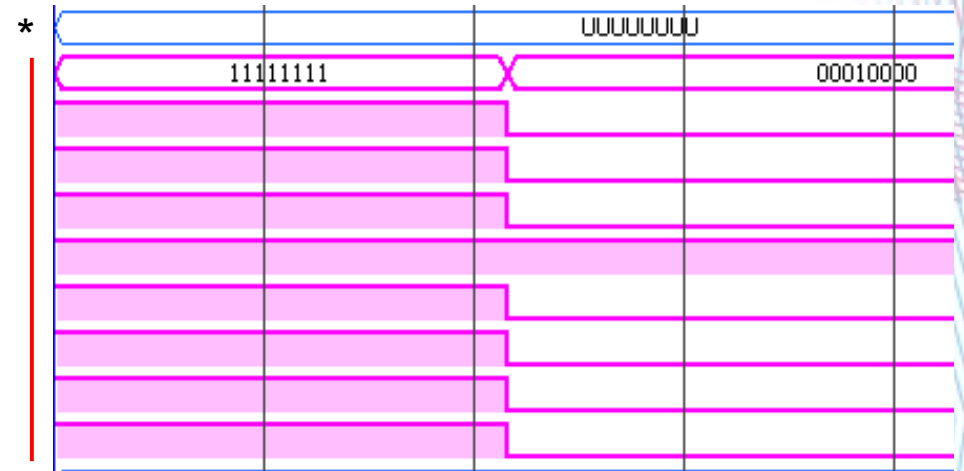


P0_IN (*) & P0_OUT

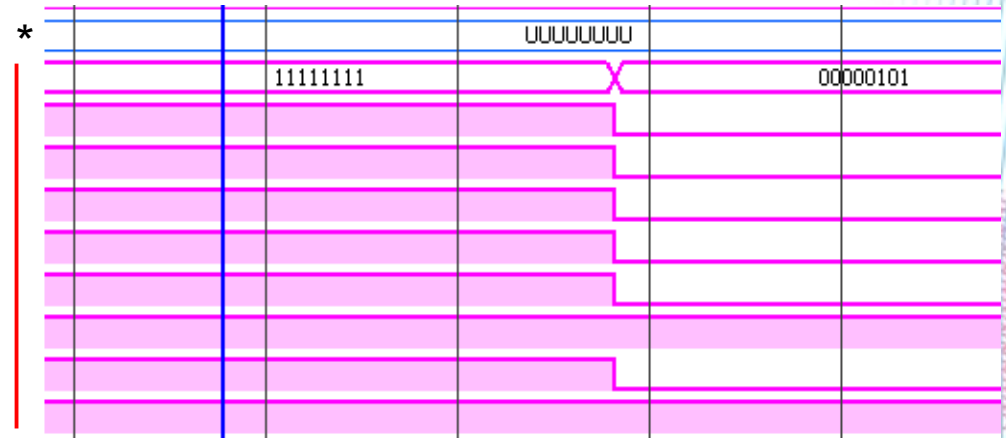


P2_IN (*) & P2_OUT

P1_IN (*) & P1_OUT



P1_IN (*) & P3_OUT



Flusso di Simulazione HW (13)

