WYSIWYG Device Primitives User Guide For Stratix

Version 1.52

November 26, 2007

Table of Contents:

1. Stratix Logic Cell

The logic cell of the Stratix™ devices is most similar to that of the Mercury™ devices. Its major differences from Mercury are that:

(1) It does not have a preset or a multiplier mode.

(2) It has a new inverta signal that allows more efficient implementation of adder/subtractor modules

(3) It has a new synch-enable mode for the LE to allow access to sload and sclear signals in normal mode.

(4) There are new modes that allow the register and LUT in an lcell to be connected in new configurations.

1.1 Logic Cell Primitive

```
stratix_lcell <lcell_name>
(
      .clk(<clock source>),
      .dataa(<data_a source>),
      .datab(<data_b source>),
      .datac(<data_c source>),
      .datad(<data_d source>),
      .aclr(<asynchronous clear source>),
      .aload(<asynchronous load source>),
      .sclr(<synchronous clear source>),
      .sload(<synchronous load source>),
      .ena(<clock enable source>),
      .cin(<carry in source>),
      .cin0(<internal carry in source for 0 carry chain>),
      .cin1(<internal carry in source for 1 carry chain>),
      .inverta(<inverts .dataa into the lut>),
      .regcascin(<register cascade source>),

      .combout(<combinational output>),
      .regout(<registered output>),
      .cout(<carry output>),
      .cout0(<internal carry output for 0 carry chain>),
      .cout1(<internal carry output for 1 carry chain>),
);
defparam <lcell_name>.operation_mode = <operation mode>;
defparam <lcell_name>.synch_mode = <synchronous usage mode>;
defparam <lcell name> register_cascade_mode = <reg cascade usage>
defparam <lcell_name>.sum_lutc_input = <sum lut input choice>;
```

```
defparam <lcell_name>.lut_mask = <lut mask>;
defparam <lcell_name>.created_from = <created from register name>;
```

1.2 Logic Cell Input Ports

<*lcell name*>**:** is the unique identifier for the logic cell. This is any identifier name which is legal for the given description language (e.g. Verilog, VHDL, AHDL, etc.). This field is required.

.clk(<*clock source*>**):** designates the clock input to the logic cell, which drives the clock input on the logic cell register. The logic cell cannot have registered output (or qfbk) without the .clk port specified.

.**dataa(**<*data_a source*>**), … .datad(**<*data_d source*>**):** are the four LUT inputs for the logic cell. .dataa corresponds to DATA1 in the data book, and .datad corresponds to DATA4.

For the Stratix architecture the .datac input can be stolen for the following purposes: synchronous or asynchronous data to the register. The .datac port cannot be directly set to gnd.

Note that the LUT inputs may be used for different purposes depending on the mode of the logic cell and may be rotated to improve speed and fitting.

.aclr(<*asynchronous clear source*>**)** designates the asynchronous clear signal for the logic cell. Only allowed when the .clk port on the logic cell is specified

.aload(<*asynchronous load source*>**),** designates the asynchronous load signal for the logic cell. The asynchronous data signal is taken from .datac, and .datac must be connected. Only allowed when the .clk port on the logic cell is specified.

.sclr(<*synchronous clear source*>**),** designates the synchronous clear signal for the logic cell. Only allowed when the .clk port on the logic cell is specified. Only one sclr value is allowed in a single logic array block (LAB), which contains 10 logic cells, so sclr should not be used for low-fanout signals. It should be used only for true, high-fanout synchronous clear signals, or a serious degradation in fitting and circuit speed will result. This port can only be connected on a cell that has synch_mode = on.

.sload(<*synchronous load source*>**),** designates the synchronous load signal for the logic cell. When .sload is high, the register is synchronously loaded from .datac. Hence .datac must be connected if .sload is connected. Only allowed when the .clk port on the logic cell is specified. Only one sload value is allowed in a single logic array block (LAB), which contains 10 logic cells, so sloads should not be used for low-fanout signals This port can only be connected on a cell that has synch_mode = on.

.ena(<*clock enable source*>**),** designates the clock enable signal for the logic cell. Only allowed when the .clk port on the logic cell is specified.

.cin(<*carry in source*>**)** is the carry-in signal of the logic cell. The signal should come from the .cout port of another logic cell. It cannot be directly set to VCC/GND, or inverted.

In post-fitting simulation, this port is used to represent the block-carry input to the logic cell.

