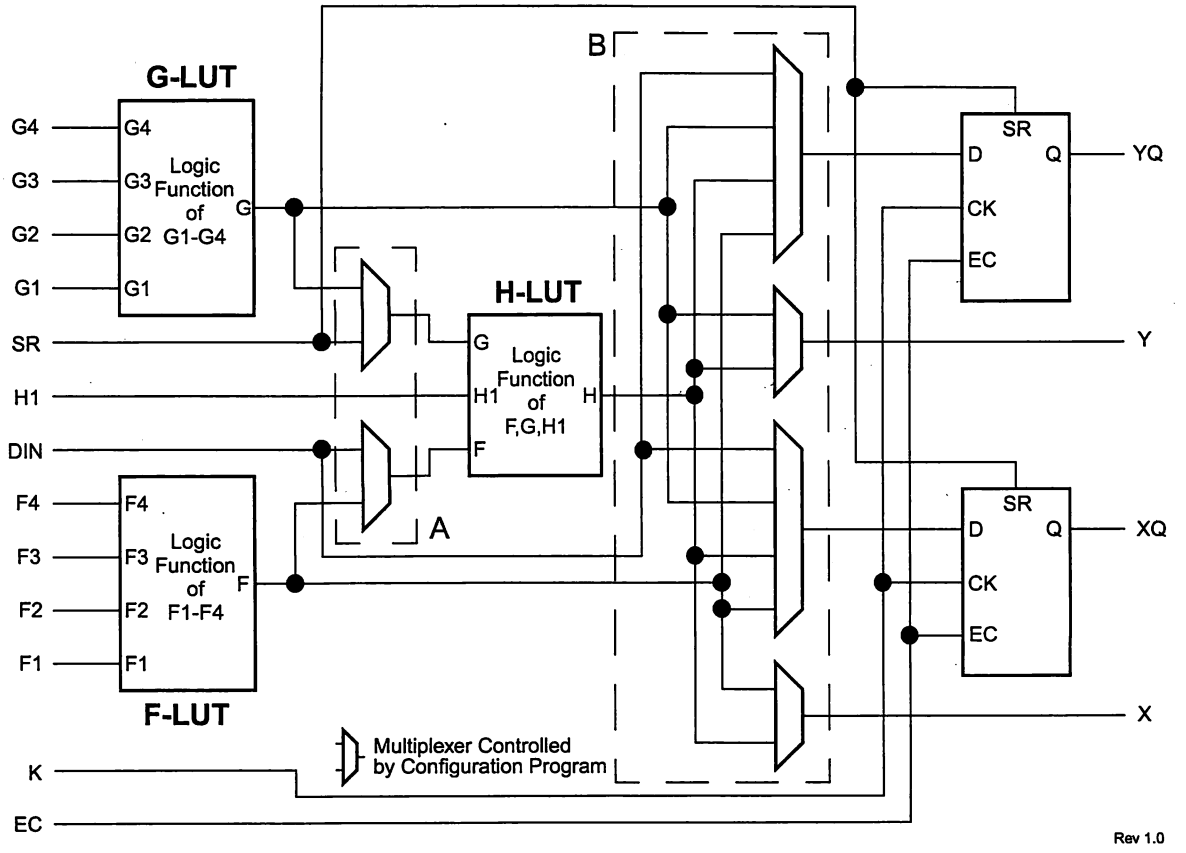
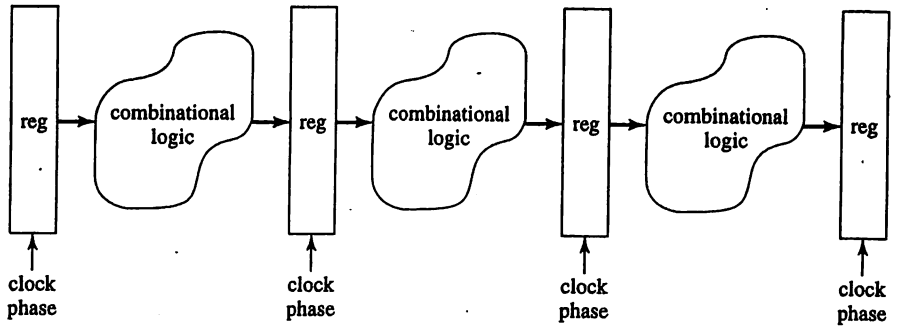


Figure 2: Spartan Simplified CLB Logic Diagram (some features not shown)



General Synchronous Design Paradigm.