definizione del clock

Source file

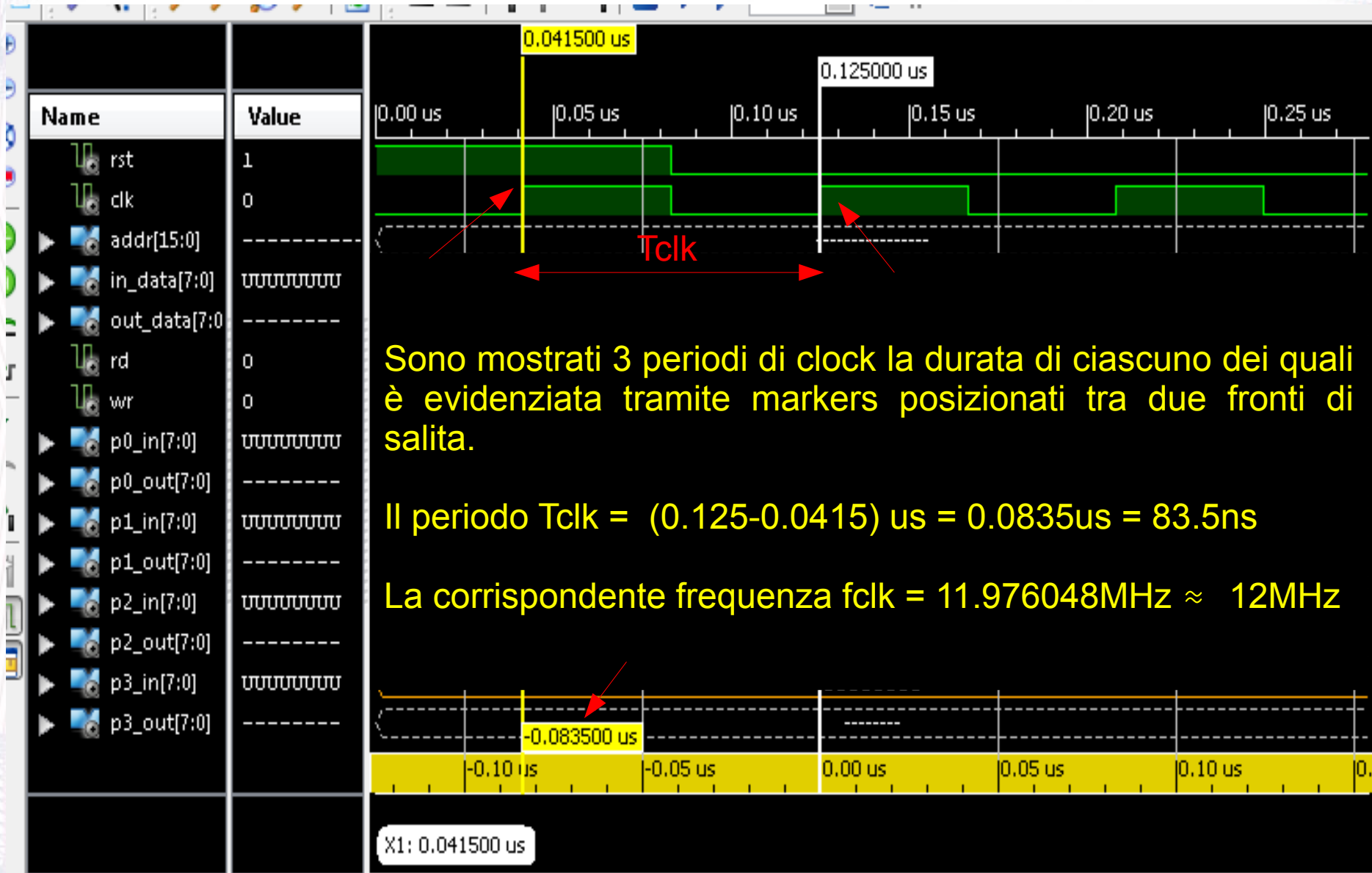
testbench.vhd

```
62     signal p0_in    : UNSIGNED (7 downto 0);
63     signal p0_out   : UNSIGNED (7 downto 0);
64     signal p1_in    : UNSIGNED (7 downto 0);
65     signal p1_out   : UNSIGNED (7 downto 0);
66     signal p2_in    : UNSIGNED (7 downto 0);
67     signal p2_out   : UNSIGNED (7 downto 0);
68     signal p3_in    : UNSIGNED (7 downto 0);
69     signal p3_out   : UNSIGNED (7 downto 0);
70     begin
71
72         rst <= '0' after 83.33333333 ns;
73         clk <= not clk after 41.66666667 ns;
74
75         U_ALL : I8051_ALL port map (rst, clk,
76                                     addr, out_data, in_data, rd, wr,
77                                     p0_in, p0_out, p1_in, p1_out,
78                                     p2_in, p2_out, p3_in, p3_out);
79
80         U_XRM : I8051_XRM port map (rst, clk, addr, out_data, in_data, rd, wr);
81
82     end BHV;
83
84     -----
85
86     configuration CFG_I8051_TSB of I8051_TSB is
87         for BHV
88         end for;
```

i8051_tsb.vhd

Flusso di Simulazione HW (14)