It is an error to have .cin connected when the operation mode is normal unless the

parameter sum_lutc_input is set to “cin”.

.cin0(<internal *carry in source for 0 carry chain* >**)** is the “internal” carry-in signal for the 0 carry-chain of the logic cell. It is used in post-fitting simulation to more accurately model the exact carry circuitry of the LE. It should never be used in an input netlist**.**

This signal should come from the .cout0 port of another logic cell. It cannot be directly set to VCC/GND, or inverted.

.cin1(<internal *carry in source for 1 carry chain* >**)** is the “internal” carry-in signal for the 1 carry-chain of the logic cell. It is used in post-fitting simulation to more accurately model the exact carry circuitry of the LE. It should never be used in an input netlist**.**

This signal should come from the .cout1 port of another logic cell. It cannot be directly set to VCC/GND, or inverted.

.inverta(<*invert the dataa input* >**)** is a signal that inverts the dataa input of the lcell. For the first cell of the carry chain (has .cout and no .cin), this signal is also connected internally to the .cin port of the cell. This signal is used to build efficient addsub modules that take 1 LE per it rather than 2 LEs per bit. This signal can only be used if the cell has either .cin or .cout used.

.regcascin(<*register cascade source*>**)** is a signal that feeds directly into the register input. This port can only be connected on a logic cell when *register_cascade_mode = on*. The source of a .regcascin signal must be the .regout port of another logic cell, and fitting will be forced to place the source logic cell in the LE position immediately before the receiving logic cell (e.g. place the source in LE #1 and the destination in LE #2).

1.3 Logic Cell Output Ports

.combout(<*combinational output*>**)** is the combinational output of the logic cell.

.regout(<*registered output*>**)** is the registered output of the logic cell.

Timing assignments can be made to the register for a logic cell in qfbk mode by simply specifying that the regout port connects to a wire with no fanout.

.cout(<*carry output*>**)** is the carry out of the logic cell. Note that if .cout is connected to another logic cell the signal must go to the logic cell’s .cin port.

In post-fitting simulation, this port is used to represent the block-carry output of the logic cell.

.cout0(<internal *carry output for 0 carry chain*>**)** is the “internal” carry-out signal for the 0 carry-chain of the logic cell. It is used in post-fitting simulation to more accurately model the exact carry circuitry of the LE. It should never be used in an input netlist**.**

If connected to another logic cell, this signal must go to the desination logic cell’s .cin0 port.

.cout1(<internal *carry output for 1 carry chain*>**)** is the “internal” carry-out signal for the 1 carry-chain of the logic cell. It is used in post-fitting simulation to more accurately model the exact carry circuitry of the LE. It should never be used in an input netlist**.**

If connected to another logic cell, this signal must go to the desination logic cell's .cin1 port.

1.4 Logic Cell Modes

<*operation_mode*> is one of *{normal, arithmetic}*. In normal mode, the look-up table (LUT) computes a single function of up to 4 inputs. The 4 inputs come from .dataa (inverted or not according to inverta), .datab, the output of the sum_lutc_input, and .datad (see Figure 2).

In arithmetic mode the look-up table is split into two 3-input look-up tables, one to compute the sum and the other to compute the carry. Hence .datad cannot be connected. The regular look-up table output computes the sum as a function of .dataa, .datab, and the output of the sum_lutc_input. The second look-up table computes the carry signal as a function of .dataa, .datab, and .cin (or inverta for the first cell of a carry chain). The sum_lutc_input is ignored for the second look-up table. Arithmetic mode can only be specified a logic cell that has its .cout port connected.

There is no qfbk_counter in Stratix. Like Mercury, counter mode has been removed, and arithmetic mode should be used instead for counters. *This field is required.*

<*synch_mode*> is one of *{off, on}*. This mode represents what synchronous control signals are available for the register.

If the mode is *off*, then .sload and .sclr have to be disconnected.

If the mode is *on*, then either .sload or .sclr must be used. It is an error for the mode to be *on* when neither .sload nor .sclrl are used. It also means that this lcell is affected by the lab wide .sload and .sclr even if does not use them itself.

This field is optional, and defaults to off. This field has no meaning if the register in a logic cell is not used. It is an error to have synch_mode = on and to leave the .clk disconnected.

<*register_cascade_mode*> is one of *{off, on}*. This mode represents if the register cascade feeds the current register or not.