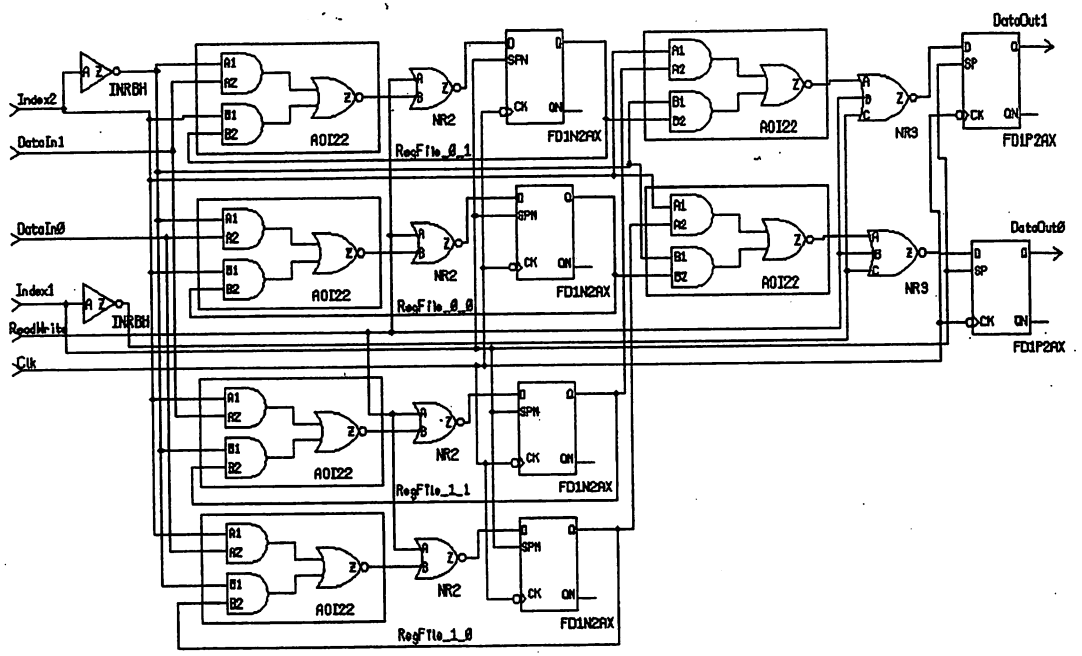
Table 14.1 *Xilinx Memory Primitives.*

Name	*Remarks*	*Augmented Ports*
RAM16X1		WE
RAM16X1D	dual port, synch write, asynch read	DPRA3,2,1,0,WCLK,WE
RAM16X1S	dual port, synch write, asynch read	WCLK
RAM32X1		A4
RAM32X1S	dual port, synch write, asynch read	A4, WCLK
ROM16X1	read only	
ROM32X1	read only	A4

```
module TPRX(clk, Din, raddr, waddr, we, Dout); //wrapper for synthesis to Xilinx LUT's
 input clk, we;
 input [3:0] raddr, waddr;
 input [7:0] Din;
 output [7:0] Dout;
 wire  [7:0] Dout, Din;
 // invoke 8 of the two port ram primitives from table 14 -1:
 RAM16X1D u0(.WCLK(clk),  .SPO(Din[0]), .WE(we), .DPO(Dout[0],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),    .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]), .DPRA2(raddr[2]), .DPRA3(raddr[3]) );
 RAM16X1D u1(.WCLK(clk),  .SPO(Din[1]), .WE(we), .DPO(Dout[1],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
       .DPRA0(raddr[0]), .DPRA1(raddr[1]), .DPRA2(raddr[2]), .DPRA3(raddr[3])  );
 RAM16X1D u2(.WCLK(clk),  .SPO(Din[2]), .WE(we),  .DPO(Dout[2],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
       .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]),  .DPRA3(raddr[3]) );
 RAM16X1D u3(.WCLK(clk),  .SPO(Din[3]), .WE(we),  .DPO(Dout[3],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]), .DPRA3(raddr[3])  );
 RAM16X1D u4(.WCLK(clk),  .SPO(Din[4]), .WE(we),  .DPO(Dout[4],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]), .DPRA3(raddr[3]) );
 RAM16X1D u5(.WCLK(clk),  .SPO(Din[5]), .WE(we),  .DPO(Dout[5],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]), .DPRA3(raddr[3])  );
 RAM16X1D u6(.WCLK(clk),  .SPO(Din[6]), .WE(we),  .DPO(Dout[6],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]), .DPRA3(raddr[3])  );
 RAM16X1D u7(.WCLK(clk),  .SPO(Din[7]), .WE(we),  .DPO(Dout[7],
          .A0(waddr[0]),     .A1(waddr[1]),     .A2(waddr[2]),  .A3(waddr[3]),
        .DPRA0(raddr[0]), .DPRA1(raddr[1]),.DPRA2(raddr[2]), .DPRA3(raddr[3])  );
endmodule
```

Specification #2: RAM Wrapper for Xilinx.

reg [1:0] mem [1:0]



A 2-by-2 register file.

Intellectual Property

Integrated Synthesis Environment

Related Products

- Silicon Devices
- Design Tools
- Intellectual Property
- Boards & Kits
- Engineering Services
- Technology

Key Documentation

- IP Release Notes Guide
- Product Discontinuation and other Customer Notices
- Support

Xilinx Intellectual Property (IP) are key building blocks of Xilinx Targeted Design Platforms. An extensive catalog of base-level cores is available to address the general needs of FPGA designers, as well as robust domain- and market-specific cores to address requirements found in DSP, Embedded, and Connectivity designs. Use Xilinx IP to free up precious time and resources you would normally spend developing standard functions and focus instead on the aspects of your design that differentiate your product from the competition.

Search for IP cores

Search

Advanced search for IP

Browse for Intellectual Property

- By Function
- By Type
- By Vendor

Platforms and Solutions

IP Licensing and Evaluation

- IP Licensing
- IP License Installation Help
- IP Evaluation

Xilinx IP Life Cycle Management

- Life Cycle State Definitions
- ISE CORE Generator™ Removed Cores
- EDK Removed Cores

New IP Cores

- 3GPP LTE PUCCH Receiver
- 7 Series Integrated Block for PCI Express (PCIe)
- Video Direct Memory Access (axi_vdma)
- Linear Algebra Toolkit
- Object Segmentation
- Triple-Rate SDI
- 7 Series FPGAs Transceiver Wizards
- XADC Wizard

Most Requested IP Products

- PCI-Express®
- PCI and PCI-X Technology
- Spartan-6 FPGA Integrated Endpoint Block for PCI Express (PCIe)
- Virtex-6 Integrated Block for PCI Express (PCIe)
- Forward Error Correction Solutions
- EDK Supported IP
- Xilinx Ethernet Solutions
- MicroBlaze™
- External Memory Solutions
- Architecture Wizards

Quick Links

- AXI4 at Xilinx
- IP Evaluation
- License IP
- Licensing Solutions
- What's New in ISE Design Suite 13
- IP Update 13.1
- Contact Local Sales
- Product Support & Documentation
- Sign up for IP Design Advisories

System Requirements

- Supported Operating Systems
- Memory Recommendations

Videos and Webcasts

- All ISE Design Suite Videos
- All ISE Design Suite Webcasts

Advanced Search Results

inurl:xilinx.com Search Within Results

Start a new search

Results **1** - **11** of about **11**.

