Comparator v8.0

DS224 April 28, 2005 **Product Specification**

Features

- Drop-in module for Virtex™, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan™-II, Spartan-IIE, Spartan-3 and Spartan-3E FPGAs
- Generates comparison logic for A = B, A <> B, A <= B, A < B, A >= B or A > B
- Operates on twos-complement signed or unsigned data
- Supports inputs from 1 to 256 bits wide
- Optional compare-to-constant capability
- Optional clock enable, asynchronous and synchronous controls on synchronous outputs
- For use with CORE Generator system v7.1i Service Pack 2 or higher.
- Incorporates Xilinx Smart-IP™ technology for utmost parameterization and optimum implementation

Functional Description

The comparator is used to create comparison logic that performs one of the following functions: A = B, A <> B, A <= B, A < B, A >= B or A > B. A and B are external ports of up to 256 bits wide and B can optionally be set to a constant value. The module can operate on twos-complement signed or unsigned data. Options are provided for Clock Enable, Asynchronous Set and Clear, and Synchronous Set and Clear. It is possible to implement pipelined versions of the core to improve performance.

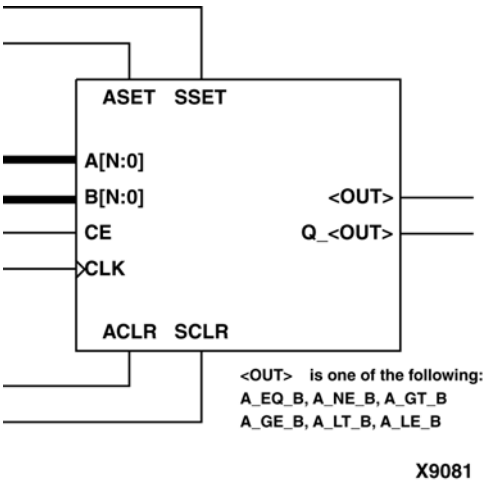

Figure 1: **Core Schematic Symbol**

Pinout

Signal names for the schematic symbol are shown in Figure 1 and described in Table 1.

Table 1: **Core Signal Pinout**

Name	Direction	Description
A[N:0]	Input	Comparator Input
B[N:0]	Input	Comparator Input
A_EQ_B	Output	A Equals B Asynchronous Output
QA_EQ_B	Output	A Equals B Registered Output
A_NE_B	Output	A Not Equal to B Asynchronous Output
QA_NE_B	Output	A Not Equal to B Registered Output
A_LE_B	Output	A Less Than or Equal to B Asynchronous Output
QA_LE_B	Output	A Less Than or Equal to B Registered Output
A_LT_B	Output	A Less Than B Asynchronous Output

Table 1: **Core Signal Pinout** ***(Continued)***

Name	Direction	Description
QA_LT_B	Output	A Less Than B Registered Output
A_GE_B	Output	A Greater Than or Equal to B Asynchronous Output
QA_GE_B	Output	A Greater Than or Equal to B Registered Output
A_GT_B	Output	A Greater Than B Asynchronous Output
QA_GT_B	Output	A Greater Than B Registered Output
CE	Input	Clock Enable
CLK	Input	Clock - rising edge clock signal
ASET	Input	Asynchronous Set - forces the registered output to a high state when driven
ACLR	Input	Asynchronous Clear - forces outputs to a low state when driven
SSET	Input	Synchronous Set - forces the registered output to a high state on next concurrent clock edge
SCLR	Input	Synchronous Clear - forces the registered output to a low state on next concurrent clock edge

Notes:

All control inputs are Active High. Should an Active Low input be required for a particular control pin an inverter must be in the path to the pin. The inverter will be absorbed appropriately by the core logic during mapping.

XCO and VHDL Generic Parameters

XCO parameters and VHDL generics are broadly equivalent. Table 2 describes the parameters, legal values and meaning of each. The descriptions below refer to the VHDL generic parameters:

- **c_width** (integer): Specifies the width of the operands to be compared. This should be in the range 1 to 256 bits for unsigned data and 2 to 256 for signed data. The default value is 16.
- **c_ainit_val** (string): Specifies the power-on reset initialization value for all registers used in the core (including any pipeline registers). The default value is "0".
- **c_data_type** (integer): Specifies whether the operands are signed or unsigned data. Valid values are 0 (signed) and 1 (unsigned). The default value is 0.
- **c_b_constant** (integer): Specifies if the B value that the data on the A port is to be compared to is a constant. The constant value can be specified with the **c_b_value** generic. Valid options are 0 (variable B port value) and 1 (constant B port value). The default value is 0.
- **c_b_value** (string): Specifies the constant value to compare to when the B operand has been selected to be a constant (by setting **c_b_constant** to 1). If this string is shorter than the width of the A port (defined by **c_width**), the string will be sign-extended up to the MSB (left side of the string) when **c_data_type** = 0 (signed comparison) and zero-extended if **c_data_type** = 1 (unsigned comparison). If the string is longer than **c_width** then it will be truncated at the MSB. The default value is "0000000000000000". The Comparator GUI will allow the value to be entered in binary, decimal (up to 32 bits wide) or hexadecimal.
- **c_pipe_stages** (integer): Specifies the degree of pipelining to be implemented in the core. The number of pipeline stages is limited by the width and/or the operation being performed.
  - **= or <> operation**: Up to 4 levels of pipelining are possible. The maximum possible result latency is 5 cycles (including the optional output register). The number of possible pipeline stages increases with width:

