FPGA implementation of a memory-mapped multicore coprocessor

Tutorial 12 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2019-20

Table of Contents

tutorial outline

this tutorial deals with:

- FPGA implementation of the first project idea proposed in lecture 12
  - multicore coprocessor design
  - coprocessor hardware interface
  - coprocessor register map
  - Nios II system with coprocessor and Performance Counter
  - software driver
- test and performance measurement with the Monitor Program
  - test with no status register read
  - test with status register read

project workflow

development main phases:

- VHDL description of the multicore coprocessor
- VHDL description of the multicore coprocessor with Avalon MM interface
- Qsys construction of a Nios II system with coprocessor and performance counter, system mapping to FPGA, and compilation
- production of the software driver and of TCL script for its generation in HAL
- production of the software application for testing and performance measurement, in two versions:

  sequential: execution with no read of the coprocessor status register

  status-tested: execution with read of the coprocessor status register
- compilation and execution of the application under the Monitor Program, for two variants of each version: one with defaut value of the optimization level, the other with level O3
- save of performance reports and project archiving

multicore coprocessor design

two-step production of the VHDL description of the multicore coprocessor:

- 1-core coprocessor with 64-bit input
- 2^n-core coprocessor with 2^n-bit status output

the respective sources delay_collatz.vhd and multicore_delay_collatz.vhd are available in folder vhdl of the attached archive, as well as in the VHDL/code/e12 folder of the reserved lab area

the 1-core coprocessor is derived from that of the previous lab tutorial, by a straightforward modification of the Gezel source and consequent variation of the VHDL output produced by the fdlvhd translator

the coprocessor is endowed with the core_select n-bit input, that encodes the core which the I/O operation is addressed to, while the done outputs of the individual cores are exposed as global status in a 2^n-bit parallel output port

the x0 and delay data ports of the individual cores are deployed on internal $64*2^n$-bit and $16*2^n$-bit parallel signals, respectively, wherein the decoding of core_select selects the relevant part for the I/O operation

empty folders delay_collatz, mc_delay_collatz, and mc_interface are meant to host compilation and simulation projects for the two mentioned sources and the next one; folders with the same names under tests provide respective input files for simulation

coprocessor hardware interface

an instance of the multicore coprocessor component is embedded in the Avalon memory-mapped interface described by multicore_delay_collatz_avalon_interface.vhd and accesses the following Avalon bus signals:

clock, resetn, read, write, chipselect, address, waitrequest, writedata, readdata

- address has an $(n+2)$-bit width and encodes the coprocessor register offset (see next page)
- the writedata, readdata signals have a 32-bit width each, for a single-cycle data transfer with the Nios II processor

the gathering of the 64-bit input for the coprocessor thus takes two bus cycles, therefore the interface must store the first-cycle data and later concatenate it with the second-cycle data; this leads to the classical two-process structure of the description:

- one for the first-cycle data register update,
- the other one for its concatenation with the next input data by the combinational network

on the other hand, the 32-bit output of a 16-bit data produced by a coprocessor core requires a zero-extension of the latter, that is done by the interface

consultation of multicore_delay_collatz_avalon_interface.vhd shows the relationships between the I/O signals of the computational component and the Avalon interface signals

coprocessor register map

the Qsys construction of a Nios II system with the coprocessor component, similar to that of the previous lab tutorial, assigns the coprocessor a base address and, starting at it, a memory area for its I/O registers

memory is byte-addressable and I/O register addresses are word-aligned, with 4-byte words, yet the programming model of the software driver of a component on the Avalon bus, by default, prescribes register identification by a word index, termed *register offset*, that coincides with the address input of its Avalon interface

the following register map also shows the coprocessor component signals determined by the corresponding register offsets, indexed by the value of core_select in parentheses, where $k = 2^n$ is the number of parallel cores, and with *legenda*:

ro: register offset
ao: memory address offset (with respect to the base address)

ro	**signal**	**ao**		**ro**	**signal**	**ao**
0	x0(0)[31..0]	0		2k	delay(0)	8k
1	x0(0)[63..32]	4			...	
	...			3k-1	delay(k-1)	12k-4
2(k-1)	x0(k-1)[31..0]	8(k-1)		3k	status	12k
2k-1	x0(k-1)[63..32]	8k-4				

Nios II system with coprocessor and Performance Counter

the subsequent development phases are similar to those of the previous lab tutorial:

- construction of the coprocessor Qsys component
- Nios II system construction with coprocessor and Performance Counter
- mapping to FPGA and compilation

the Qsys construction of the Nios II system goes quicker if performed as a modification of the Qsys system out of the previous lab tutorial, by removing the delay_collatz_avalon_interface component and adding an instance of the multicore_delay_collatz_avalon_interface component

beware: pay attention to save the modified system in the current project directory rather than in the project directory of the system to be modified

software driver

the TCL scripts for the generation of the software driver in the project BSP, provided in folder codesign/ip/multicore_delay_collatz_avalon_interface of the attached archive, are similar to those of the previous lab tutorial

the C sources of the software driver, provided in folder HAL under the same path, differ from those of the previous lab tutorial in the following aspects:

- definition of constant MDC_N_CORES = 32, the number of cores in the coprocessor
- for the start of a core computation on a trajectory of given start point the function mdc_start is available, that performs two bus write operations (because of the double word length of the start point), unlike the macro that performs only one bus write in the previous case
- besides the function delay, to read the computation result out of a given core, function status is available, to read the coprocessor status register

test and performance measurement programs

the test and performance measurement programs provided in folders codesign/amp* of the attached archive compute the delay for 2M initial points, starting with X_BASE = 1128784494896128

N.B.: X_BASE+14 is the class record of class 1746

in both versions of the test, the program assigns core j the delay computation for the initial points x0 in the congruence class j = x0 mod MDC_N_CORES, thus for 2M/32 = 64K trajectories (on the average in the second version); the difference between the versions in codesign/amp_s* and those in codesign/amp_t* is as follows:

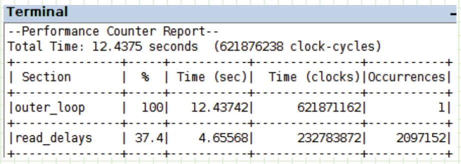
- in the former case, termed *sequential*, 64K iterations of a loop are executed, where each of 32 core reads is followed by the core restart, with no status register read (the processor thus waits after any request to read a not yet available result)
- in the latter case, termed *status-tested*, the processor reads the status register and processes its content bit by bit, while requesting results from only those cores which are done with their computation

project creation parameters for the Monitor Program are summarized in the attached file MulticoreMonitorNotes.txt

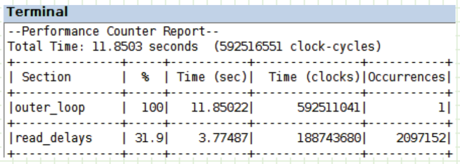
performance measurement outcomes

compilation, loading on the FPGA and execution of program sequential_multicore_delay_collatz_timing.c, in the two projects codesign/amp_s and codesign/amp_s_o3, produces the Performance Counter Reports in the figure

- as in the previous lab tutorial, the more significant reduction of the execution time of the delay read operations in the second variant may be explained by the function *inlining* under compilation O3

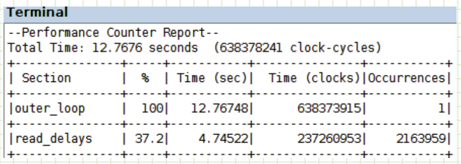


```
Terminal
--Performance Counter Report--
Total Time: 12.4375 seconds  (621876238 clock-cycles)
+---------------+-----+----------+----------------+----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+----------------+----------+
|outer_loop     |  100|  12.43742|       621871162|         1|
+---------------+-----+----------+----------------+----------+
|read_delays    | 37.4|   4.65568|       232783872|   2097152|
+---------------+-----+----------+----------------+----------+
```

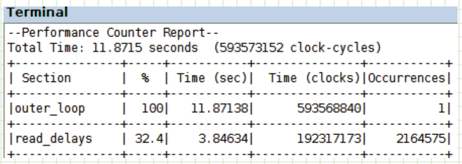


```
Terminal
--Performance Counter Report--
Total Time: 11.8503 seconds  (592516551 clock-cycles)
+---------------+-----+----------+----------------+----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+----------------+----------+
|outer_loop     |  100|  11.85022|       592511041|         1|
+---------------+-----+----------+----------------+----------+
|read_delays    | 31.9|   3.77487|       188743680|   2097152|
+---------------+-----+----------+----------------+----------+
```

the next Performance Counter Reports come out of the execution of program statustest_multicore_delay_collatz_timing, in the two projects codesign/amp_t and codesign/amp_t_o3



```
Terminal
--Performance Counter Report--
Total Time: 12.7676 seconds  (638378241 clock-cycles)
+---------------+-----+----------+----------------+----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+----------------+----------+
|outer_loop     |  100|  12.76748|       638373915|         1|
+---------------+-----+----------+----------------+----------+
|read_delays    | 37.2|   4.74522|       237260953|   2163959|
+---------------+-----+----------+----------------+----------+
```



```
Terminal
--Performance Counter Report--
Total Time: 11.8715 seconds  (593573152 clock-cycles)
+---------------+-----+----------+----------------+----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+----------+----------------+----------+
|outer_loop     |  100|  11.87138|       593568840|         1|
+---------------+-----+----------+----------------+----------+
|read_delays    | 32.4|   3.84634|       192317173|   2164575|
+---------------+-----+----------+----------------+----------+
```

references

useful materials for the proposed lab experience:

- archive with source files for project reproduction
- Intel Corp. documents referred to in the previous lab tutorial
- M. Zwoliński, Ch. 4, Sect. 4.2.3, 4.5.2