Realizzazione su FPGA di un coprocessore mappato in memoria

Esercitazione 11 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2017-18

Indice

argomenti dell'esercitazione

in questa esercitazione si trattano:

- realizzazione su FPGA dell'idea di progetto proposta nella lezione 11
  - interfaccia hardware del coprocessore
  - coprocessore quale componente Qsys
  - sistema Nios II con coprocessore e Performance Counter
  - driver software
- test e misura delle prestazioni con il Monitor Program
  - test con accelerazione bloccante
  - test con accelerazione non bloccante

flusso di lavoro del progetto

fasi principali di sviluppo:

- descrizione in VHDL del coprocessore con interfaccia Avalon MM
- costruzione Qsys di un sistema Nios II con coprocessore e performance counter
- mapping del sistema su FPGA e compilazione
- stesura di script TCL per la generazione del driver software HAL
- stesura dell'applicazione software per test e misura della prestazione, in due versioni:

  sequenziale: esecuzione bloccante del calcolo nel coprocessore

  pipelined: esecuzione non bloccante del calcolo nel coprocessore
- compilazione ed esecuzione dell'applicazione mediante Monitor Program, per due varianti di ciascuna versione: una con valore di default del livello di ottimizzazione, l'altra con livello O3
- salvataggio dei performance report e archiviazione del progetto

interfaccia hardware del coprocessore

due sorgenti VHDL realizzano il coprocessore memory-mapped:

- delay_collatz.vhd, versione modificata dell'output del traduttore fdlvhd dal sorgente Gezel presentato nella seconda lezione, come da terza esperienza di laboratorio
- delay_collatz_interface.vhd, che incorpora un'istanza del componente di calcolo e accede ai seguenti segnali del bus Avalon: clock, resetn, read, write, chipselect, waitrequest, writedata, readdata

entrambi i sorgenti sono disponibili nella cartella vhdl dell'archivio allegato, nonché nella cartella VHDL/code/e11 dell'area riservata di laboratorio

> la cartella contiene anche std_logic_arithext.vhd, necessario alla compilazione del componente di calcolo, delay_collatz_codesign.vhd, di cui si dirà appresso, e delay_collatz.diff, quest'ultimo solo per documentazione delle modifiche apportate alla traduzione prodotta da fdlvhd

la consultazione del sorgente delay_collatz_interface.vhd mostra le relazioni tra i segnali di I/O del componente di calcolo e i segnali all'interfaccia Avalon

coprocessore quale componente Qsys (1)

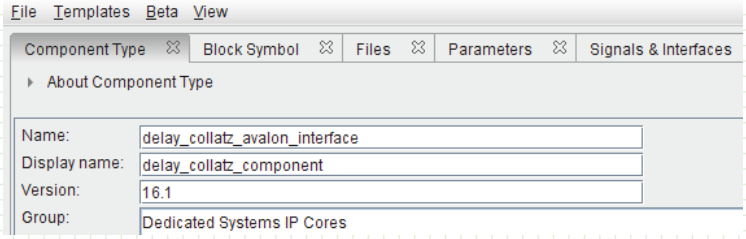
la cartella codesign nell'archivio allegato è predisposta per ospitare lo sviluppo del progetto

> dopo avervi copiato i file *.vhd dalla cartella vhdl, la costruzione del componente custom Qsys procede come nel tutorial visto nell'esercitazione 10, con le dovute differenze del caso

creato il progetto delay_collatz_codesign, con omonima entità top-level, si procede alla creazione del componente custom delay_collatz_interface

> in particolare, non occorre dotarlo dell'interfaccia Avalon Conduit, poiché il componente non usa periferiche esterne alla FPGA

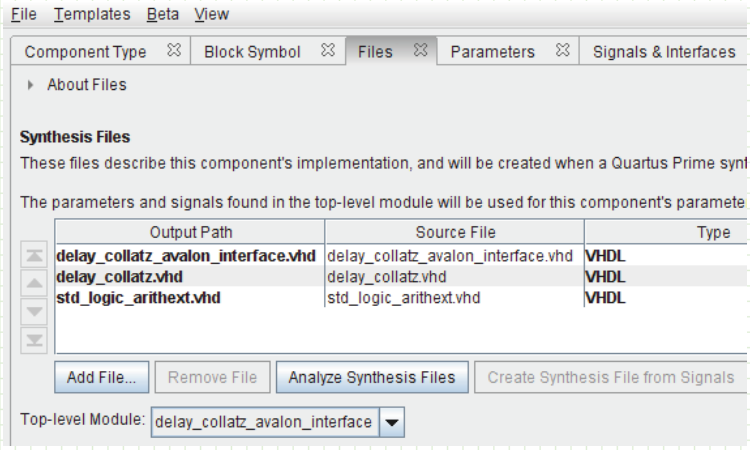
la definizione del nuovo tipo di componente è mostrata in figura



coprocessore quale componente Qsys (2)