If the mode is *on*, then regcascin has to be connected. The .regcascin is sourced by the regout of the "previous" logic cell in the register cascade chain. This mode allows efficient implementation of shift registers; a shift register can be implemented in a chain of logic cell registers, and all the logic cell look-up tables are still available for use. We recommend synthesis not use this mode, and instead let the Quartus® II fitter use this mode to collapse logic cells together for maximum density, without impacting fitting.

If the mode is off, then .regcascin has to be disconnected.

This field is optional, and defaults to off. This field has no meaning if the register in a logic cell is not used. It is an error to have register_cascade_mode = on and to leave the .clk disconnected.

<*sum_lutc_input*> is one of *{datac, cin, qfbk}*. This mode represents what feeds the datac "location" of the sum LUT, independent of the mode of the cell.

If the mode is datac then the upper LUT (or the full 4-input LUT) is fed by the datac signal

If the mode is cin, then the upper LUT (or the full 4 input LUT) is fed by the cin signal

If the mode is qfbk, then the upper LUT (or the full 4 input LUT) is fed by the Q output of the register in this logic cell.

Note that the input .cin cannot be connected in normal mode unless sum_lutc_input is set to cin.

This field is optional. It defaults to cin if the .cin port is connected; otherwise, it defaults to datac. This default behavior occurs in both normal and arithmetic modes.

<*lut_mask*> designates the content of the LUT(s) in the logic cell. It is a four digit hexadecimal number representing the 16 bits of data in the LUT. The LUT mask uses positive logic, and is formatted as follows:

Table 1 shows how the data is ordered when the bits are ordered as: o_{16} o_{15} o_{14} … o_3 o_2 o_1

Table 1 -- LUT Mask Data Ordering when not in arithmetic mode.

datad	Datac or cin or qfdbk	datab	dataa	LUT Output
0	0	0	0	o_1
0	0	0	1	o_2
0	0	1	0	o_3
0	0	1	1	o_4
0	1	0	0	o_5
0	1	0	1	o_6
0	1	1	0	o_7
0	1	1	1	o_8
…	…	…	…	…
1	1	1	1	O_{16}

Note that when the cell is in arithmetic mode, two three-input LUTs implement the sum logic and carry logic. In this case the LUT mask is divided in half, with the bits corresponding to datad = 1 going to the sum logic and the bits corresponding to datad = 0 going to the carry logic. The sum logic feeds the “regular” LUT output that always drives .combout and also drives the logic cell register input except when register cascade is used. The carry logic feeds the .cout signal. Figure 1 shows how the LUT mask AA34 is mapped onto the sum and carry LUTs.

