Versatile library

A collection of frequently used modules with synthesis support

Brought to You By ORSoC / OpenCores

Legal Notices and Disclaimers

Copyright Notice

General Disclaimer

The Publisher has strived to be as accurate and complete as possible in the creation of this ebook, notwithstanding the fact that he does not warrant or represent at any time that the contents within are accurate due to the rapidly changing nature of information.

The Publisher will not be responsible for any losses or damages of any kind incurred by the reader whether directly or indirectly arising from the use of the information found in this ebook.

This ebook is not intended for use as a source of legal, business, accounting, financial, or medical advice. All readers are advised to seek services of competent professionals in the legal, business, accounting, finance, and medical fields.

No guarantees of any kind are made. Reader assumes responsibility for use of the information contained herein. The Publisher reserves the right to make changes without notice. The Publisher assumes no responsibility or liability whatsoever on the behalf of the reader of this report.

Distribution Rights

The Publisher grants you the following rights for re-distribution of this ebook.

[YES] Can be given away.
[YES] Can be packaged.
[YES] Can be offered as a bonus.
[NO] Can be edited completely and your name put on it.
[YES] Can be used as web content.
[NO] Can be broken down into smaller articles.
[NO] Can be added to an e-course or auto-responder as content.
[NO] Can be submitted to article directories (even YOURS) IF at least half is rewritten!
[NO] Can be added to paid membership sites.
[NO] Can be added to an ebook/PDF as content.
[NO] Can be offered through auction sites.
[NO] Can sell Resale Rights.
[NO] Can sell Master Resale Rights.
[NO] Can sell Private Label Rights.

Table of Contents

Description

A Verilog HDL library with frequently used functions.

Care have been taken to fully support synthesis of all modules. Different versions exist for optimal synthesis support. Currently ACTEL and ALTERA are supported

The following types of functions are included

1. Clock and reset
   Global buffers, PLL and sync reset
2. Registers
   with clock enable, async set and reset, ...
3. Counters
   Binary, Gray, LFSR
4. Memories
   RAM, dual port RAM, ...
5. Wishbone
   Wishbone system-on-chip bus realted logic

Most modules uses parameter for per instance uniqeness.

Different version of the library exist. All included module names are identical in all library versions. Retargeting a design by changing into different library.

1. versatile_library.v
   Target independent library
2. versatile_library_actel.v
   ACTEL version with synthesis constraints for synplify / Libero
3. versatile_library_altera.v
   ALTERA version with synthesis constraints for QuartusII

Note: The target independent could also be used with target specific constarints simply by adding the following switch during simulation and or synthesis

+define+ACTEL or
+define+ALTERA

Clock and reset

vl_gbuf

A global buffer to be used to ensure global routing resources for high fanout signals such as clock and reset networks.

Signal	Direction	Comment
i	Input	
o	Output	Global signal

ACTEL specific functions

By specifying define SIM_GBUF at simulation time a simulation model will be used for RTL simualtion. Simulation will not need ACTEL primitives during simulation. Apply to all insatnces in design.

vl_sync_reset

To ensure proper startup behaviour each clock network should have a dedicated synchronous reset signal.

Signal	Direction	Comment
rst_n_i	Input	Async active low reset input, normally driver from external reset and/or PLL lock signal
rst_o	Input	Global synchronous reset signal, active high
clk		Clock source

vl_pll

Most FPGA have PLL functions built-in. This function is one of few which are not possible to describe in HDL and have synthesis support. A functional model can be described.

