FPGA implementation of a memory-mapped coprocessor

Tutorial 11 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

tutorial outline

this tutorial deals with a hardware acceleration case study

- design and FPGA implementation of a computation accelerator of the Collatz delay
  - problem statement and design decisions
  - coprocessor hardware interface
  - coprocessor as a Qsys component
  - Nios II system with coprocessor and Performance Counter
  - software driver
- test and performance measurement with the Monitor Program
  - test with blocking acceleration
  - test with nonblocking acceleration

example: design decisions for a hardware acceleration case study

the previous lab tutorial presented a software implementation of the delay computation of a Collatz trajectory with given start point

hardware implementations of the same function were the subject of previous lab experiences

e.g. the third lab experience produces a VHDL description of it

the performance measurements carried out on the software implementation show that it consumes almost all of the program execution time

problem: accelerate the program execution by using the hardware implementation of the aforementioned function

a first alternative to evaluate: to integrate the hardware function as a custom instruction or as a memory-mapped coprocessor?

the second option seems better, for at least two reasons:

- the first option is *blocking*
- the data transfer size in each interaction is very small

other design decisions depend on this first decision, as follows

Avalon interface and programming model for the case study

the VHDL description of the circuit which computes the function is to be embedded into a component equipped with Avalon interfaces for the Clock, Reset, and Avalon MM Slave signals, so as to receive the initial data by a write operation and to return the result by a reply to a read operation

> *multicycle* data transfers are possible thanks to the Avalon signal `waitrequest`, set by the slave to defer the response to a read or write request by an arbitrary number of cycles

addressing of the coprocessor: since the (initial data) write and (final result) read operations take place at different times and have the same data size, a single address suffices

> for the sake of simplicity, it is convenient to use the 32-bit Avalon signals `writedata`, `readdata` in the hardware interface for this address, with internal conversion to 16-bit for the corresponding internal I/O ports of the circuit which computes the function

software driver: two macros and a function may be defined for the bus access software interface: `DC_RESET(d)`, `DC_START(d,x0)`, `unsigned int delay(d)`, where `d` is the address assigned to the coprocessor

> these project ideas will be developed with a few options according to the following workflow

project workflow

development main phases:

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

coprocessor hardware interface

two VHDL sources implement the memory-mapped coprocessor:

- delay_collatz.vhd, modified version of the output by the fdlvhd translation of the Gezel source presented in the second lecture, according to the third lab experience
- delay_collatz_interface.vhd, which contains an instance of the computational component and accesses the following Avalon bus signals: clock, resetn, read, write, chipselect, waitrequest, writedata, readdata

both files are available in the vhdl folder of the attached archive, which is also located in the Nios II folder of the reserved lab area

the folder also contains std_logic_arithext.vhd, which is needed to compile the computational component, and delay_collatz_codesign.vhd, which is explained next

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

coprocessor as a Qsys component (1)

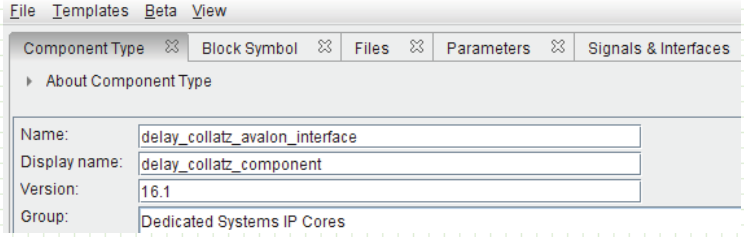
folder codesign in the attached archive is preset to host the project development

after having copied the *.vhd files from folder vhdl into it, the Qsys custom component construction goes much like in the tutorial seen in lab tutorial 10, with due differences for the present case

after creation of project delay_collatz_codesign, with top-level entity having the same name, the construction of the custom component delay_collatz_interface may proceed

in particular, the Conduit Avalon interface is not needed by the present component, since it makes no use of peripherals outside the FPGA

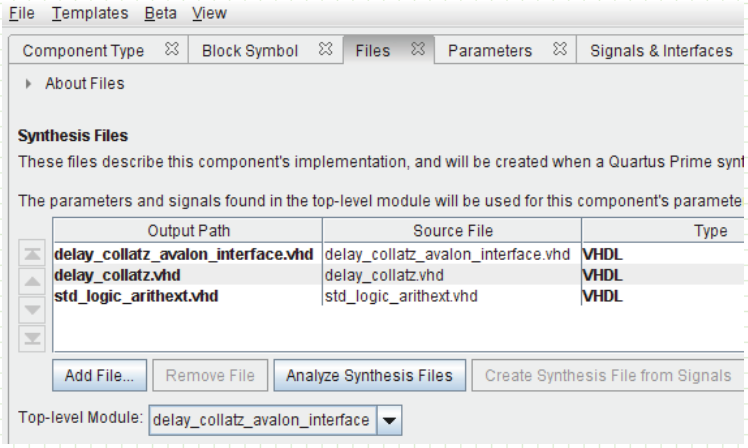
the new component type definition is shown in the figure



coprocessor as a Qsys component (2)