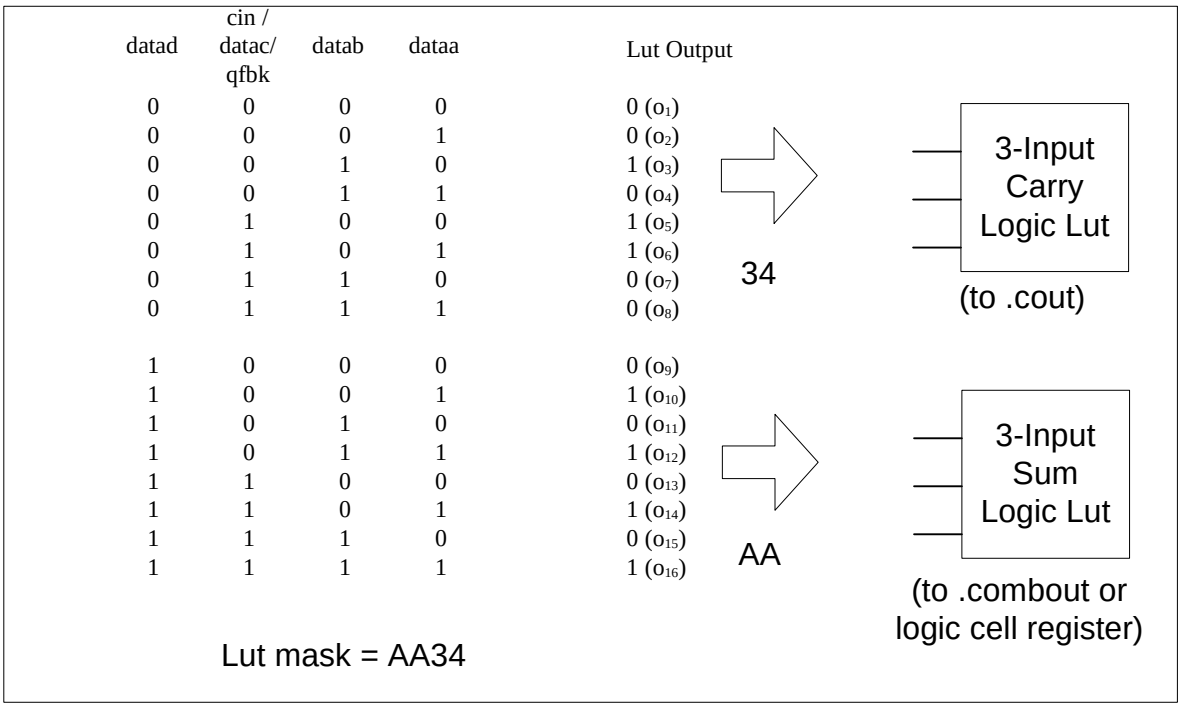


Figure 1 How a 16-bit LUT Mask is Divided into the Sum and Carry LUTs (arithmetic mode)

Care must be taken to use only the data signals that are valid for the selected logic cell mode. A LUT mask must be designated for each device primitive instantiation.

The first bit in a carry chain (an arithmetic mode cell with no .cin port connected) should set its lutmask such that .cin is a don't care unless the carry chain is using .inverta to create an adder-subtractor. If inverta is used, then inverta is fed to the carry input circuitry internally (to allow a carry-in of 1 for subtraction), so the truth table should be set appropriately.

<*created_from*> specifies the name of the register from which the registered output of this lcell was created. This setting is used to preserve assignments when fitter netlist optimizations are used. Fitter netlist optimizations include duplication of registers in which case it is vital that any timing assignments made to the original register also apply to the duplicate.

1.5 Logic Cell Polarities and Default Values

Table 2 describes the polarity and programmable inversion ability of all the logic cell inputs. Signals which can be programmably inverted in a logic cell can be provided in either polarity.

Table 2 -- Polarity of Logic Cell Inputs

Signal	*Polarity*	*Programmable Inversion*
.clk	Rising Edge	Yes
.dataa, .datab, .datad	Positive Logic	Yes (By lutmask manipulation)
.datac	Active High	Yes (by lutmask manipulation) when used as an input to the LUT. No when used as sync_data input, or async_data input.
.aclr	Active High	Yes
.aload	Active High	Yes
.sclr	Active High	Yes
.sload	Active High	Yes
.ena	Active High	Yes
.cin	Active High	No
.inverta	1 -> invert data, 0 -> no inversion	Yes
.regcascin	Active High	No

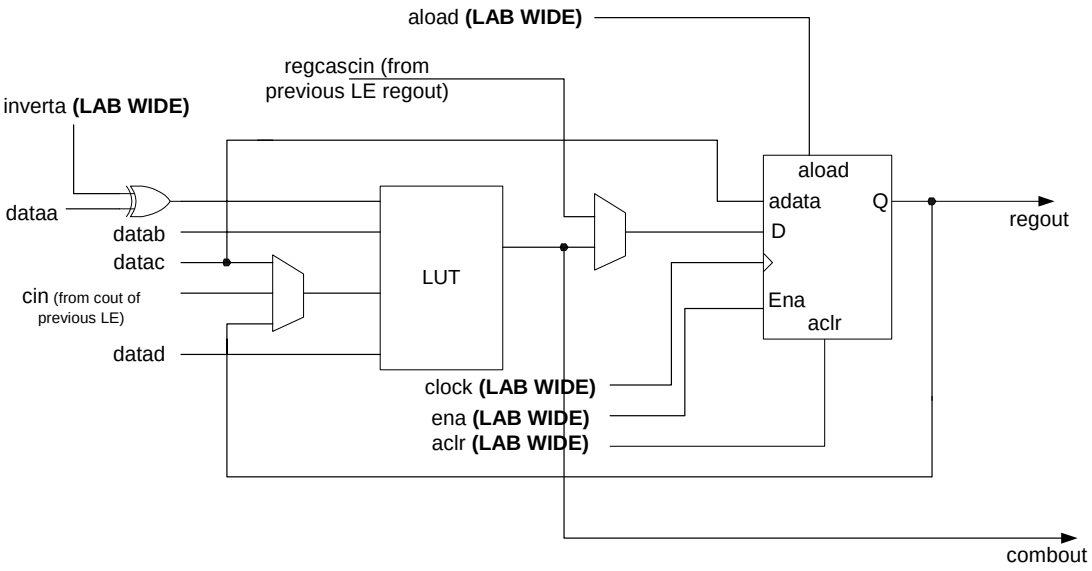
If not explicitly set in the device primitive instantiation, signals default to unconnected on the logic cell.