This library contains the following:

1. vl_pll in versatile_library.v
   a functional model for simulation only
2. vl_pll in versatile_library_target.v
   a wrapper for one or more actual PLL
   a functional model for simulation only
   the later used if define SIM_PLL is applied at simulation time

Simulation model

Parameters

Parameter	Default value	Comment
index	0	Only used for synthesis. Parameter used in PLL wrapper to give a unique name to actual PLL. Parameter is concateneted with pll to give actual name. Example: index=0 => instance name pll0

Parameter	Default value	Comment
number_of_clocks	3	Defines number of generated clocks
period_time_0 .. period_time_n	20	Defines input period time of input clock source, in ns
lock_delay	2000	Time to lock PLL, used in simulation only. Time in ps

Top level signals

Signal	Direction	Comment
clk_i	Input	Clock source
rst_n_i	Input	Async active low reset input used for synchronous reset generation
lock	Output	Signal indicating PLL is in locked state and that clock outputs are valid
clk_o [0:number_of_clocks-1]	Output	Vector with generated clocks

ACTEL specific functions

By specifying define SIM_PLL when cmpiling in simulator a functional model is used for PLL.

Appendix A have a detailed usage example of PLLs when used with ACTEL ProASIC3 as target.

Registers

dff

Simple D-type flip-flop. Parameter to set width

Parameters

Parameter	Default value	Comment
width	1	Defines vector size
reset_value	0	Defines value applied at reset

Module interface signals

Signal	Direction	Comment
d	Input	D input
q	Output	Latched signal
clk	Input	Clock source
rst	Input	Active high asynchronous reset

dff_ce

Simple D-type flip-flop with clock enable. Parameter to set width

Parameters

Parameter	Default value	Comment
width	1	Defines vector size
reset_value	0	Defines value applied at reset

Module interface signals

Signal	Direction	Comment
d	Input	D input
ce	Input	Clock enable
q	Output	Latched signal
clk	Input	Clock source
rst	Input	Active high asynchronous reset

dff_sr

D-type flip-flop with asynchronous set and reset, also called SR flip-flop.

Parameters

Parameter	Default value	Comment
reset_value	0	Defines value applied at reset

Module interface signals

Signal	Direction	Comment
data	Input	D input
aclr	Input	Asynchronous clear
aset	Input	Asynchronous set
q	Output	Latched signal
clk	Input	Clock source

Not all FPGA devices support this type of flip-flop.

ALTERA specific implementation

A generated model with synthesis support is included in the ALTERA version of the library.

Counters

Counters can be implemented in different ways. The actual use case should dictate what type is best suited to use. Below is a table with pros and cons for various types.

1. Binary

   The normal implementation.
   Typical usage include

   1. address generator

2. Linear Feedback Shift Register, LFSR

   Extremely area efficient, uses no carry chains. Very useful where high performance is important. The count sequence is one short as compared to binary counters, that is with a vector length of n the count sequence is 2^n – 1.
   Typical usage includes

   1. timer with timeout assertion

3. Gray encoded

   For every state change in a gray encoded counter there is only one bit changing. Implementation uses a normal binary counter with an encoder on the output. That means that both gray and binary outputs can be used
   Typical uses include

   1. address generator for asynchronous FIFOs

4. Shift register

   A shift register encoded in a one hot fashion can be used as a counter. Implementation uses only flip-flop and no logic resources. Since the number of flip flop used grows rapidly with increased number of state this type is most suited for small counters.

   1. Delay counters where multiple assertions could be used for different delays

Note: All counters except shift register based is derived from OpenCores project versatile_counter

http://opencores.org/project,versatile_counter

Binary counters

The following binary counters are implemented. Other types can easily be generated from the Versatile Counter project

Module	clr	set	cke	rew	l1	comment
cnt_bin_ce			X			Binary counter with clock enable
cnt_bin_ce_clear	X		X			Binary counter with clock enable and syn clear
cnt_bin_ce_clear_set_rew	X	X	X	X		Binary counter with clock enable and syn clear and set
cnt_bin_ce_rew_l1			X	X	X	Binary up/down counter with clock enable and level indicator

Parameters

Parameter	Default value	Comment
length	4	Defines vector length
clear_value	0	Defines value applied at clear
set_value	1	Defines value applied at set
wrap_value	0	Defines maximum counter value before wrap to init state
level1_value	1	Level where output level1 shall be set