    For comparisons versus a variable B value:
    - > 2 bits wide - up to 1 pipeline stage
    - > 8 bits wide - up to 2 pipeline stages
    - > 32 bits wide - up to 3 pipeline stages
    - > 128 bits wide - up to 4 pipeline stages

    For comparisons versus a constant B value
    - > 4 bits wide - up to 1 pipeline stage
    - > 16 bits wide - up to 2 pipeline stages
    - > 64 bits wide - up to 3 pipeline stages
  - **<, >, <= or >= operation**: A maximum of one level of pipelining can be implemented (with an optional output register), so the maximum possible latency is 2 cycles.
- **Output Options**: The core can have registered and/or non-registered output ports, but only one operation can be implemented per core instance. Set a generic to 0 to disable an output or set it to 1 to enable that output port. When a pipelined comparison is selected, the asynchronous output is still available, but will be subject to the pipeline latency.
  - **c_has_a_eq_b** (integer): Specifies if the core has an asynchronous A = B output port. This is the default output.
  - **c_has_a_ne_b** (integer): Specifies if the core has an asynchronous A <> B output port.

- **c_has_a_lt_b** (integer): Specifies if the core has an asynchronous A < B output port.
- **c_has_a_gt_b** (integer): Specifies if the core has an asynchronous A > B output port.
- **c_has_a_le_b** (integer): Specifies if the core has an asynchronous A <= B output port.
- **c_has_a_ge_b** (integer): Specifies if the core has an asynchronous A >= B output port.
- **c_has_qa_eq_b** (integer): Specifies if the core has a synchronous A = B output port.
- **c_has_qa_ne_b** (integer): Specifies if the core has a synchronous A <> B output port.
- **c_has_qa_lt_b** (integer): Specifies if the core has a synchronous A < B output port.
- **c_has_qa_gt_b** (integer): Specifies if the core has a synchronous A > B output port.
- **c_has_qa_le_b** (integer): Specifies if the core has a synchronous A <= B output port.
- **c_has_qa_ge_b** (integer): Specifies if the core has a synchronous A >= B output port

- **c_enable_rlocs** (integer): Redundant in the Version 8 core. The default value is 0.
- **c_sync_enable** (integer): This parameter controls whether or not the SSET, SCLR, and SINIT inputs are qualified by CE. When **c_sync_enable** = 0, these synchronous controls override the CE signal. When **c_sync_enable** = 1, they have effect only when CE is high. This parameter is ignored unless **c_has_ce** = 1. Note that on the primitives, the SCLR and SSET controls override CE, so choosing **c_sync_enable** = 1 will generally result in extra logic. The default is **c_sync_enable** = 0.
- **c_sync_priority** (integer): Controls the relative priority of SCLR and SSET. When **c_sync_priority** = 1, SCLR overrides SSET. When **c_sync_priority** = 0, SSET overrides SCLR. The default is 1, since this is the way the primitives behave, hence results in no extra logic.
- **c_has_ce** (integer): Specifies if the core has a clock enable for the output register and the pipeline registers. Possible values are 0 or 1. The default value is 0 (no CE pin).

Asynchronous Settings: All asynchronous controls are implemented using the dedicated inputs on the flip-flop primitives.

- **c_has_aclr** (integer): Specifies if the core has an asynchronous clear pin on the output register. Possible values are 0 or 1. The default value is 0 (no ACLR pin).
- **c_has_aset** (integer): Specifies if the core has an asynchronous set pin on the output register. Possible values are 0 or 1. The default value is 0 (no ASET pin).

Synchronous Settings: When no asynchronous register controls are requested, the synchronous register controls can be implemented using the dedicated inputs on the flip-flop primitives. Exceptions to this are described in the sections for the **c_sync_priority** and **c_sync_enable** parameters.

- **c_has_sclr** (integer): Specifies if the core has a synchronous clear pin on the output register AND the pipeline registers. Possible values are 0 or 1. The default value is 0 (no SCLR pin).
- **c_has_sset** (integer): Specifies if the core has a synchronous set pin on the output register. Possible values are 0 or 1. The default value is 0 (no SSET pin).