Basic Logic

Product Name	Vendor	Type	SignOnce	Supported Devices
LD-based Parallel Latch	Xilinx, Inc.	LogiCORE		Virtex-II Pro, Virtex-II, 3E, Spartan-3, Sparta
Binary Counter	Xilinx, Inc.	LogiCORE		Virtex-6, Virtex-5, Virt 3A, Spartan-3A DSP,
RAM-based Shift Register	Xilinx, Inc.	LogiCORE		Virtex-6, Virtex-5, Virt 3A, Spartan-3A DSP,
8b/10b Decoder	Xilinx, Inc.	LogiCORE		Virtex-4 LX, Virtex-4 S Pro, Virtex-II, Virtex-E II, Spartan-IIE
8b/10b Encoder	Xilinx, Inc.	LogiCORE		Virtex-4 FX, Virtex-4 L Pro, Virtex-II, Virtex-E II, Spartan-IIE
Comparator	Xilinx, Inc.	LogiCORE		Virtex-5 LX, Virtex-4 F Virtex-II Pro, Virtex-II,
Binary Decoder	Xilinx, Inc.	LogiCORE		Virtex-II, Virtex-E, Virt Spartan-II, Spartan-IIE
BUFE-based Multiplexer Slice	Xilinx, Inc.	LogiCORE		Virtex-II Pro, Virtex-II, 3E, Spartan-3, Sparta
Linear Feedback Shift Register (LFSR)	Xilinx, Inc.	LogiCORE		Virtex-II Pro, Virtex-II, Spartan-IIE
Floating Point Comparator (DFPCOMP)	Digital Core Design	AllianceCORE	X	Virtex-5 FXT, Virtex-5 SXT, Virtex-4 FX, Virt Spartan-3A, Spartan-3
FD-based Parallel Register	Xilinx, Inc.	LogiCORE		Virtex-II Pro, Virtex-II, 3E, Spartan-3, Sparta

Binary Counter v11.0

Introduction

The Xilinx® LogiCORE™ IP Binary Counter core provides LUT and single XtremeDSP™ slice counter implementations. The Binary Counter is used to create up counters, down counters, and up/down counters with outputs of up to 256-bits wide. Support is provided for one threshold signal that can be programmed to become active when the counter reaches a user defined count. The upper limit of the count is user programmable and the counter's increment value can be user defined. When the counter reaches terminal count or the *count to value*, the next count will be zero.

Features

- Drop-in module for Virtex®-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/AN/3A DSP/XA FPGAs
- Backwards compatible with version 9.1
- Generates up, down, and up/down counters
- Supports fabric implementation counters ranging from 1 to 256 bits wide
- Supports DSP48 implementation counters ranging from 1 to 36, or 48 bits wide (varies with device family)
- Pipelining added for maximal speed performance
- Predictive detection used for threshold and terminal count detection
- Optional synchronous set and synchronous init capability for legacy fabric implementations
- Optional user programmable threshold outputs
- Optional clock enable and synchronous clear
- Counter increment value can be user defined
- User-programmable count limit
- For use with Xilinx CORE Generator™ and Xilinx System Generator™ v11.1 or later

Pinout

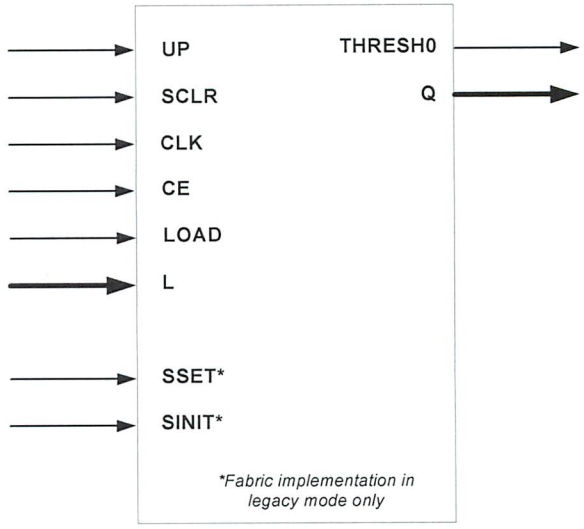


Figure 1: **Core Symbol**

Signal names for the core symbol are shown in Figure 1 and described in Table 1. Note that Figure 1 shows the SSET and SINIT pins which appear only on legacy fabric implementations. The fabric legacy mode has been provided to preserve backwards compatibility but allow performance to be improved. This is where **Implement using** = Fabric, **Latency** = 1 and **FB_Latency** = 0.

Table 1: **Core Signal Pinout**

Signal	Direction	Description
CLK	Input	Clock. Rising edge clock signal
UP	Input	Controls the count direction on an up/down counter. Counts up when high, down when low
CE	Input	Active high Clock Enable
SCLR	Input	Synchronous Clear: forces the output to a low state when driven high
THRESH0	Output	User-programmable active high threshold signal
Q[N:0]	Output	Output
L[N:0]	Input	Load data port
LOAD	Input	Load control signal
SSET[1]	Input	Synchronous Set: forces the output to a high state when driven high
SINIT[1]	Input	Synchronous Initialize: forces the outputs to a user defined state when driven high

1. Available only when in fabric legacy mode: **Implement using** = Fabric, **Latency** = 1, and **Feedback Latency** = 0

CORE Generator Graphical User Interface Parameters

The CORE generator GUI parameters for this module are described below:

- **Implement using**: Sets the implementation type to Fabric or DSP48. The default value is Fabric.
- **Output Width**: Specifies the width of the counter. The valid range is 1 to 256. Default value is 16.
- **Restrict Count**: When this parameter is true the counter will only count up (or down) to the value specified in the **Final Count Value** parameter. When it is false the counter will count up to the maximum value that can be represented using the specified output width.The default is for no count restriction. This option is mutually exclusive with the up/down counter option and with synchronous set controls.
- **Final Count Value**: When **Restrict Count** = true, this parameter specifies the hex representation of the upper limit of the counter. The valid range for values is 1 to $2^{\text{Output Width}} - 2$. The default value is 1.
- **Increment Value**: Specifies in hex the increment value of the counter. When **Restrict Count** is false, the valid range is 1 to $2^{\text{Output Width}} - 1$. When **Restrict Count** is true the valid settings for **Increment Value** are governed by the equation:

  Final Count Value / **Increment Value** = Integer

  for up counters, and:

  $2^{\text{Output Width}}$ - **Final Count Value** / **Increment Value** = Integer

  for down counters. The default value is 1.
- **Count Mode**: This parameter specifies whether the counter will count up, down, or will have its direction specified on the UP pin (up/down). The valid values are UP, DOWN, and UPDOWN. The default value is UP.
- **Sync Threshold Output**: When this parameter equals true, the THRESH0 combinatorial output will be generated. The default is to not generate a THRESH0 output.

1553 Encoder/Decoder

April 2005 | Reference Design RD1021

Introduction

The MIL-STD-1553 is a low-speed serial bus used in avionics systems. This reference design implements Manchester II encoding and decoding required by the 1553 along with synchronization pattern insertion and identification, data serialization and de-serialization and parity checking and insertion functions.