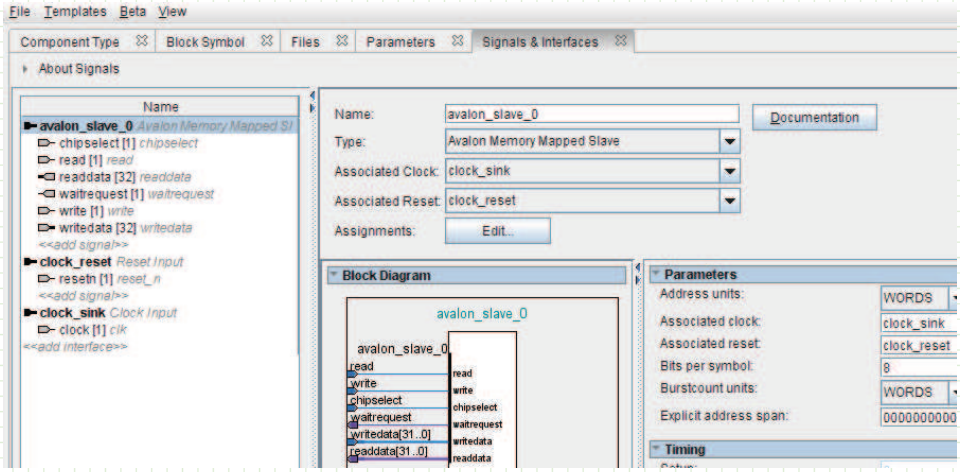
the next step is the assignment of VHDL files that describe the component and their analysis, as shown in the figure

N.B. for this project, it is not necessary to copy files for simulation

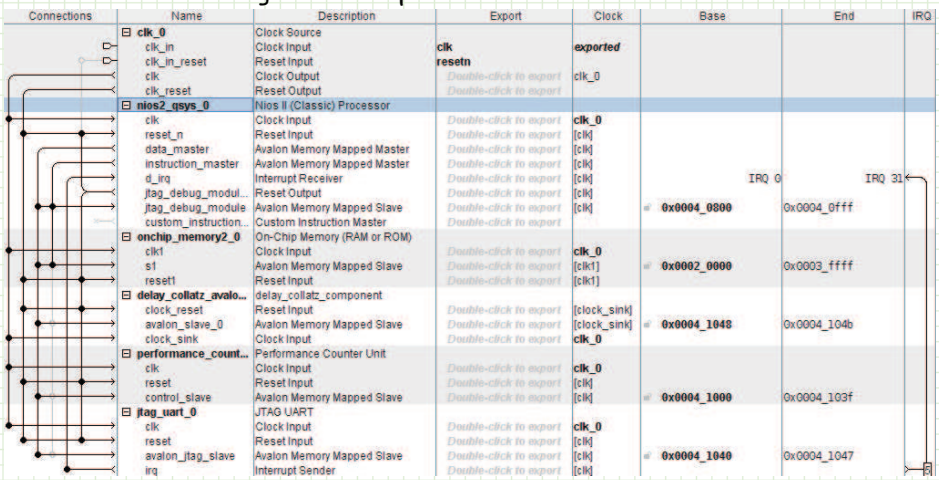


coprocessor as a Qsys component (3)

finally, the new component definition ends with the definition of its Avalon interfaces and placement of its signals under the appropriate interfaces, as shown in the figure



Nios II system with coprocessor and Performance Counter



Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk_0	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	resetn				
	clk	Clock Output	Double-click to export	clk_0			
	clk_reset	Reset Output	Double-click to export				
	nios2_qsys_0	Nios II (Classic) Processor					
	clk	Clock Input	Double-click to export	clk_0			
	reset_n	Reset Input	Double-click to export	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	d_irq	Interrupt Receiver	Double-click to export	[clk]			IRQ 0
	jtag_debug_modul...	Reset Output	Double-click to export	[clk]			IRQ 31
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_0800	0x0004_0fff	
	custom_instruction...	Custom Instruction Master	Double-click to export				
	onchip_memory2_0	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to export	clk_0			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0002_0000	0x0003_ffff	
	reset1	Reset Input	Double-click to export	[clk1]			
	delay_collatz_avalo...	delay_collatz_component					
	clock_reset	Reset Input	Double-click to export	[clock_sink]			
	avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock_sink]	0x0004_1048	0x0004_104b	
	clock_sink	Clock Input	Double-click to export	clk_0			
	performance_count...	Performance Counter Unit					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_1000	0x0004_103f	
	jtag_uart_0	JTAG UART					
	clk	Clock Input	Double-click to export	clk_0			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_1040	0x0004_1047	
	irq	Interrupt Sender	Double-click to export	[clk]			