traccia del clock





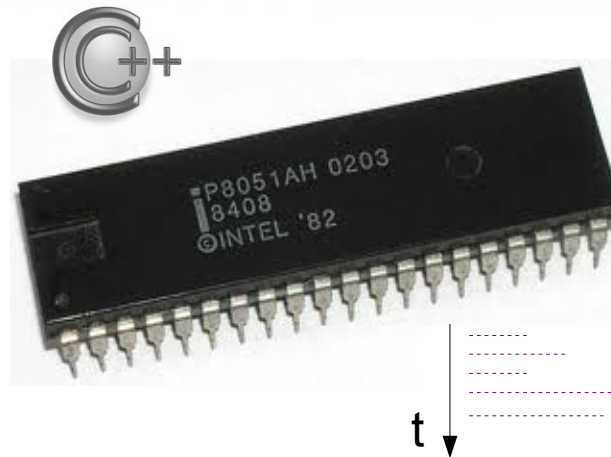
Programmabilità & Configurabilità

Programmabilità sequenzialità

e Configurabilità (1) ma anche parallelismo

Il microcontrollore 8051 è un dispositivo programmabile sequenziale:

il codice ad alto livello è eseguito temporalmente nell'ordine in cui è scritto.



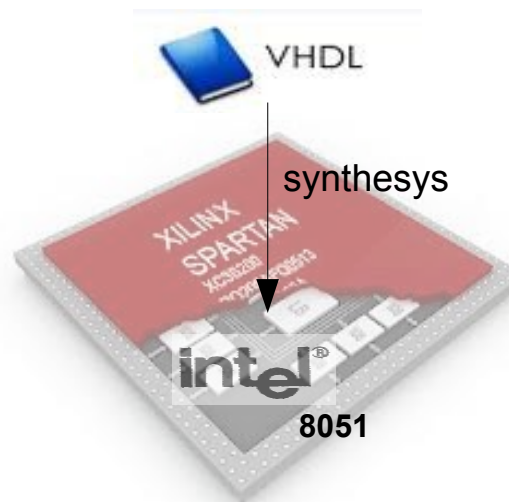
Una riga del file sorgente è eseguita solo al completamento della riga precedente anche se i dati di "ingresso" allo step attuale non dipendono dai risultati ottenuti allo step precedente. In questo senso i uP, uC possono considerarsi SEQUENZIALI.

Attenzione!

Non confondere con una rete logica sequenziale che è una rete avente memoria nella quale lo stato attuale dell'uscita dipende dallo stato attuale degli ingressi e dallo stato precedente dell'uscita, ad es. Flip-flop.

La FPGA è un array 2D di porte logiche con il grosso vantaggio che vi si possono sintetizzare dispositivi a logica combinatoria e/o sequenziale e/o programmabile.

Questo viene fatto CONFIGURANDO il GATE ARRAY al suo interno e NON programmandola!



L'ordine di scrittura in hdl non è sempre importante quando vengono scritti i componenti ed i collegamenti elettrici - sì, i fili! - tra essi.

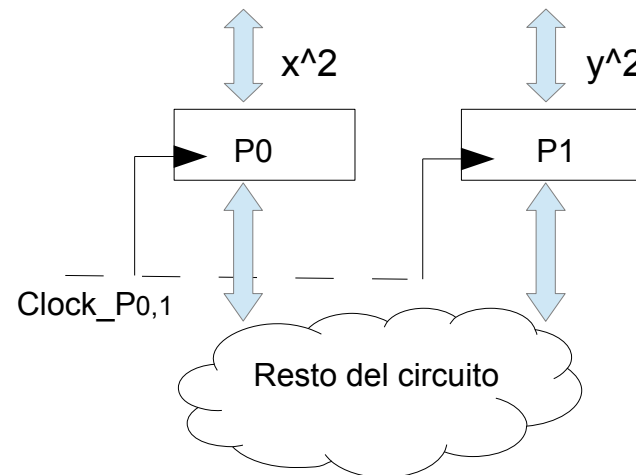
Ad es. si consideri il file sqroot.c

Viene fornito in uscita dapprima il dato sulla porta P0 ed in seguito su P1. Equivalentemente se le si ipotizzassero come porte di ingresso e per come è scritto il codice, nel pieno rispetto temporale di quest'ultimo, si leggerebbe prima dalla porta P0 e solo poi da P1.

```
#include <reg51.h>
#include <math.h>
void main() {
    float x = 3.0;
    float y = 4.0;
    float xx, yy, xx_yy,
          sqrt_xx_yy;
    xx = x * x;
    P0 = (unsigned char)xx;
    yy = y * y;
    P1 = (unsigned char)yy;
    xx_yy = xx + yy;
    P2 = (unsigned char)xx_yy;
    sqrt_xx_yy = sqrt(xx_yy);
    P3 =
(unsignedchar)sqrt_xx_yy;
while(1);
}
```