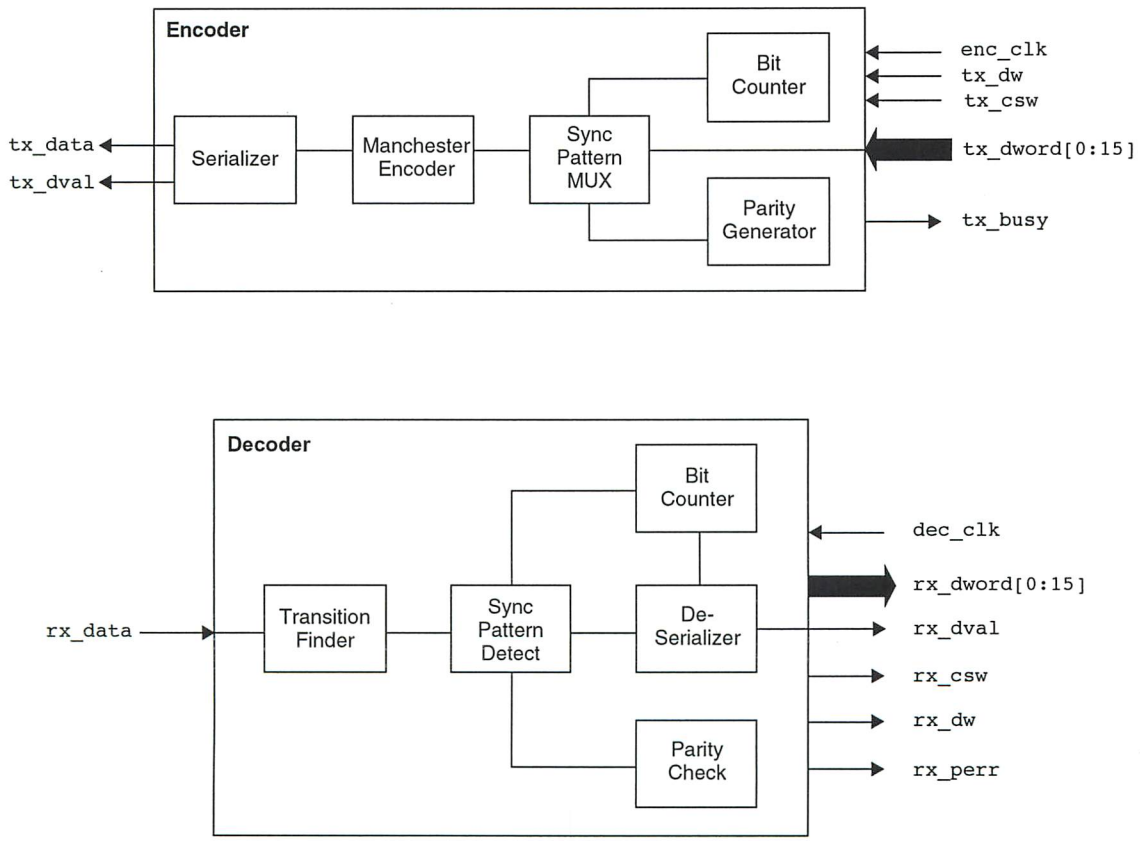
Features

- MIL-STD-1553 Compatible
- 1 Mbps Data Rate
- Sync Pattern Identification and Insertion
- Manchester II Encoding/Decoding
- Parity Checking and Insertion
- Serialization and De-serialization

Functional Description

The following figure shows a block diagram of the different functions implemented in this 1553 Encoder/Decoder along with the input/output signals.

Figure 1. 1553 Encoder/Decoder Block Diagram



Encoder Operation

The encoder requires a single clock with a frequency (2 MHz) of twice the desired data rate (1 Mbps) for enc_clk. The encoder cycle begins with either tx_csw or tx_dw pulse along with the command-status or data word to be transmitted. Then the encoder asserts tx_busy until it transmits this word serially through all the encoder functions and then de-asserts tx_busy to accept the next word.

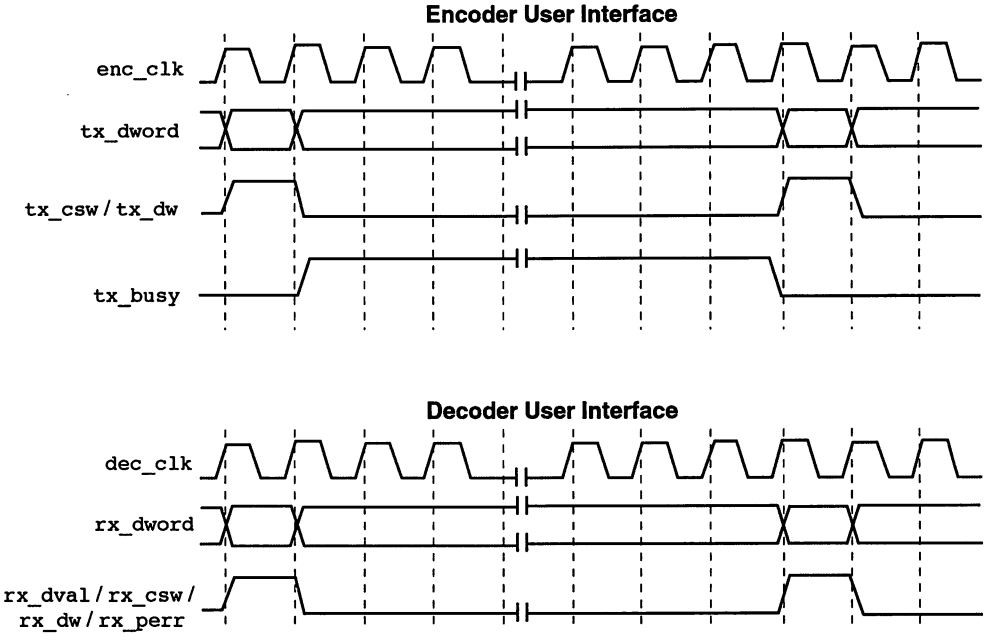
Decoder Operation

The decoder requires a single clock with a frequency (8 MHz) of 8 times the desired data rate (1 Mbps) for dec_clk. The decoder is free running and continuously hunting for the synchronization pattern on the serial input. When a valid synchronization pattern is detected, it identifies the boundary of the word and determines it as either a command-status word or a data word. Then the serial bits are passed through shift registers, and a parallel word is presented to the user interface along with the type of word and when it is valid. Also, the word is checked for parity errors.

