xmodz user manual

Title	xmodz (IP core collection)
Author	Nikolaos Kavvadias (C) 2012, 2013
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	03 June 2013
Version	1.0.0
Rev. history	
v1.0.0	03-06-2013 First public release.

1. Introduction

The `xmodz` IP collection provides fast hardware implementations for the `x mod z` computation on integers. The collection comprises of two distinct IP modules, `modk` for modulo by a fixed integer constant and `modv` for modulo by an integer variable.

The algorithm used for implementing `x mod z` is based on modulo reduction where at each stage, the magnitude of `x` is reduced, but the residue remains the same.

Modulo reduction is widely used in cryptographically-secure systems, for fast pseudo-random number generation and is suitable for RNS (Residue Number System) applications.

This implementation has been inspired by this published work:

J. T. Butler and T. Sasao, "Fast hardware computation of x mod z," Proceedings of the 18th Reconfigurable Architectures Workshop (RAW 2011), May 16-17, 2011, Anchorage, Alaska, USA.

The following sections provide details on the contents of the delivered IP cores, which include all necessary materials such as source files and scripts for RTL simulation and logic synthesis.

Reference documentation for XMODZ can be found in the `/doc` subdirectory in plain text, HTML and PDF form.

2. Functional description

XMODZ is implemented as fully-parameterized RTL VHDL using a clean process-based style with combinational-only and sequential-only processes. The following table summarizes the parameters (as VHDL generics) that are supported by each design.

Design	Parameter	Description
modk, modv	MODE	Choose between a fully-pipelined design (`REGISTERED`) and a combinational one (`COMBINATIONAL`).
modk, modv	N	Data bitwidth.
modk	K	Constant value for `z` in `x mod z`.

The registered designs have a throughput of one clock cycle, which means that a new result can be computed every single clock cycle. The same applies for the fully combinational designs. In terms of latency, the `modv` design has a latency of `N+2` while the `modk` design has a latency of `N-M`, where M is the number of bits required for representing the constant value `K`. Both registered and combinational designs provide a fully-synchronous interface by registering their outputs.

The following table provides an overview of the throughput and latency metrics for all supported modes of the `modk` and `modv` designs.

Design	Mode	Latency	Throughput
modv	COMBINATIONAL	1	1
modv	REGISTERED	N+2	1
modk	COMBINATIONAL	1	1
modk	REGISTERED	N-M+2	1

The interface block diagrams for both designs are shown below. Each core uses a single external clock source, connected to signal `CLK`. It can be asynchronously reset with the active high signal `RESET`. Signal `START` activates the core. Data inputs `X` and `Z` (the latter only in the `modv` case) are the numerator and denominator involved in the modulus operation. Data output `Y` is the outcome of this computation. `DONE` signifies the end of the current computation. `READY` indicates that the core can accept new input.

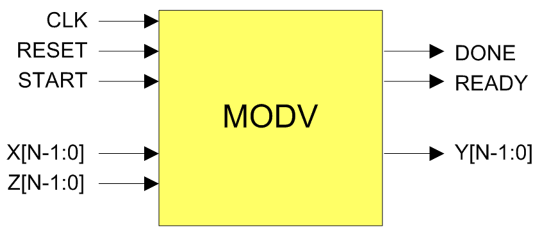

Figure 1: `modv` I/O interface.

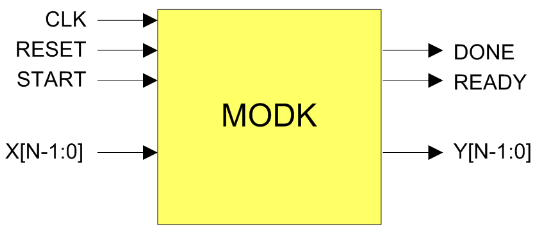

Figure 2: `modk` I/O interface.

3. File listing

The XMODZ distribution includes the following files.

/xmodz	Top-level directory
/bench/vhdl	Benchmarks VHDL directory
modk_ieee_tb.vhd modv_ieee_tb.vhd std_logic_textio.vhd	Self-checking testbench for the `modk` IP. Self-checking testbench for the `modv` IP. Draft version of the `std_logic_textio` package.
/common/fgmp	Source code directory for the free GMP library
fgmp.c fgmp.h Makefile notes	ANSI C implementation of the free GMP library API. Header file for the free GMP library. Makefile for building the libfgmp.a static library. Documentation for the free GMP library.
/doc	Documentation directory
AUTHORS LICENSE xmodz-modk-if.png xmodz-modv-if.png xmodz-pb.pdf README README.html README.pdf VERSION	List of authors. End-user license agreement for using `xmodz`. PNG image illustrating the `modk` IP I/O interface. PNG image illustrating the `modv` IP I/O interface. Product brief (brochure) for the XMODZ IP cores. This file. HTML version of README. PDF version of README. Current version of the XMODZ IP cores.
/rtl/vhdl	RTL source code directory for the IP core
modk_ieee.vhd modv_ieee.vhd	RTL VHDL design file for the `modk` IP core. RTL VHDL design file for the `modv` IP core.
/sim/rtl_sim	RTL simulation files directory
/sim/rtl_sim/bin	RTL simulation scripts directory
modk.do modk.mk modv.do modv.mk	`do` script for simulating `modk` with Modelsim. GNU Makefile for simulating `modk` with GHDL. `do` script for simulating `modv` with Modelsim. GNU Makefile for simulating `modv` with GHDL.
/sim/rtl_sim/out	Dumps and other useful output from RTL simulation