Ragionando in hw le due porte, P0 e P1 sono considerati registri PIPO, sono fisicamente differenti ed i rispettivi termini 3^2 e 4^2 , nell'operazione $\sqrt{3^2+4^2}$, sono indipendenti tra loro ed inoltre sono “dati pronti” nello stesso istante temporale.

Non c'è ragione di dover fornire dapprima x^2 e solo poi y^2 , se non per il fatto che non si può fare altrimenti a causa del file sorgente .c e della tipologia di device, uC!



P0 e P1 si possono leggere/scrivere simultaneamente, nello stesso istante temporale.

In tal senso, in hw è possibile l'esecuzione **PARALLELA**

Perchè allora anche in FPGA l'esecuzione non è parallela (vedi simulazione ISE)?

Essa è configurata per funzionare da uC Intel-8051: dispositivo programmabile ad esecuzione sequenziale.

Conclusioni

In Conclusione (1)

La simulazione dell'I.S.S. che esegue un programma di test è veloce temporalmente ed il livello di dettaglio è abbastanza ridotto: info statistiche in merito all'esecuzione appena terminata e dati alle porte solo quando “pronti”.

Il tempo di simulazione è di qualche secondo ma si riesce a simulare un tempo di esecuzione, a bordo del uC, diverso a seconda di quanto si impegna il calcolatore: da decimi di secondo nel caso peggiore di dettaglio nel rapporto testuale, a centinaia di secondi nel caso di non generazione del rapporto a fine simulazione.

In tutte le situazioni analizzate nella parte “Flusso di Simulazione SW” si è considerato sempre il clock del 8051 alla frequenza massima di lavoro ammessa: $f_{clk} = 12\text{MHz}$.

In Conclusione (2)

La simulazione in ISE è notevolmente più complessa: è mostrata l'evoluzione nel tempo di ogni linea dato/controllo che è presente nei modelli hdl dei singoli elementi costituenti il uC!

Tuttavia, per brevità, ci si è soffermati soltanto sulla analisi dell'evoluzione temporale delle linee dato delle porte IO poichè tali info sono visibili anche dalla simulazione dell'I.S.A. mediante I.S.S.

Nei modelli HDL il contenuto informativo nonché il livello di dettaglio consentono di studiare il funzionamento del dispositivo (la cui ROM differisce in termini di contenuto da hex-file in hex-file) ed anche capire, bit a bit, cosa “significa” una istruzione all'interno del particolare hw eventualmente riprodotto.

In Conclusione (3)

La complessità implica un maggior carico computazionale per la macchina sulla quale la simulazione gira. Tuttavia anche le simulazioni in ISE risultano abbastanza veloci fino al punto in cui il dato sulle porte è stabile, “pronto”. La simulazione una volta lanciata viene stoppata dopo circa 10s e si riscontra che il tempo di “funzionamento” simulato è “appena”

Sim Time: 157,816,183,234 ps

dunque circa 160ms reali (0.16s). Tale situazione è simile al caso dell'ISS in cui le #define... sono abilitate (caso di massimo dattaglio).

L'8051 inoltre può operare con frequenza di clock variabile ed è possibile realizzare tale condizione soltanto nei modelli HDL editando il file testbench.vhd

Symbol	Parameter	12 MHz Oscillator		Variable Oscillator		Units
		Min	Max	Min	Max	
1/TCLCL	Oscillator Frequency			3.5	12.0	MHz

Tutt'altro per la sintesi: il processo è molto più oneroso computazionalmente ed il tempo richiesto non è dell'ordine dei secondi bensì dei minuti, anche quasi una decina a seconda del modulo sintetizzato, pur utilizzando calcolatori con prestazioni di tutto rispetto.

Grazie per l'attenzione!

That's all Folks!

Angelo Marinacci