

# Sviluppo di SoC su FPGA con profiling dell'applicazione

## Esercitazione 10 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania  
Dipartimento di Matematica e Informatica  
Corso di Laurea Magistrale in Informatica, AA 2019-20

### Indice

1. Sviluppo di SoC su FPGA con profiling dell'applicazione
2. argomenti dell'esercitazione
3. integrazione di sistema nello sviluppo di SoC
4. strumenti per il profiling di un'applicazione software
5. costruzione di un sistema Nios II con performance counter
6. un esempio semplice e ben noto
7. uso del performance counter nell'applicazione software
8. generazione del BSP e integrazione HW/SW
9. debugging ed esecuzione
10. esperienza di laboratorio
11. riferimenti

in questa esercitazione si trattano:

- integrazione di componenti hardware per lo sviluppo di SoC mediante Qsys
- strumenti per il *profiling* di applicazioni software
- progetto di un sistema Nios II dotato di un *performance counter*
- uso dell'API del *performance counter* in un esempio ben noto: calcolo del delay su una sequenza di traiettorie di Collatz, con input di utente della lunghezza della sequenza
- integrazione HW/SW mediante generazione del BSP nel Monitor Program, compilazione, caricamento su FPGA, debugging ed esecuzione
- esperienza di laboratorio:
  - progetto e realizzazione di un sistema HW/SW con struttura e caratteristiche simili a quelle dell'esempio mostrato in questa esercitazione, adoperando gli stessi strumenti di sviluppo e *profiling*, ma per un'applicazione diversa

### integrazione di sistema nello sviluppo di SoC

lo sviluppo di un SoC con applicazioni è una tipica attività di codesign HW/SW consiste di progetto e sviluppo di componenti di entrambi i tipi e la loro *integrazione* a formare un sistema

lo strumento di Quartus usato in questa esercitazione per l'integrazione di componenti hardware nello sviluppo di SoC è Qsys

si consiglia di consultare l'introduzione a Qsys e di ri-eseguire sulla DE1-SoC l'esempio ivi mostrato, come proposto nella prima parte dell'esercitazione precedente, per acquisire familiarità con l'uso dello strumento

un esempio un po' più complesso è oggetto di questa esercitazione:

- sviluppo di un SoC simile a quello indicato ma dotato di un componente per il *profiling* accurato di applicazioni software (v. appresso)
- sviluppo di una realizzazione software del calcolo del delay di traiettorie di Collatz, e di un'applicazione software per la misura del suo tempo di esecuzione

## strumenti per il profiling di un'applicazione software

profiling di un programma: misura del tempo speso in diverse parti del programma, per identificarne quelle critiche per la velocità di esecuzione

utile nel codesign HW/SW per valutare quali parti di un programma siano meritevoli di eventuale accelerazione hardware, e stimare quindi il guadagno di prestazione conseguibile

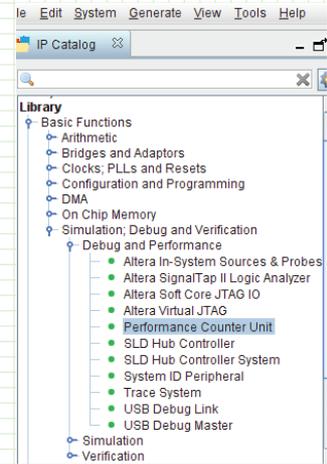
tre strumenti considerati nel documento (datato) *Profiling Nios II Systems*:

*GNU gprof*: misura software, alto sovraccarico di risorse software, alta distorsione della misura

*Interval Timer*: misura hardware, minimo sovraccarico di risorse, distorsione limitata

*Performance Counter Unit*: misura hardware, significativo sovraccarico di risorse hardware, distorsione minima, limite (7) sul numero di sezioni misurabili

in questa esercitazione usiamo il terzo metodo, che dà la precisione migliore, l'uso più semplice nel programma, e poiché il detto limite non è un problema per l'applicazione in gioco



## costruzione di un sistema Nios II con performance counter

la figura mostra lo schema Qsys del sistema Nios II con Performance Counter Unit  
lo schema è simile a quello dell'esempio nell'introduzione a Qsys, eccetto per l'assenza della PIO LEDs e per la presenza del componente per il profiling

Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk_0	Clock Source		exported			
	clk_in	Clock Input	clk				
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	clk_0				
	clk_reset	Reset Output	reset				
	nios2_qsys_0	Nios II (Classic) Processor					
	clk	Clock Input	clk_0				
	reset_n	Reset Input	reset				
	data_master	Avalon Memory Mapped Master	[clk]				
	instruction_master	Avalon Memory Mapped Master	[clk]				
	d_irq	Interrupt Receiver	clk_0		IRQ 0	IRQ 31	
	jtag_debug_module	Reset Output	clk_0				
	jtag_debug_module	Avalon Memory Mapped Slave	clk_0		0x0002_0800	0x0002_0fff	
	custom_instruction	Custom Instruction Master	clk_0		0x0000_0000	0x0001_ffff	
	onchip_memory2_0	On-Chip Memory (RAM or ROM)	clk_0				
	clk1	Clock Input	clk_0				
	s1	Avalon Memory Mapped Slave	[clk1]				
	reset1	Reset Input	reset				
	performance_count...	Performance Counter Unit	clk_0				
	clk	Clock Input	clk_0				
	reset	Reset Input	reset				
	control_slave	Avalon Memory Mapped Slave	clk_0		0x0002_1000	0x0002_103f	
	jtag_uart_0	JTAG UART	clk_0				
	clk	Clock Input	clk_0				
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0		0x0002_1050	0x0002_1057	
	irq	Interrupt Sender	clk_0				
	switches	PIO (Parallel I/O)	clk_0				
	clk	Clock Input	clk_0				
	reset	Reset Input	reset				
	s1	Avalon Memory Mapped Slave	clk_0		0x0002_1040	0x0002_104f	
external_connection	Conduit	switches					

un esempio semplice e ben noto

la funzione C in figura è una realizzazione software del calcolo del delay di una traiettoria di Collatz di dato inizio

le direttive di preprocessing, tranne la quarta, fanno riferimento alla piattaforma hardware precedentemente costruita con Qsys, v. appresso

```
delay_collatz_timing.c + *
1 #include "altera_avalon_performance_counter.h"
2 #include "system.h"
3 #define switches (volatile unsigned int *) SWITCHES_BASE // from "system.h"
4 #define N_FACTOR (unsigned int const) 8 // *switches scale factor
5 #define pca (void *) PERFORMANCE_COUNTER_0_BASE // from "system.h"
6
7 unsigned int delay_collatz(unsigned int x0) {
8     int d = 0;
9     int x = x0;
10    int hx;
11    while (x > 1) {
12        d++;
13        hx = x >> 1;
14        if ((x % 2) > 0) {
15            d++;
16            x += hx + 1;
17        }
18        else
19            x = hx;
20    }
21    return d;
22 }
```

uso del performance counter nell'applicazione software

a differenza delle precedenti esperienze di laboratorio relative a realizzazioni hardware della funzione considerata, qui l'input di utente determina la lunghezza della sequenza di traiettorie da generare nel programma principale, ovvero il numero di invocazioni della funzione

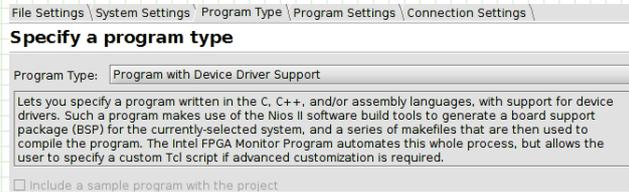
all'input dagli switch si applica un fattore di scala per avere una durata ragionevole del test

```
delay_collatz_timing.c + *
24 int main() {
25     unsigned int i, n, taps, c, x, d;
26     n = *switches << N_FACTOR;
27     alt_u32 cpu_freq = alt_get_cpu_freq();
28     unsigned int seed = 0x1234;
29     c = seed;
30     PERF_RESET(pca);
31     PERF_START_MEASURING(pca);
32     for (i = 0; i < n; i++) {
33         PERF_BEGIN(pca, 1);
34         x = c;
35         taps = (c & 0x1) ^ ((c & 0x4) >> 2) ^ ((c & 0x8) >> 3) ^ ((c & 0x20) >> 5);
36         c = (taps << 15) | (c >> 1);
37         PERF_END(pca, 1);
38         PERF_BEGIN(pca, 2);
39         d = delay_collatz(x);
40         PERF_END(pca, 2);
41     }
42     PERF_STOP_MEASURING(pca);
43     perf_print_formatted_report(
44         pca, // peripheral's HW base address
45         cpu_freq, // CPU frequency (Hz)
46         2, // how many sections to print
47         "traject_start", // display-names of sections
48         "delay_collatz");
49     return 0;
50 }
```