Pin Descriptions

Port Name	Active State	I/O	Signal Description
Decoder			
dec_clk	—	In	Decoder clock of 8 MHz
rst_n	Low	In	Asynchronous reset
rx_data	—	In	Serial data Input
rx_dword[0:15]	—	Out	Received output data word to user
rx_dval	High	Out	Data valid indication for rx_dword
rx_csw	High	Out	Indicates rx_dword has command or status word
rx_dw	High	Out	Indicates rx_dword has data word
rx_perr	High	Out	Indicates parity error in rx_dword
Encoder			
enc_clk	—	In	Encoder clock of 2 MHz
rst_n	Low	In	Asynchronous reset
tx_dword[0:15]	—	In	Data word from user for transmission
tx_csw	High	In	Indicates tx_dword has command or status word
tx_dw	High	In	Indicates tx_dword has data word
tx_data	—	Out	Serial data output
tx_dval	High	Out	Data valid indication for tx_data
tx_busy	High	Out	Indicates Encoder is not ready to accept the next word

Timing Diagrams



Implementation Results

The design software used for this implementation is Lattice ispLEVER® version 4.2 and the Synplicity Synplify synthesis tool. The device utilization and performance summary for LatticeEC™ and LatticeXP™ devices (-4 speed grade) is shown below.

Device	Size	Reported Frequency
Decoder		
LFEC20E-4	53 SLICEs	8 MHz
LFXP10E-4	53 SLICEs	8 MHz
Encoder		
LFEC20E-4	39 SLICEs	2 MHz
LFXP10E-4	33 SLICEs	2 MHz

File List

The files provided in this LatticeEC reference design package are:

1. /1553_enc_dec/docs/1553_enc_dec.doc	Design document
2. /1553_enc_dec/docs/readme.txt	Read me file
3. /1553_enc_dec/source/decoder_1553.v	Decoder source verilog file
4. /1553_enc_dec/source/encoder_1553.v	Encoder source verilog file
5. /1553_enc_dec/par/EC/decoder_1553.prf	Decoder Constraint file for place and route
6. /1553_enc_dec/par/EC/encoder_1553.prf	Encoder Constraint file for place and route
7. /1553_enc_dec/par/EC/decoder_1553.syn	Decoder Project file for place and route
8. /1553_enc_dec/par/EC/encoder_1553.syn	Encoder Project file for place and route
9. /1553_enc_dec/simulation/EC/scripts/runsim_1553.do	Scripts for RTL simulation
10. /1553_enc_dec/synthesis/EC/synplify/decoder_1553.prj	Decoder Project file for synthesis using Synplify
11. /1553_enc_dec/synthesis/EC/synplify/encoder_1553.prj	Encoder Project file for synthesis using Synplify
12. /1553_enc_dec/synthesis/EC/synplify/decoder_1553.sdc	Decoder Constraint file for synthesis using Synplify
13. /1553_enc_dec/synthesis/EC/synplify/encoder_1553.sdc	Encoder Constraint file for synthesis using Synplify
14. /1553_enc_dec/testbench/test_1553.v	Testbench for simulation

The files provided in this LatticeXP reference design package are:

1. /1553_enc_dec/docs/1553_enc_dec.doc	Design document
2. /1553_enc_dec/docs/readme.txt	Read me file
3. /1553_enc_dec/source/decoder_1553.v	Decoder source verilog file
4. /1553_enc_dec/source/encoder_1553.v	Encoder source verilog file
5. /1553_enc_dec/par/xp/decoder_1553.prf	Decoder Constraint file for place and route
6. /1553_enc_dec/par/xp/encoder_1553.prf	Encoder Constraint file for place and route
7. /1553_enc_dec/par/xp/decoder_1553.syn	Decoder Project file for place and route
8. /1553_enc_dec/par/xp/encoder_1553.syn	Encoder Project file for place and route
9. /1553_enc_dec/simulation/xp/scripts/runsim_1553.do	Scripts for RTL simulation
10. /1553_enc_dec/synthesis/xp/synplify/decoder_1553.prj	Decoder Project file for synthesis using Synplify
11. /1553_enc_dec/synthesis/xp/synplify/encoder_1553.prj	Encoder Project file for synthesis using Synplify
12. /1553_enc_dec/synthesis/xp/synplify/decoder_1553.sdc	Decoder Constraint file for synthesis using Synplify
13. /1553_enc_dec/synthesis/xp/synplify/encoder_1553.sdc	Encoder Constraint file for synthesis using Synplify
14. /1553_enc_dec/testbench/test_1553.v	Testbench for simulation

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-408-826-6002 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

art of silicon

IP Cores

IP Cores

At Art of Silicon we have drawn on our experience to create a portfolio of multimedia IP.

All of our IP is created in line with the Reuse Methodology Manual, and delivered fully verified and documented with a reference model and testsuite.

ispLeverCORE Connection

Art of Silicon is a Lattice ispLeverCore Connection partner

IP cores

All of our IP Cores are available as Verilog or VHDL RTL ready for ASIC synthesis, or as an EDIF Netlist targetting Lattice, Xilinx or Altera FPGAs.

We can interface to any system bus through one of our bus interface blocks.

If you don't see what you require here, contact us about becoming a development partner for our future IP cores.

Contact us for pricing or further information.

JPEG cores

JPEG Decoder

1 colour component decoder datasheet

3 colour component decoder datasheet

JPEG Encoder

1 colour component encoder datasheet

3 colour component encoder datasheet

Coming Soon

Contact us for more details of our future products.