1.6 LE Register Setting Priority

There are several different signals that can cause the LE register to change its stored value. The priority of these signals is: aclr, aload, sclr, and sload. If more than one of these signals is active (logic 1), the highest priority signal determines the stored register state. If none of these signals are high on a given cycle, the normal (D) input to the register sets the stored register value at the rising clock edge. The D input to the register is driven by the LUT output or .regcascin, depending on the state of register_cascade_mode.

1.7 Logic Cell Mode Block Diagrams

Some of the input signals that can be connected to a logic cell are “LAB-wide” signals. These signals are shared between all the logic cells in a structure called a logic array block, or LAB. There are ten logic cells in a LAB. LAB-wide signals are identified in the diagrams below, since these signals must be treated with more care than other signals in order to avoid fitting problems. See the Stratix EDA FD for details on fitting issues due to LAB-wide signals.



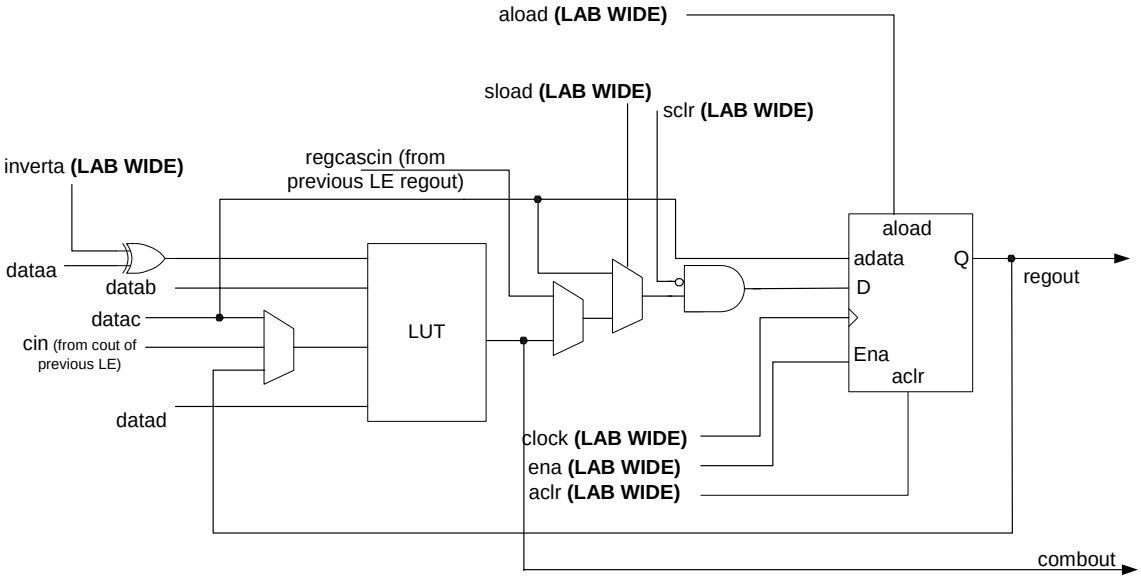
operation_mode = Normal

synch_mode = off

register_cascade_mode = {off, on}

sum_lutc_input = {datac, cin, qfbk}

Figure 2 Operation_mode = Normal, Synch_mode = Off



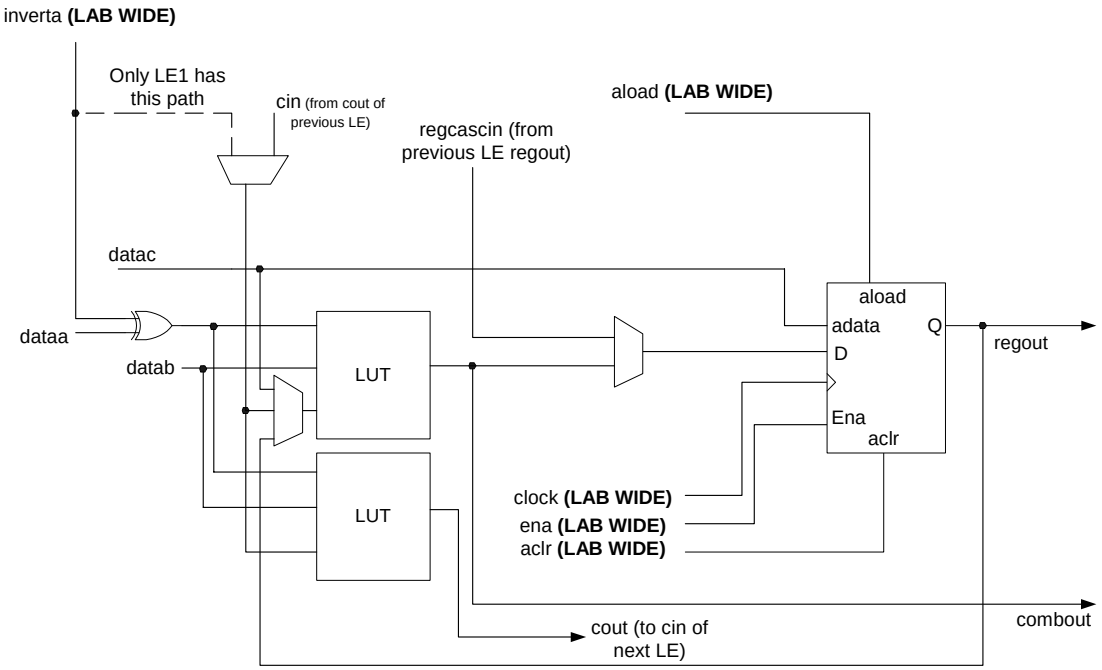
operation_mode = Normal

synch_mode = on

register_cascade_mode = {off, on}

sum_lutc_input = {datac, cin, qfbk}

Figure 3 Operation_mode = Normal, Synch_mode = On



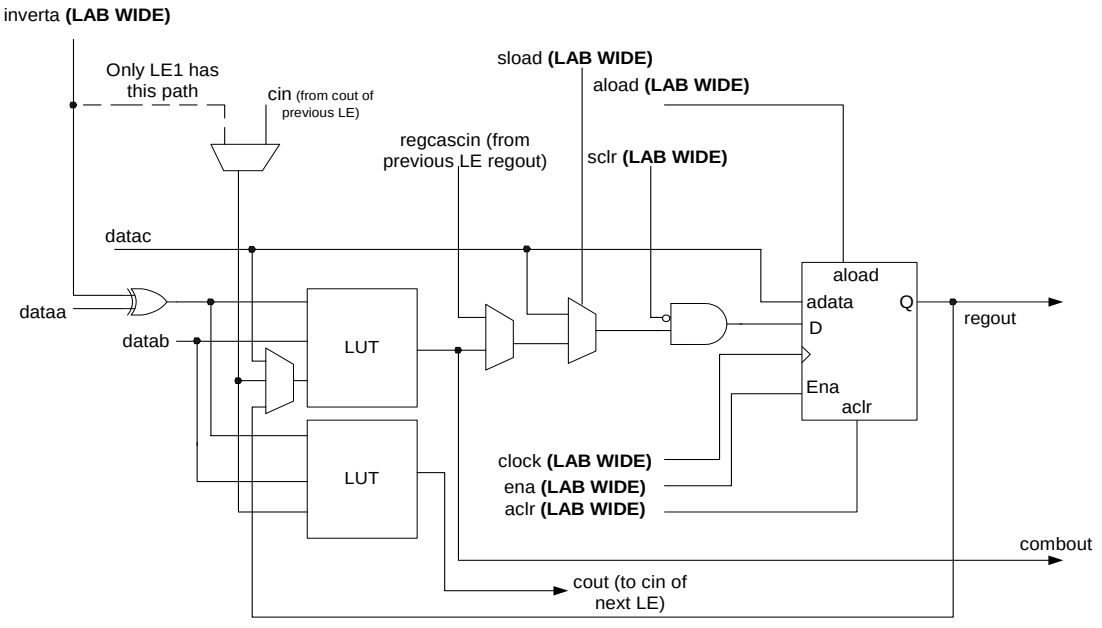
operation_mode = Arithmetic

synch_mode = off

register_cascade_mode = {off, on}

sum_lutc_input = {datac, cin, qfbk}

Figure 4 Operation_mode = Arithmetic, Synch_mode = off



operation_mode = Arithmetic

synch_mode = on

register_cascade_mode = {off, on}

sum_lutc_input = {datac, cin, qfbk}

Figure 5 Operation_mode = Arithmetic, Synch_mode = on

1.8 Emulating Packed registers

The synch enable mode can emulate APEX and Mercury style packed registers by connecting sload to VCC which means the lcell is constantly feeding datac into the D input of the register. Figure 6 shows the synch-enable mode emulating packed registers.