modk_results_all.txt modv_results_all.txt	Output from multiple RTL simulations of modk. Output from multiple RTL simulations of modv.
/sim/rtl_sim/run	Files for running RTL simulations
ghdl.sh mti.sh run-sim-mod[k|v].sh	Bash script for running a single GHDL simulation. Bash script for running a single Modelsim simulation. Bash script for running multiple simulations of modk/modv with either GHDL or Modelsim.
/sim/rtl_sim/src	Various source files for running RTL simulations
chg-generics-modk.pl	Perl script for producing a version of modk as modk.vhd with updated generics.
chg-generics-modk.pl	Perl script for producing a version of modv as modv.vhd with updated generics.
/sw	Software utilities
Makefile modk.c modv.c	GNU Makefile for building modk.exe/modv.exe. Reference I/O data generator for modk. Reference I/O data generator for modv.
/syn/xise	Synthesis files for use with Xilinx ISE
/syn/xise/bin	Synthesis scripts directory
xst.mk	Standard Makefile for command-line usage of ISE.
/syn/rtl_sim/run	Files for running synthesis
run-xst-modk.sh run-xst-modv.sh	Bash shell script for synthesizing modk with ISE. Bash shell script for synthesizing modv with ISE.

4. Simulation

The XMODZ IP cores distribution supports both GHDL and Mentor Modelsim simulation.

4.1. GHDL

For running the GHDL simulation, change directory to the `/sim/rtl_sim/run` subdirectory:

```
$ cd $XMODZ_HOME/sim/rtl_sim/run
```

assuming `XMODZ_HOME` is the directory where the top-level `/xmodz` is found. Then, the corresponding shell script is executed, e.g. for the `modv` design:

```
$ ./run-sim-modv.sh ghdl
```

The simulation produces two files, a VCD (waveform) dump named `modv.vcd` and the diagnostic text file `modk_results.txt` which are automatically copied to the `/sim/rtl_sim/out` subdirectory. The generated result files from multiple VHDL simulations are concatenated into a master diagnostics file named `modv_results_all.txt`.

The `modk` design can be simulated in the same way, if you replace `modk` for `modv` in the instructions above.

4.2. Modelsim

For running the Modelsim simulation, the corresponding shell script is executed from the `/sim/rtl_sim/run` subdirectory:

```
$ ./run-sim-modv.sh mti
```

As in the GHDL case, the VCD dump and the diagnostic text file are produced.

Again, the `modk` design can be simulated in the same way, if you replace `modk` for `modv` in the instructions above.

5. Synthesis

The XMODZ IP cores distribution includes scripts for logic synthesis automation supporting Xilinx ISE. The corresponding synthesis script can be edited in order to specify the following for adapting to the user's setup:

- `XDIR`: the path to the `/bin` subdirectory of the Xilinx ISE/XST installation where the `xst.exe` executable is placed
- `arch`: specific FPGA architecture (device family) to be used for synthesis
- `part`: specific FPGA part (device) to be used for synthesis

5.1. Running the synthesis script

For running the Xilinx ISE synthesis tool, change directory to the `/syn/xise/run` subdirectory:

```
$ cd $XMODZ_HOME/syn/xise/run
```

and execute the corresponding script (for synthesizing `modv`):

```
$ ./run-xst-modv.sh
```

The synthesis procedure invokes several Xilinx ISE command-line tools for logic synthesis as described in the corresponding Makefile, found in the the `/syn/xise/bin` subdirectory.

Typically, this process includes the following:

- Generation of the `*.xst` synthesis script file.
- Generation of the `*.ngc` gate-level netlist file in NGC format.
- Building the corresponding `*.ngd` file.
- Performing mapping using `map` which generates the corresponding `*.ncd` file.
- Place-and-routing using `par` which updates the corresponding `*.ncd` file.
- Tracing critical paths using `trce` for reoptimizing the `*.ncd` file.
- Bitstream generation (`*.bit`) using `bitgen`, however with unused pins.

Finally, the `modv.bit` bitstream file is produced.
The same process can be applied for synthesizing the `modk` design as well.

6. Reference software application

The reference C applications for `modk` and `modv` are available in the `/sw` subdirectory.
For the case of `modk` the C application is built with the help of the associated GNU Makefile by the corresponding simulation script as follows:

```
make K=${cnst} NUMBITS=${bw} modk.exe
```

where `cnst` is an integer value for the constant denominator `K` and `bw` is the number of bits for the input and output ports of the design.
For the case of `modv` the C application is built as follows:

```
make NUMBITS=${bw} modv.exe
```

The produced executables can be used for generating reference I/O data as follows:

```
./modk.exe >& modk_data.txt
```

Again, the corresponding simulation script (`run-sim-modk.sh` or `run-sim-modv.sh`) automatically takes care of generating the executables and running them to produce reference I/O data, so this process needs not be run manually.

7. Prerequisities

- Standard UNIX-based tools (tested with gcc-4.6.2 on MinGW/x86).
  - make
  - bash (shell)
  - perl

  For this reason, MinGW (http://www.mingw.org) or Cygwin (http://sources.redhat.com/cygwin) are suggested, since POSIX emulation environments of sufficient completeness.
- GHDL simulator (http://ghdl.free.fr) or Modelsim (http://www.model.com). The latest GHDL distribution (0.29.1, Windows version) also installs GTKwave on Windows.
- Xilinx ISE (free ISE webpack is available from the Xilinx website: http://www.xilinx.com)

8. Contact

You may contact me at:

Nikolaos Kavvadias <nikos@nkavvadias.com>

http://www.nkavvadias.com
http://www.perfeda.gr
Perfeda Technologies headquarters
35100 Lamia, Fthiotis
Greece