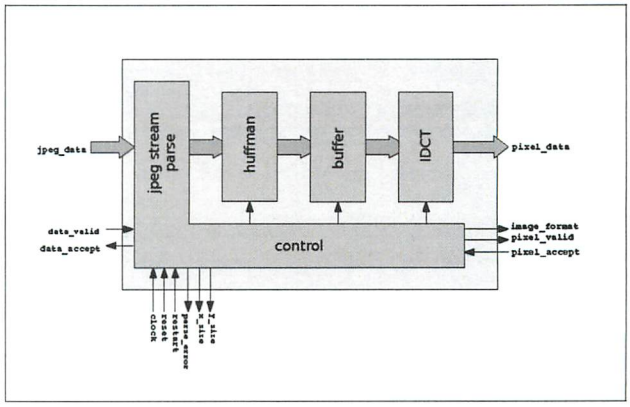
MPEG2 Cores

JPEG2000 Cores

Email: **all.enquiries@artofsilicon.com**

art of silicon

Colour JPEG Decoder Datasheet



Features

- Supports baseline JPEG decode as per ISO/IEC 10918-1
- Decodes one sample per clock cycle
- Decodes colour and greyscale images
- Self configuring by parsing JPEG header
- Supports image sizes up to 64k x 64k
- 8 bits per pixel
- Simple valid-accept interfaces for easy integration
- Available optimised for both ASIC and FPGA

Description

The Art of Silicon Colour JPEG decoder supports high speed baseline progressive JPEG decode. It is self configuring by reading header information from an incoming JPEG header. Any errors in the JPEG stream are detected and flagged.

All internal state is clocked from a single clock input. A synchronous reset is provided. Clock and reset senses are configurable.

This block is available as optimized FPGA bitstream or verilog source code, targeted at either ASIC or FPGA. A full testbench and test suite is included with the design.

Implementation Results

Family	Frequency (MHz)	Slices	Block Rams	DSP Blocks
Lattice ECP2	>85	3183	14	2
Lattice SC	>105	3678	14	-
Lattice XP	>45	3746	14	-

Deliverables

- Documentation
- EDIF Netlist or RTL
- Instantiation Template
- Constraints
- Testbench
- Reference C Model

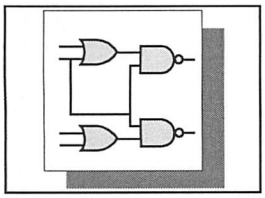
Art of Silicon IP

This is one member of a family of Intellectual Property available from Art of Silicon. Each block is designed with simple interfaces for ease of integration into your designs. We are also happy to customise our IP to your requirements. Telephone and email support is available for all our IP.

Contact Art of Silicon for pricing and ordering information.

Art of Silicon Ltd
Grosvenor Court
Westfield Park
Bristol
UK
BS6 6LT

email: all.enquiries@artofsilicon.com



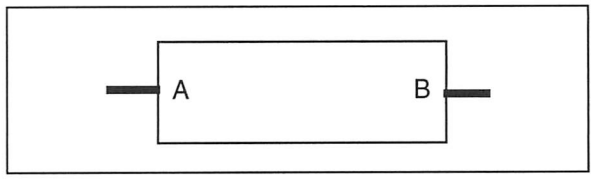
DW01_decode

Decoder

Last Revised: Release DWF_0206

Features and Benefits

- Parameterized word length
- Inferable using a function call



Description

DW01_decode decodes an address on input port `A` to a single bitline on output port `B`. The selected bitline on port `B` is active high.

Table 1: Pin Description

Pin Name	Width	Direction	Function
A	*width*	Input	Binary input data
B	2^{width}	Output	Decoded output data

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Word length of input A is *width*. Word length of output B is 2^{width}

Table 3: Synthesis Implementations

Implementation Name	Function	License Feature Required
str	Synthesis model	DesignWare-Foundation

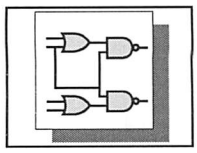


Table 4: Simulation Models

Model	Function
DW01.DW01_DECODE_CFG_SIM	Design unit name for VHDL simulation
dw/dw01/src/DW01_decode_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW01_decode.v	Verilog simulation model source code

Table 5: Truth Table (width = 3)

A(2:0)	B(7)	B(6)	B(5)	B(4)	B(3)	B(2)	B(1)	B(0)
000	0	0	0	0	0	0	0	1
001	0	0	0	0	0	0	1	0
010	0	0	0	0	0	1	0	0
011	0	0	0	0	1	0	0	0
100	0	0	0	1	0	0	0	0
101	0	0	1	0	0	0	0	0
110	0	1	0	0	0	0	0	0
111	1	0	0	0	0	0	0	0

Related Topics

- Logic – Combinational Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Function Inferencing - VHDL

```
library IEEE,DW01;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DW01.DW01_components.all;

entity DW01_decode_func is
  generic(func_width : integer := 5);
  port( func_A:     in  std_logic_vector(func_width-1 downto 0);
        B_func_TC:  out std_logic_vector(2**func_width-1 downto 0);
        B_func_UNS: out std_logic_vector(2**func_width-1 downto 0);
        B_func:     out std_logic_vector(2**func_width-1 downto 0));
end DW01_decode_func;

architecture func of DW01_decode_func is
begin

  B_func_TC  <= std_logic_vector(DWF_decode (signed (func_A)));
  B_func_UNS <= std_logic_vector(DWF_decode (unsigned (func_A)));
  B_func     <= DWF_decode (func_A);
end func;
```

HDL Usage Through Function Inferencing - Verilog

```
module DW01_decode_func (func_A,B_func);
  parameter func_width = 4;

  // Passes the width to the decode function
  parameter width = func_width;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW01_decode_function.inc"

  input [func_width-1:0] func_A;

  output [(1 << func_width)-1:0] B_func;

  assign B_func = DWF_decode(func_A);

endmodule
```

HDL Usage Through Component Instantiation - Verilog

Because Verilog does not support an exponentiation operator, you must explicitly set your input and output port widths. Check Table 1 on page 1 for details on the relationship between input and output port widths. You also must explicitly set a value for the parameter in your instantiation statement.

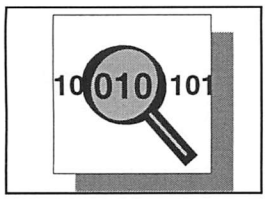
```
module DW01_decode_inst( inst_A, B_inst );

  parameter width = 8;

  input [width-1 : 0] inst_A;
  output [(1<<width)-1 : 0] B_inst;

  // Instance of DW01_decode
  DW01_decode #(width)
    U1 ( .A(inst_A), .B(B_inst) );

endmodule
```

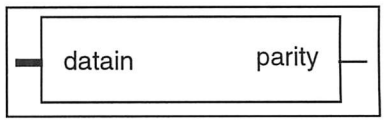


DW04_par_gen

Parity Generator and Checker

Last Revised: Release DWF_0206

Features and Benefits



- Generates parity for given input data
- Supports even and odd parity, selectable via a parameter
- Supports variable word widths
- Inferable using a function call

Applications

- Parity generation and checking on external DRAM
- Parity generation and checking on internal or external SRAM
- Parity generation and checking on words transmitted in telecommunications, via UART, disk drives, etc.