.

Table 2: **XCO and VHDL Generic Parameters**

XCO Parameter	XCO Values	Generic Parameter	Generic Values	Description
Width	(Integer) 1 to 256 (unsigned), 2 to 256 (signed)	c_width	(Integer) 1 to 256 (unsigned), 2 to 256 (signed)	Width of the operands to be compared.
DataType	Signed, Unsigned	c_data_type	(Integer) 0, 1	Indicates if the operands are signed or unsigned data. 0 = signed, 1 = unsigned
ConstantBPort	(Boolean) false, true	c_b_constant	(Integer) 0, 1	Specifies if the value applied to the A port is to be compared to an internal constant B value. 0 (false) = B not constant 1 (true) = B constant

Table 2: **XCO and VHDL Generic Parameters**

XCO Parameter	XCO Values	Generic Parameter	Generic Values	Description
ConstantBPortValue	(String) e.g. "00000000000" with base specified by Radix XCO parameter	c_b_value	(String) e.g. "0000000000" with binary values only	Value of the B operand for comparison against a constant. The number base of the XCO value is determined by the Radix XCO parameter. Only binary strings should be passed as generic values.
Radix	(Integer) 2, 10 or 16	None	None	Specifes the radix of the ConstantBPortValue XCO value. 2 = binary, 10 = decimal 16 = hexadecimal. A radix value of 10 is only available up to and including a width of 32 bits. Above this, only 2 and 16 are valid radix values.
PipelineStages	(Integer) 0, 1 for <, <=, >, >= comparisons. 0, 1, 2, 3 for = and <> comparisons	c_pipe_stages	(Integer) 0, 1 for <, <=, >, >= comparisons. 0, 1, 2, 3 for = and <> comparisons	Specifies the number of pipeline stages to implement - restricted by the width of the comparison in the cases of = and <> comparisons. See above for details.
CE	(Boolean) false, true	c_has_ce	(Integer) 0, 1	0 (false) = No CE control 1 (true) = Has CE control
ACLR	(Boolean) false, true	c_has_aclr	(Integer) 0, 1	0 (false) = No ACLR control 1 (true) = Has ACLR control
ASET	(Boolean) false, true	c_has_aset	(Integer) 0, 1	0 (false) = No ASET control 1 (true) = Has ASET control
SCLR	(Boolean) false, true	c_has_sclr	(Integer) 0, 1	0 (false) = No SCLR control 1 (true) = Has SCLR control
SSET	(Boolean) false, true	c_has_sset	(Integer) 0, 1	0 (false) = No SSET control 1 (true) = Has SSET control
AInitVal	(String) Single binary value e.g. "0"	c_ainit_val	(String) Single binary value e.g. "0"	Power-on reset initialization value for all registers in the design in the absence of any register controls
Operation	(String) "eq", "ne", "lt", "gt", "le", "ge"	Determines which of the RegisteredOutput and NonRegisteredOutput ports are available		The operation determines which of the output ports are available. For example, if "eq" is selected as the operation, only c_has_qa_eq_b and c_has_a_eq_b are available as core outputs

Table 2: **XCO and VHDL Generic Parameters**

XCO Parameter	XCO Values	Generic Parameter	Generic Values	Description
RegisteredOutput	(Boolean) false, true	c_has_qa_eq_b	(Integer) 0, 1	Synchronous "equal-to" output. Only available when selected operation is "eq". 0 (false) = no synchronous "=" output 1 (true) = use synchronous "=" output
		c_has_qa_ne_b	(Integer) 0, 1	Synchronous "not-equal-to" output. Only available when selected operation is "ne". 0 (false) = no synchronous "<>" output 1 (true) = use synchronous "<>" output
		c_has_qa_lt_b	(Integer) 0, 1	Synchronous "less-than" output. Only available when selected operation is "lt". 0 (false) = no synchronous "<" output 1 (true) = use synchronous "<" output
		c_has_qa_gt_b	(Integer) 0, 1	Synchronous "greater-than" output. Only available when selected operation is "gt". 0 (false) = no synchronous ">" output 1 (true) = use synchronous ">" output
		c_has_qa_le_b	(Integer) 0, 1	Synchronous "less-than-or-equal-to" output. Only available when selected operation is "le". 0 (false) = no synchronous "<=" output 1 (true) = use synchronous "<=" output
		c_has_qa_ge_b	(Integer) 0, 1	Synchronous "greater-than-or-equal-to" output. Only available when selected operation is "ge". 0 (false) = no synchronous ">=" output 1 (true) = use synchronous ">=" output