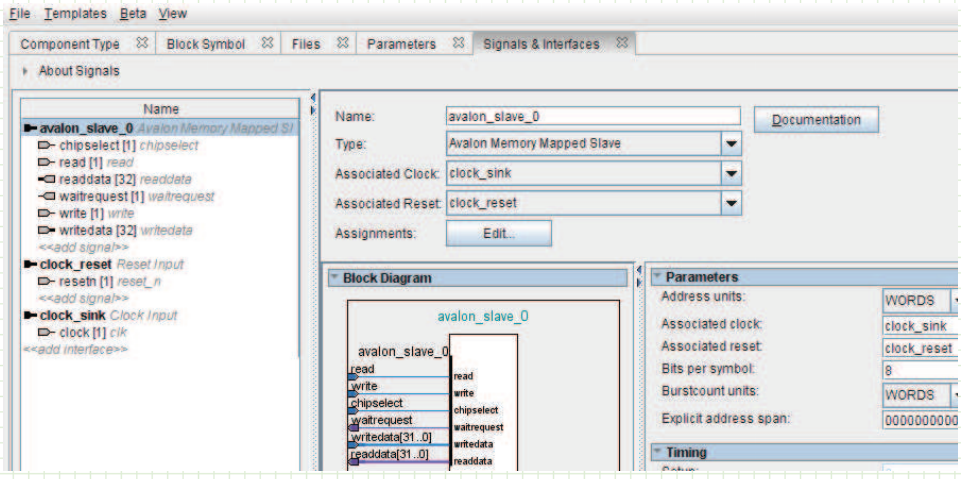
si procede quindi all'assegnazione dei file VHDL che descrivono il componente e alla loro analisi, come mostrato in figura

N.B. per questo progetto non occorre copiare i file per la simulazione

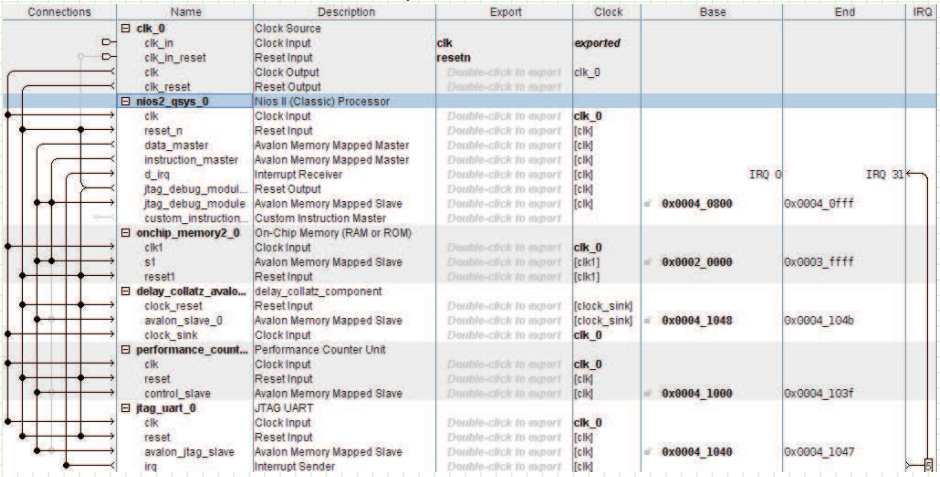


coprocessore quale componente Qsys (3)

infine, si conclude la definizione del nuovo tipo di componente con la definizione delle sue interfacce Avalon e la collocazione dei segnali nelle interfacce appropriate, come illustrato in figura