Description

The DW04_par_gen is a parity generator and checker circuit that is designed for systems such as computers, peripherals, and communications devices that require improved data integrity over unprotected systems.

Table 1: Pin Description

Pin Name	Width	Direction	Function
datain	*width* bit(s)	Input	Input data word to check or generate parity
parity	1 bit	Output	Generated parity (see Table 5)

Table 2: Parameter Description

Parameter	Values[a]	Description
width	1 to 256	Defines the width of the input bus
par_type	0 or 1	Defines the type of parity

a. The upper bound of the legal range is a guideline to ensure reasonable compile times.

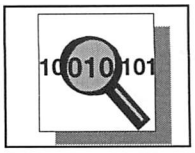


Table 3: Synthesis Implementations

Implementation Name	Function	License Feature Required
str	Synthesis model	DesignWare

Table 4: Simulation Models

Model	Function
DW04.DW04_PAR_GEN__SIM	Design unit name for VHDL simulation
dw/dw04/src/DW04_par_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW04_par.v	Verilog simulation model source code

Table 5: Truth Table

datain	par_type	parity
even # 1's	0 (even)	1
odd # 1's	0 (even)	0
even # 1's	1 (odd)	0
odd # 1's	1 (odd)	1

When input data is applied to `datain`, which is *width* bits in size, parity is generated at the output, `parity`, based on the value of the `par_type` parameter. This parameter selects whether odd or even parity is generated, as shown in Table 5 on page 2.

Application Example

Extending the DW04_par_gen to a parity checker is accomplished by performing an exclusive-OR of the parity output of DW04_par_gen with the parity bit of the parity-checked input word received.

Related Topics

- Application Specific – Data Integrity Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Function Inferencing - VHDL

```
library IEEE, DW04;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DW04.DW04_components.all;

entity DW04_par_gen_func is
  generic(func_width :integer:=8;   func_par_type: INTEGER := 1);
  port(func_data_in : in std_logic_vector(func_width-1 downto 0);
       parity_func  : out std_logic);
end DW04_par_gen_func;

architecture func of DW04_par_gen_func is
begin
  parity_func <= DWF_parity_gen(func_par_type, func_data_in);
end func;
```

HDL Usage Through Function Inferencing - Verilog

DW_parity_gen is an obsoleted function. Using this function multiple times may cause inconsistent logic when the initial parameter value is smaller than the actual parameter value. This is because HDL Compiler currently does not support defparam Verilog functions. For the detailed information, refer to STARs 59352 and 59348.

This function is released only for backward compatibility. New users should not use this function. Instead, new users should use this part through component instantiation.

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,DW04;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DW04.DW04_components.all;

entity DW04_par_gen_inst is
  generic (inst_width    : POSITIVE := 8;
           inst_par_type : INTEGER := 1  );
  port (inst_datain : in std_logic_vector(inst_width-1 downto 0);
        parity_inst : out std_logic  );
end DW04_par_gen_inst;
```


```
architecture inst of DW04_par_gen_inst is
begin

  -- Instance of DW04_par_gen
  U1 : DW04_par_gen
    generic map (width => inst_width,    par_type => inst_par_type )
    port map ( datain => inst_datain,    parity => parity_inst );
end inst;

-- pragma translate_off
configuration DW04_par_gen_inst_cfg_inst of DW04_par_gen_inst is
  for inst
  end for; -- inst
end DW04_par_gen_inst_cfg_inst;
-- pragma translate_on
```

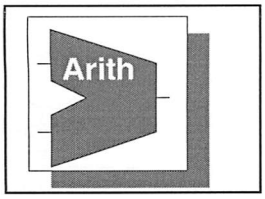
HDL Usage Through Component Instantiation - Verilog

```
module DW04_par_gen_inst( inst_datain, parity_inst );

  parameter width = 8;
  parameter par_type = 1;

  input [width-1 : 0] inst_datain;
  output parity_inst;

