Design and implementation of a memory-mapped multicore coprocessor

Lecture 12 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2018-19

Table of Contents

lecture topics

outline:

- codesign evolution for computations on Collatz trajectories
- evolution of the delay computation
  - performance enhancement
  - domain extension
- project development
  - hardware design of the multicore coprocessor
  - 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

design space

possible evolutions of the codesign implemented in the previous lab tutorial may be conceived along two orthogonal development directions:

- performance enhancement
- functional extensions

an example of orthogonal combination of both directions is given by the following objectives for a first project that here is going to be dealt with:

- a definite performance enhancement may be delivered by parallel replication of the hardware unit in charge of the delay computation
- on the other hand, a minimal functional extension of definite interest is the widening of the input width, to make the delay computation applicable to a wider domain of trajectories

more significant functional extensions may be identified from consideration of functions, defined over Collatz trajectories, other than the delay but whose computation requires trajectory generation anyway

> the challenge here would be the design of multifunction hardware units as well as of a more complex HW/SW interface, apt to selection of the functions of interest and to the related data transfer

codesign evolution of the delay computation

a first design alternative to be considered about the replication of the hardware unit for the delay computation is:

- multiple instances of the computational component on the Avalon bus, or
- construction of a more complex component, embedding multiple copies of the individual computational component

the latter option is preferable in view of possible further extensions that would require access by the different instances to shared data, e.g. defined as configuration parameters

bus transactions for this purpose are avoided in this way, moreover a control hierarchy is implemented whereby the coprocessor is assigned local control functions

other design decisions relate to the number of parallel instances of the computational component, henceforth termed *cores*, and the size of the coprocessor I/O data

- taking it into account that the Nios II processor may transfer at most 32 bits in a single bus transaction, and that it may be useful to have a status register in the coprocessor, with one bit per core, it proves convenient to set the limit on the number of cores at 32
- the data available in the website on the 3x+1 problem by Eric Roosendaal, e.g. in the Class Records Table, indicate a 64-bit minimum size for input data of actual interest, while a 16-bit data size suffices for the delay output

structure of a multicore coprocessor

the 64-bit extension of the single core input is easily obtained by a straightforward modification of the Gezel source from the previous lab tutorial, with the same correction to the VHDL output of the `fdlvhd` translator

the structural description of the multicore coprocessor in VHDL is eased by the iterative construct **for** <identifier> **in** <range> **generate**, whose usage is exemplified in Zwoliński, 4.5.2, that in our case allows the generation of the $2^n = 32$ core instances, with $n = 5$

the multicore coprocessor then ought to have circuits for the correct dispatching of I/O data between the interface and a core selected by the processor:

- a multiplexer with n-bit selection input and 16-bit data ports for the core delay output
- a demultiplexer with n-bit selection input and 64-bit data ports for the core initial data input, as well as a similar one but with 1-bit data ports for the core `start` signal input

description in VHDL of multiplexing is simple if the core outputs are chained in a 2^nx16-bit vector, as it is enough to use a selection operator on the vector

the more complex description in VHDL of demultiplexing is feasible by means of a logical shift operator, as it is exemplified for a generic decoder in Zwoliński, 4.2.3

hardware interface constraints

signal exchange at the multicore coprocessor I/O ports is to be adapted to the available signals of the Avalon-MM interface, taking a few constraints on these into account, such as:

- the data transfer signals, writedata and readdata, must have the same width
- signal address identifies an I/O register in the memory address space assigned to the coprocessor, starting from 0 and with word-addressing by default

since the Nios II processor may transfer at most 32 bits in a single bus transaction, it is convened that this be the coprocessor word width at the Avalon interface, that is to say, the width of the writedata and readdata signals

this entails that the delay output is to be zero-extended, whereas the trajectory input data is to be acquired in two bus cycles

the register address space of the coprocessor is thus the interval $[0, 3x2^n]$, taking one address for the status register into account, hence address is $(n+2)$-bit wide

a suitable management of register addresses allows a quick identification of the core (or status register) from the value of the address input, for example:
decode(address[n,1]) if address[n+1] = 0
decode(address[n-1,0]) if address[n+1,n] = 10
status register if address[n+1,n] = 11

project workflow

development main phases:

- VHDL description and simulation of the multicore coprocessor
- VHDL description and simulation 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

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

> the 1-core coprocessor is obtained in a similar way as that of the previous lab tutorial, with straightforward modification of the Gezel source and similar correction of the 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

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

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 coprocess register address (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 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_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 in the congruence class j 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 MonitorNotes.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: 10.5684 seconds  (528420451 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|outer_loop    |  100|   10.56834|     528416826|          1|
+--------------+-----+----------+--------------+-----------+
|read_delays   | 41.7|    4.40402|     220200960|    2097152|
+--------------+-----+----------+--------------+-----------+
```

```
Terminal
--Performance Counter Report--
Total Time: 9.39399 seconds  (469699747 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|outer_loop    |  100|    9.39393|     469696577|          1|
+--------------+-----+----------+--------------+-----------+
|read_delays   | 40.6|    3.81682|     190840832|    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: 12.0294 seconds  (601467952 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|outer_loop    |  100|   12.02929|     601464327|          1|
+--------------+-----+----------+--------------+-----------+
|read_delays   | 37.4|    4.49599|     224799527|    2164772|
+--------------+-----+----------+--------------+-----------+
```

```
Terminal
--Performance Counter Report--
Total Time: 11.1297 seconds  (556485961 clock-cycles)
+--------------+-----+----------+--------------+-----------+
| Section      |  %  | Time (sec)|  Time (clocks)|Occurrences|
+--------------+-----+----------+--------------+-----------+
|outer_loop    |  100|   11.12966|     556482791|          1|
+--------------+-----+----------+--------------+-----------+
|read_delays   |   35|    3.88987|     194493464|    2164781|
+--------------+-----+----------+--------------+-----------+
```


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
- Zwoliński, Ch. 4, Sect. 4.2.3, 4.5.2