sistema Nios II con coprocessore e Performance Counter



System Contents | Address Map | Interconnect Requirements

System: unsaved **Path:** nios2_qsys_0

	nios2_qsys_0.data_master	nios2_qsys_0.instruction_master
nios2_qsys_0.jtag_debug_module	0x0004_0800 - 0x0004_0fff	0x0004_0800 - 0x0004_0fff
onchip_memory2_0.s1	0x0002_0000 - 0x0003_ffff	0x0002_0000 - 0x0003_ffff
delay_collatz_avalon_interface_0....	0x0004_1048 - 0x0004_104b	
performance_counter_0.control_sl...	0x0004_1000 - 0x0004_103f	
jtag_uart_0.avalon_jtag_slave	0x0004_1040 - 0x0004_1047	

mapping su FPGA e compilazione

per la costruzione del sistema Nios II illustrato nelle figure precedenti può essere utile la consultazione del tutorial di introduzione a Qsys

con alcune differenze, e.g. la dimensione della memoria è di 128 KB in questo caso, gli indirizzi di base sono tutti assegnati dal sistema ecc.

i passi finali per il mapping del sistema sulla FPGA sono i seguenti:

in Qsys:

- salvare il sistema con il nome embedded_system in File > Save As...
- generare il suo codice VHDL mediante Generate > Generate HDL...

uscire da Qsys, quindi in Quartus:

- assegnare il file embedded_system.qip (in embedded_system/synthesis) al progetto
- importare gli assegnamenti dal file DE1_SoC.qsf nella cartella de1soc dell'archivio allegato
- File > Save Project
- compilare delay_collatz_codesign.vhd

driver software

la cartella script nell'archivio allegato contiene due script TCL per la generazione del driver software nel BSP del progetto

- i due script differiscono solo per un comando, presente in uno di essi, che prescrive il livello O3 di ottimizzazione invece del livello O1 di default

questi script vanno copiati nella cartella codesign/ip/delay_collatz_avalon_interface

- nella stessa cartella, rispettivamente in HAL/inc e HAL/src, vanno copiati i sorgenti C delay_collatz_avalon_interface.h e delay_collatz_avalon_interface.c del driver software contenuti nella cartella src dell'archivio allegato

gli script TCL sono stati scritti in analogia allo script TCL del driver software del Performance Counter, reperibile nella distribuzione del software Quartus Prime Lite 16.1 al percorso
$SOPC_KIT_NIOS2/../ip/altera/sopc_builder_ip/altera_avalon_performance_counter

- similmente, i sorgenti C del driver software sono stati scritti in (più limitata) analogia con i sorgenti C del driver software dello stesso IP Core, nella cartella HAL al suddetto percorso

la motivazione per questo modo, forse poco ortodosso, di produzione del driver software sta nel duplice fatto che

- l'interfaccia Avalon del componente custom non rientra in alcuna delle classi di modelli di dispositivi generici HAL di cui al Cap. 7 del manuale di sviluppo software per Nios II Classic
- come pure non vi rientra quella del Performance Counter Unit IP Core ...

a cui si aggiunge un discreto grado di analogia operativa dei due componenti

- tuttavia si raccomanda la consultazione del Cap. 7 del suddetto manuale, per acquisire una migliore comprensione della struttura e del contenuto del driver software

programmi di test e misura delle prestazioni (1)

la cartella src nell'archivio allegato contiene i programmi in questione, da copiare nelle cartelle di creazione dei progetti di test e misura delle prestazioni mediante il Monitor Program, come segue:

- delay_collatz_sequential_timing.c in codesign/amp_s e in codesign/amp_s_o3
- delay_collatz_pipelined_timing.c in codesign/amp_p e in codesign/amp_p_o3

i parametri di creazione dei progetti sono indicati nel file allegato MonitorNotes.txt

- occorre alimentare la DE1-SoC e collegarla al PC perché la creazione di ciascun progetto si possa concludere con la programmazione della FPGA

differenze principali tra il sorgente dell'esercitazione 09 e l'attuale versione sequenziale:

- direttive #include e #define relative al componente custom
- sostituzione dell'input da dispositivo switches con una costante
- rimpiazzamento del corpo della funzione delay_collatz con due istruzioni del driver software del componente custom

programmi di test e misura delle prestazioni (2)

la versione *pipelined* del programma presenta differenze molto più marcate rispetto al programma dell'esercitazione 09:

- si rende *non bloccante* l'interazione con l'hardware custom sostituendo la chiamata della funzione delay_collatz con un *inlining* del suo corpo, dove però si ricolloca fra le due istruzioni, rispettivamente di avvio del calcolo hardware e di lettura del risultato, il calcolo in software del punto di inizio della traiettoria successiva

il meccanismo di sincronizzazione è molto semplice, grazie a proprietà del componente custom e del segnale waitrequest del protocollo Avalon MM:

- per traiettorie di durata più breve del calcolo software, il componente custom mantiene il risultato nel registro interno e attende il comando di lettura
- per traiettorie di durata più lunga del calcolo software, il comando di lettura viene tenuto in attesa all'interfaccia Avalon mediante il segnale waitrequest

test con accelerazione bloccante

la compilazione, caricamento sulla FPGA ed esecuzione del programma delay_collatz_sequential_timing.c, nei due progetti codesign/amp_s e codesign/amp_s_o3 produce i Performance Counter Report in figura

- la notevole riduzione del tempo di esecuzione della sezione delay_collatz nella seconda variante si spiega con l'*inlining* della funzione nella compilazione O3

Terminal

```
--Performance Counter Report--
Total Time: 0.499495 seconds  (24974733 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|traject_start |   38|    0.19005|       9502720|      65536|
+--------------+-----+----------+--------------+-----------+
|delay_collatz | 44.4|    0.22162|      11081057|      65536|
+--------------+-----+----------+--------------+-----------+
```

Terminal

```
--Performance Counter Report--
Total Time: 0.346551 seconds  (17327539 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|traject_start | 49.5|    0.17170|       8585216|      65536|
+--------------+-----+----------+--------------+-----------+
|delay_collatz | 35.7|    0.12373|       6186326|      65536|
+--------------+-----+----------+--------------+-----------+
```

uno *speed-up* di un ordine di grandezza, rispetto al calcolo software nell'esercitazione 09, risulta dai dati di prestazione in quel caso, con gli stessi livelli di ottimizzazione

Terminal

```
--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|
+--------------+-----+----------+--------------+-----------+
```

Terminal

```
--Performance Counter Report--
Total Time: 4.55965 seconds  (227982443 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|traject_start | 3.77|    0.17170|       8585216|      65536|
+--------------+-----+----------+--------------+-----------+
|delay_collatz | 95.1|    4.33682|     216841223|      65536|
+--------------+-----+----------+--------------+-----------+
```

test con accelerazione non bloccante

è lecito attendersi un ulteriore guadagno di prestazione dall'esecuzione non bloccante del calcolo nel componente hardware custom

dai Performance Counter Report che seguono, a confronto con i dati analoghi della realizzazione con tutto il calcolo in software, risulta uno speed-up 21x con ottimizzazione di default O1 e 16x con ottimizzazione O3; i corrispondenti valori dello speed-up con accelerazione bloccante sono 15x con O1 e 13x con O3

> N.B. lo speed-up è calcolato sui valori del tempo totale; i dati di sezione sono meno significativi con accelerazione non bloccante poiché le sequenze di esecuzione delle due sezioni sono parallele nel tempo

Terminal

```
--Performance Counter Report--
Total Time: 0.359145 seconds  (17957243 clock-cycles)
+--------------+----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+----+----------+--------------+-----------+
|traject_start  | 52.9|    0.19005|        9502720|      65536|
+--------------+----+----------+--------------+-----------+
|delay_collatz  | 86.5|    0.31065|       15532367|      65536|
+--------------+----+----------+--------------+-----------+
```

Terminal

```
--Performance Counter Report--
Total Time: 0.285762 seconds  (14288114 clock-cycles)
+--------------+----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+----+----------+--------------+-----------+
|traject_start  | 60.1|    0.17170|        8585216|      65536|
+--------------+----+----------+--------------+-----------+
|delay_collatz  | 89.4|    0.25561|       12780693|      65536|
+--------------+----+----------+--------------+-----------+
```


riferimenti

materiali utili per l'esperienza di laboratorio proposta:

- archivio con file sorgenti per la riproduzione del progetto
- *Avalon® Interface Specifications*, Ch. 1-3
  MNL-AVABUSREF, Intel Corp., 2017.05.08
- *Making Qsys Components* - For Quartus Prime 16.1
  Intel Corp. - FPGA University Program, November 2016
- *Introduction to the Qsys System Integration Tool* - For Quartus Prime 16.1,
  Intel Corp. - FPGA University Program, November 2016
- *Nios II Classic Software Developer's Handbook*, Ch. 7
  NII5V2, Altera Corp., 2015.05.14
- *Intel FPGA Monitor Program Tutorial for Nios II* - For Quartus Prime 16.1
  Intel Corp. - FPGA University Program, November 2016