Module interface signals

Signal	Direction	Comment
clear	Input	Synchronous clear
set	Output	Synchronous set
cke	Input	Clock enable
rew	Input	Rewind input counts down
q	Output	Counter state output
level1	Output	Level1 indicator
clk	Input	Clock source
rst	Input	Active high asynchronous reset

LFSR counters

The following LFSR counters are implemented. All LFSR counters have wrap function enabled and will wrap dependent on wrap value. Other types can easily be generated from the Versatile Counter project

Module	clr	set	cke	rew	zq	l1	comment
cnt_lfsr_zq					X		LFSR counter with zero indicator
cnt_lfsr_ce_zq			X		X		LFSR counter with clock enable and zero indicator
cnt_lfsr_ce_rew_l1			X	X		X	LFSR up/down counter with clock enable and level indicator

Parameters

Parameter	Default value	Comment
length	4	Defines vector length
clear_value	0	Defines value applied at clear
set_value	1	Defines value applied at set
wrap_value	0	Defines maximum counter value before wrap to init state
level1_value	1	Level where output level1 shall be set

Module interface signals

Signal	Direction	Comment
clear	Input	Synchronous clear
set	Output	Synchronous set
cke	Input	Clock enable
rew	Input	Rewind input counts down
q	Output	Counter state output
level1	Output	Level1 indicator
clk	Input	Clock source
rst	Input	Active high asynchronous reset

LFSR counters generate a pseudo random state sequence. To calculate wrap and level values use application from OpenCores project Versatile Counter. Usage is described in documentation for project. An example of usage can also be found in Appendix B in this document.

GRAY counters

The following LFSR counters are implemented. All LFSR counters have wrap function enabled and will wrap dependent on wrap value. Other types can easily be generated from the Versatile Counter project

Module	clr	set	cke	rew	zq	bin	comment
cnt_gray_ce			X				GRAY counter with clock enable
cnt_gray_ce_bin			X			X	GRAY counter with clock enable and zero indicator

Parameters

Parameter	Default value	Comment
length	4	Defines vector length
clear_value	0	Defines value applied at clear
set_value	1	Defines value applied at set
wrap_value	0	Defines maximum counter value before wrap to init state

Module interface signals

Signal	Direction	Comment
clear	Input	Synchronous clear
set	Output	Synchronous set
cke	Input	Clock enable
q	Output	GRAY encoded output
q_bin	Output	Binary encoded output
clk	Input	Clock source
rst	Input	Active high asynchronous reset

SHREG counters

The following LFSR counters are implemented. All LFSR counters have wrap function enabled and will wrap dependent on wrap value. Other types can easily be generated from the Versatile Counter project

Module	clr	set	cke	rew	comment
cnt_shreg_ce			X		SHREG counter with clock enable
cnt_shreg_ce_clear	X				SHREG counter with clock enable and clear

Parameters

Parameter	Default value	Comment
length	4	Defines vector length

Module interface signals

Signal	Direction	Comment
clear	Input	Synchronous clear
cke	Input	Clock enable
q	Output	GRAY encoded output
clk	Input	Clock source
rst	Input	Active high asynchronous reset

Appendix A
vl_pll usage example with ACTEL ProASIC3 target

Requirements:

1. clk_i1 external clock, 66.666666667 MHz
   shall generate
   clk0 : 66.666666667 MHz
   clk1 : 25.000 MHz
2. clk_i2 external 125.000 MHz
   shall generate
   clk3 : 125.000 MHz
3. each clock domain should have a global synchronized reset signal

Appendix B
LFSR counter usage example

Assume the following scenario:

In an SDRAM controller running at 133 MHz we want to have a complete refresh in 64 ms. For a page size of 8192 words the timeout for the counter should be

133 MHz / 128 kHz = 1039

Recommended Resources

ORSoC – http://www.orsoc.se

ORSoC is a fabless FPGA/ASIC design & manufacturing services company, providing RTL to FPGA/ASIC design services and silicon fabrication service. **ORSoC** are specialists building complex system based on the OpenRISC processor platform.

Open Source IP – http://www.opencores.org

Your number one source for open source IP and other FPGA/ASIC related information.