  // Instance of DW04_par_gen
  DW04_par_gen #(width, par_type)
    U1 ( .datain(inst_datain),    .parity(parity_inst) );
endmodule
```

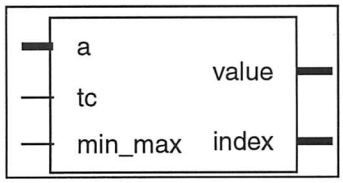


DW_minmax

Minimum/Maximum Value

Last Revised: Release DWF_0206

Features and Benefits



- Parameterized number of inputs
- Parameterized word length
- Unsigned and signed (two's complement) data operation
- Dynamically selectable mode (minimum or maximum)
- Additional output gives an index of the minimum or maximum input
- Inferable using a function call

Description

DW_minmax determines the minimum or maximum value of multiple inputs. The *num_inputs* input operands of *width* length must be concatenated into a single input vector (a) of *num_inputs* × *width* length. The `value` output is the minimum of all inputs if `min_max`=0, and the maximum if `min_max`=1. The inputs and the `value` output are interpreted as unsigned numbers if `tc`=0, and as signed numbers if `tc`=1.

Table 1: Pin Description

Pin Name	Width	Direction	Function
a	*num_inputs* × *width* bit(s)	Input	Concatenated input data
tc	1 bit	Input	Two's complement control
min_max	1 bit	Input	Minimum/maximum control 0 = minimum (a) 1 = maximum (a)
value	*width* bit(s)	Output	Minimum/maximum value
index	ceil($\log_2$[*num_inputs*]) bit(s)	Output	Index of minimum/maximum input

Table 2: Parameter Description

Parameter	Values	Description
width	≥ 1	Input word length
num_inputs	≥ 2 Default: 2	Number of inputs

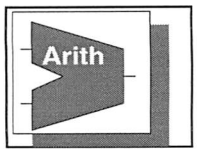


Table 3: Synthesis Implementations[a]

Implementation Name	Function	License Feature Required
cla	Carry-lookahead tree synthesis model	DesignWare
clas	Carry-lookahead/select tree synthesis model	DesignWare

a. During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the *DesignWare Building Block IP User Guide*.

Table 4: Simulation Models

Model	Function
DW01.DW_minmax_cfg_sim	Design unit name for VHDL simulation
dw/dw01/src/DW_minmax_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_minmax.v	Verilog simulation model source code

Table 5: Functional Description

min_max	a	value
0	a*n* & ... & a1 & a0[a]	min (a*n*, ..., a1, a0)
1	a*n* & ... & a1 & a0	max (a*n*, ..., a1, a0)

a. "&" denotes concatenation in VHDL (Verilog: "{a*n*, ..., a1, a0}")

The `index` output gives the index of the minimum or maximum input as a binary coded number. Therefore, the right-most input within the concatenated input vector has index 0, and the left-most input has index *num_inputs*–1. If multiple inputs are equal and minimum, the lowest of the indices is given; if multiple inputs are equal and maximum, the highest of the indices is given.

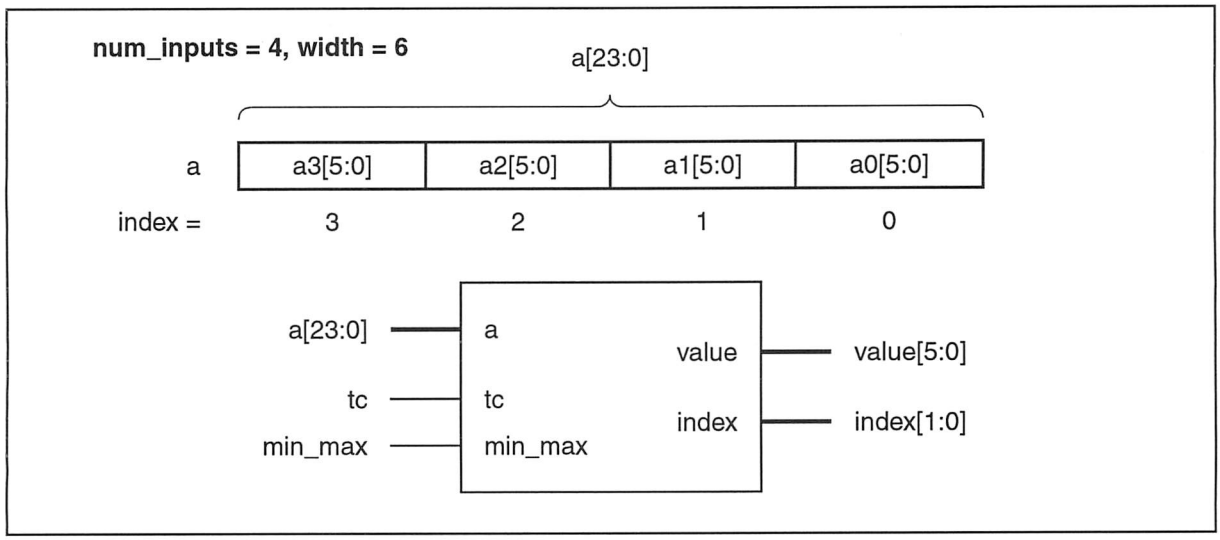
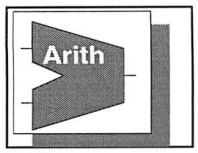


Figure 1: Functional Operation

Related Topics

- Math – Arithmetic Overview
- DesignWare Building Block IP Documentation Overview



HDL Usage Through Function Inferencing - VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_Foundation_arith.all;

-- example: functional inference of 4-input minimum/maximum component
entity DW_minmax_func is
  generic ( wordlength : natural := 8);
  port ( a0, a1, a2, a3 : in  std_logic_vector(wordlength-1 downto 0);
         tc             : in  std_logic;
         max            : in  std_logic;
         val            : out std_logic_vector(wordlength-1 downto 0) );
end DW_minmax_func;

architecture func of DW_minmax_func is
begin

  -- function calls for unsigned/signed minimum/maximum
  -- inputs are concatenated into the first function call operand "a"
  -- the second function call operand is "num_inputs"
  process (a0, a1, a2, a3, max, tc)
  begin
    if max = '0' then
      if tc = '0' then
        val <= std_logic_vector(DWF_min(unsigned(a3 & a2 & a1 & a0), 4));
      else
        val <= std_logic_vector(DWF_min(signed(a3 & a2 & a1 & a0), 4));
      end if;
    else
      if tc = '0' then
        val <= std_logic_vector(DWF_max(unsigned(a3 & a2 & a1 & a0), 4));
      else
        val <= std_logic_vector(DWF_max(signed(a3 & a2 & a1 & a0), 4));
      end if;
    end if;
  end process;

end func;
```

HDL Usage Through Function Inferencing - Verilog

```
// example: functional inference of 4-input minimum/maximum component
module DW_minmax_func (a0, a1, a2, a3, tc, max, val);

  parameter wordlength = 8;

  input  [wordlength-1 : 0] a0, a1, a2, a3;
  input  tc, max;
  output [wordlength-1 : 0] val;

  // pass "width" and "num_inputs" parameters to the inference functions
  parameter width = wordlength;
  parameter num_inputs = 4;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW_minmax_function.inc"

  // function calls for unsigned/signed minimum/maximum
  // inputs are concatenated into the function call operand
  assign val =
    (max == 1'b0) ? ((tc == 1'b0) ? DWF_min_uns({a3, a2, a1, a0}) :
                                    DWF_min_tc ({a3, a2, a1, a0})) :
                    ((tc == 1'b0) ? DWF_max_uns({a3, a2, a1, a0}) :
                                    DWF_max_tc ({a3, a2, a1, a0}));
endmodule
```

HDL Usage Through Component Instantiation - Verilog

```
// example: instantiation of 4-input minimum/maximum component
module DW_minmax_inst (a0, a1, a2, a3, tc, max, val, idx);

  parameter wordlength = 8;

  input  [wordlength-1 : 0] a0, a1, a2, a3;
  input  tc, max;
  output [wordlength-1 : 0] val;
  output [1 : 0] idx;

  // instantiation of DW_minmax
  // inputs are concatenated into the input vector
  DW_minmax #(wordlength, 4)
    U1 (.a({a3, a2, a1, a0}), .tc(tc), .min_max(max),
        .value(val), .index(idx));

endmodule
```