## generazione del BSP e integrazione HW/SW

le direttive di preprocessing mostrate prima abilitano l'uso dell'API del *performance counter* e di altri simboli (SWITCHES\_BASE in questo caso) definiti nell'interfaccia software del sistema costruito con Qsys

l'interfaccia è fornita dal BSP, la cui costruzione è qui automatizzata dal *Monitor Program*, in seguito alla scelta del tipo di programma *Program with Device Driver Support*



ulteriori aspetti del BSP (e.g. opzioni di compilazione, di linking ecc.) possono essere specificati fornendo uno script Tcl custom

in particolare, mentre il livello di ottimizzazione di default stabilito dal *Monitor Program* è *-O1*, un livello diverso, e.g. *-O3*, può aversi creando uno script (con estensione *.tcl*) di una sola riga:

```
set_setting hal.make.bsp_cflags_optimization -O3
```

e indicandone il percorso nella input box *BSP settings Tcl script (optional)* delle impostazioni *Program Settings*

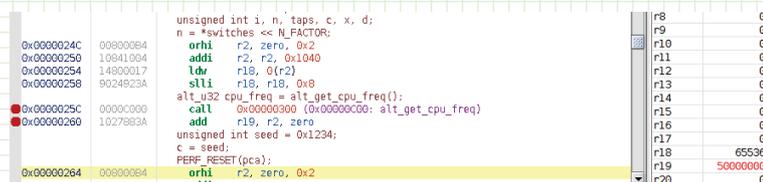
## debugging ed esecuzione

il debugging nel *Monitor Program* può essere condotto anche al livello del sorgente C (visualizzazione dei valori delle variabili)

per questo è necessaria la compilazione con livello di ottimizzazione *-O0*

rimane comunque accessibile il disassembly del programma, in cui porre breakpoint ed esaminare lo stato dell'esecuzione in punti critici per la verifica della sua correttezza

per esempio, con i breakpoint in figura si può verificare la correttezza del calcolo del n. di iterazioni e della frequenza di clock, risp. in r18 e r19



dopo la rimozione di tutti i breakpoint, reset del sistema e rilancio dell'esecuzione, il modulo di profiling genera il *performance report* mostrato in figura

```
Terminal
JTAG UART link established using cable "DE-SoC [2-1.2]", device 2, instance 0x00
--Performance Counter Report--
Total Time: 7.52118 seconds (376058945 clock-cycles)
+-----+-----+-----+-----+
| Section | % | Time (sec) | Time (clocks) | Occurrences |
+-----+-----+-----+-----+
| traject_start | 2.53 | 0.19005 | 9502720 | 65536 |
+-----+-----+-----+-----+
| delay_collatz | 96.3 | 7.24331 | 362165262 | 65536 |
+-----+-----+-----+-----+
```

si propone di progettare e realizzare un sistema HW/SW con struttura e caratteristiche simili a quelle dell'esempio mostrato in questa esercitazione, adoperando gli stessi strumenti di sviluppo e *profiling*, ma per un'applicazione diversa; precisamente, il lavoro consiste di:

- costruzione di un sistema Nios II su FPGA, dotato di un componente per il *profiling* dell'applicazione
- sviluppo software di un programma per il test di coprimialità su una sequenza di una coppia di numeri, usando una funzione di calcolo del GCD
- integrazione HW/SW del sistema e dell'applicazione, con: *profiling* di quest'ultima mediante l'API del componente di cui sopra, generazione del BSP, compilazione, caricamento ed esecuzione su FPGA, con eventuale debugging se necessario

## riferimenti

letture raccomandate:

*Introduction to the Qsys System Integration Tool - For Quartus Prime 16.1*,  
Intel Corp. - FPGA University Program, November 2016

letture per ulteriori approfondimenti:

*Profiling Nios II Systems, AN-391-3.0*, Altera Corp., July 2011

materiali utili per l'esperienza di laboratorio proposta:

*Performance Counter Unit Core, Ch. 36 in: Embedded Peripherals IP User Guide*, Intel Corp., UG-01085 | 2019.12.16

*Intel FPGA Monitor Program Tutorial for Nios II - For Quartus Prime 16.1*,  
Intel® FPGA University Program (November 2016)

file sorgenti per l'esperienza di laboratorio