Table 2: **XCO and VHDL Generic Parameters**

XCO Parameter	XCO Values	Generic Parameter	Generic Values	Description
NonRegisteredOutput	(Boolean) false, true	c_has_a_eq_b	(Integer) 0, 1	Asynchronous "equal-to" output. Only available when selected operation is "eq". 0 (false) = no asynchronous "=" output, 1 (true) = use asynchronous "=" output
		c_has_a_ne_b	(Integer) 0, 1	Asynchronous "not-equal-to" output. Only available when selected operation is "ne". 0 (false) = no asynchronous "<>" output, 1 (true) = use asynchronous "<>" output
		c_has_a_lt_b	(Integer) 0, 1	Asynchronous "less-than" output. Only available when selected operation is "lt". 0 (false) = no asynchronous "<" output, 1 (true) = use asynchronous "<" output
		c_has_a_gt_b	(Integer) 0, 1	Asynchronous "greater-than" output. Only available when selected operation is "gt". 0 (false) = no asynchronous ">" output, 1 (true) = use asynchronous ">" output
		c_has_a_le_b	(Integer) 0, 1	Asynchronous "less-than-or-equal-to" output. Only available when selected operation is "le". 0 (false) = no asynchronous "<=" output, 1 (true) = use asynchronous "<=" output
		c_has_a_ge_b	(Integer) 0, 1	Asynchronous "greater-than-or-equal-to" output. Only available when selected operation is "ge". 0 (false) = no asynchronous ">=" output, 1 (true) = use asynchronous ">=" output
SyncCtrlPriority	Set_Overrides_Reset, Reset_Overrides_Set	c_sync_priority	(Integer) 0, 1	0 = Set overrides Reset, 1 = Reset overrides Set
CEPriority	Sync_Overrides_CE, CE_Overrides_Sync	c_sync_enable	(Integer) 0, 1	0 = Sync overrides CE, 1 = CE overrides sync

Core Resource Utilization

The non-pipelined core configurations are optimized for area rather than speed, apart from the case of the equal-to and not-equal-to comparisons below 32 bits which utilize LUTs rather than the carry chain to give significant speed improvement with a minimal increase in resources. The variable inequality comparisons are implemented using LUTs or the carry chain to give the best possible resource usage and speed trade-off.

The pipelined configurations offer, where possible, the ideal LUT -> register -> LUT configuration and, where the carry chain has been used, the carry chain lengths between registers are optimized to balance the input-stage carry chain length with the output-stage chain length to give the best speed performance with an associated increase in resource usage.

Pipeline Register Controls

The register controls on any pipeline registers used in the core are implemented differently from the output register controls. If no synchronous controls are used, asynchronous reset and set controls are available on the pipeline registers. If synchronous controls are used, asynchronous controls will not be implemented on the pipeline registers, but will still be present on any output register used. Clock Enable controls are implemented on all registers if the CE option is selected.

Applications

There are two methods to include a comparator module in your design.

Method 1: GUI

Core Generator will produce several files when a core is generated. Instructions on how to instantiate a core generated by this method are automatically produced in the .vho file. Here is an example of a section of a .vho file:

-- The following code must appear in the VHDL architecture header:

------------ Begin Cut here for COMPONENT Declaration

component compare_v8_0

port (

a: IN std_logic_VECTOR(15 downto 0);

b: IN std_logic_VECTOR(15 downto 0);

clk: IN std_logic;

sclr: IN std_logic;

qa_eq_b: OUT std_logic);

end component;

------ End COMPONENT Declaration ----------

The following code must appear in the VHDL architecture body. Substitute your own instance name and net names.

------------ Begin Cut here for INSTANTIATION Template

your_instance_name : compare_v8_0

port map (

a => a,

b => b,

clk => clk,

sclr => sclr,

qa_eq_b => qa_eq_b);

----- End INSTANTIATION Template ------------

-- You must compile the wrapper file compare_v8_0.vhd when simulating the core, compare_v8_0. When compiling the wrapper file, be sure to reference the XilinxCoreLib VHDL simulation library. For detailed instructions, please refer to the "CORE Generator Help".

Method 2: Direct Instantiation

Coregen now allows cores to be directly instantiated into user code. To do this, add the following lines to the head of your VHDL file

Library XilinxCoreLib;

Use XilinxCoreLib.c_compare_v8_0_comp.all;

and instantiate the counter with appropriate values for the generics and your local signals thus:

i_my_compare c_compare_v8_0
generic map(
c_width => 16,
c_has_sclr => 1
c_has_qa_eq_b => 1,
c_has_a_eq_b => 0);
port map(
a => a ,
b => b ,
clk => clk,
sclr => sclr_i,
qa_eq_b => compare_output);

Note that generics do not need to be specified if the default value suits your application.

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator system v7.1i and later. The CORE Generator system is bundled with the ISE Foundation software at no additional charge.