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 to FPGA and compilation

for the construction of the Nios II system shown in the previous figures it may be useful to consult the Qsys introduction tutorial

with a few differences, such as: memory size is 128 KB in the present case, all base addresses are assigned by th system, etc.

the final steps to map the system to the FPGA are as follows:

in Qsys:

- save the system with name embedded_system by File > Save As...
- generate the VHDL code for it by Generate > Generate HDL...

exit Qsys, then in Quartus:

- assign the project files embedded_system.qip (in embedded_system/synthesis) and delay_collatz_timing.sdc
- import assignments from file DE1_SoC.qsf in folder de1soc of the attached archive
- File > Save Project
- compile delay_collatz_codesign.vhd

software driver

folder script in the attached archive contains two TCL scripts for the generation of the software driver in the BSP for the project

- the two scripts differ for a single command, present in one of them, that prescribes optimization level O3 rather than the default level O1

these two scripts are to be copied in folder codesign/ip/delay_collatz_avalon_interface

- in the same folder, respectively under HAL/inc and HAL/src, copy is to be made of the C sources delay_collatz_avalon_interface.h and delay_collatz_avalon_interface.c of the software driver, that are available in folder src of the attached archive

the TCL scripts were written by analogy with the TCL script for the software driver of the Performance Counter, available in the Quartus Prime Lite 16.1 distribution under path
$SOPC_KIT_NIOS2/../ip/altera/sopc_builder_ip/altera_avalon_performance_counter

- similarly, the C sources of the software driver were written by (more limited) analogy with the C sources of the software driver of the same IP Core, in folder HAL under the aforementioned path

the motivation for this, perhaps unorthodox, way of producing the software driver lies in the twofold fact that

- the Avalon interface of the custom component does not fit into any of the HAL generic device model classes defined in Chapter 7 of the Nios II Classic Software Developer's Handbook
- neither does the Performance Counter Unit IP Core fit therein ...

together with a somewhat reasonable level of operational analogy between the two components

- skimming through Chapter 7 of the handbook is recommended nonetheless, to get a better understanding of the software driver structure and contents

test and performance measurement programs (1)

folder src in the attached archive contains the subject programs, which are to be copied in the provided folders for the creation of test and performance measurement projects under the Monitor Program, as follows:

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

project creation parameters are summarized in the attached file MonitorNotes.txt

- the DE1-SoC needs to be powered-up and connected to the PC, to program the FPGA at the end of each project creation

main differences between the source of lab tutorial 10 and the present sequential version:

- #include and #define directives relating to the custom component
- replacement of the input from the switches device with a constant
- replacement of the body of function delay_collatz with two instructions from the software driver of the custom component

test and performance measurement programs (2)

the *pipelined* version of the program exhibits much stronger differences with respect to the program of lab tutorial 10:

- the interaction with the custom hardware is made *nonblocking* by replacing the `delay_collatz` function call with an *inlining* of its body, yet where the software computation of the next trajectory start point is placed in between the two inlined instructions, respectively to start the hardware computation and to read its result

the synchronization mechanism is very simple, thanks to properties of the custom component and of the `waitrequest` signal of the Avalon MM protocol:

- for trajectories faster than the software computation, the custom component keeps the result in its internal register while waiting the read command
- for trajectories slower than the software computation, the read command is kept waiting by the Avalon interface by means of the `waitrequest` signal

test with blocking acceleration

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

- the remarkable reduction of the execution time of section `delay_collatz` in the second variant may be explained by the function *inlining* under compilation 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|
+--------------+-----+----------+--------------+-----------+
```

a speed-up by an order of magnitude, w.r.t. the software computation in lab tutorial 10, results from the performance data in that case, with the same optimization levels

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 with nonblocking acceleration

it is sensible to expect a further performance gain out of the nonblocking execution of the computation by the custom hardware

the comparison of the following Performance Counter Reports with the corresponding data for the implementation with all computation done in software, yields a 21x speed-up with default optimization O1 and a 16x speed-up with optimization O3; the corresponding speed-up values with blocking acceleration are 15x with O1 and 13x with O3

> N.B the speed-up is computed on the total time; section data are less significant with nonblocking acceleration because the execution threads of the two sections overlap in time

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


references

useful materials for the proposed lab experience:

archive with source files for project reproduction

Avalon® Interface Specifications, Ch. 1-3
MNL-AVABUSREF, Intel Corp., 2018.09.26

Making Qsys Components - For Quartus Prime 16.1
Intel Corp. - FPGA University Program, November 2016

Performance Counter Unit Core, Ch. 36 in: *Embedded Peripherals IP User Guide*, Intel Corp. - UG-01085 | 2018.09.24

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