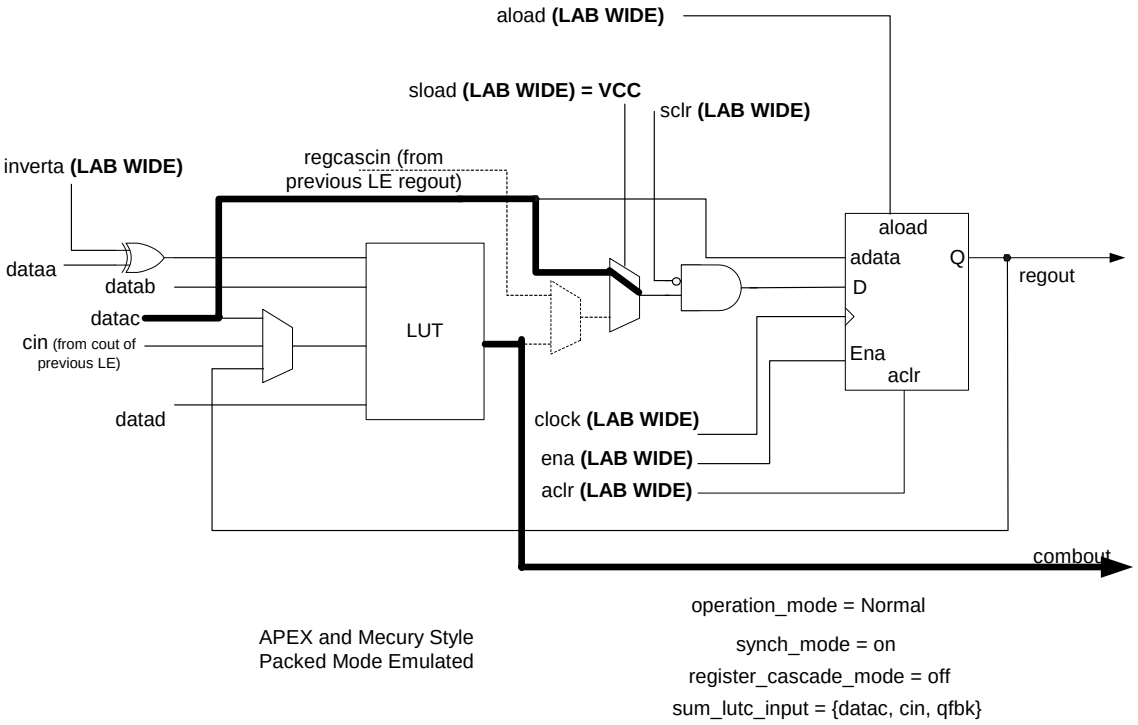


Figure 6 Operation Mode = Normal, Synch_mode = On, Packed register emulation

2. Stratix I/O Element

The Stratix I/O element is very similar to the APEX-II I/O element. The two main differences are:

(1) Stratix has a synchronous reset signal that APEX-II doesn't have.

(2) There is no peripheral bus in Stratix. All signals can reach an I/O element via general routing.

2.1 I/O Primitive

```
stratix_io <I/O name>
(
      .datain(<output source>),
      .ddiodatain(<DDIO output source >),
      .oe(<oe source>),
      .outclk(<output clock source>),
      .outclkena(<output clock-enable source>),
      .inclk(<input register clock source>),
      .inclkena(<input register clock-enable source>),
      .areset(<asynchronous set/reset source>),
      .sreset(<synchronous set/reset source>),
      .padio(<pad I/O source/output>),

      .combout(<combinational output>),
      .regout(<input register output>),
      .ddioregout(<DDIO register output>),
      .dff_data_out(<internal register output>),
      .dff_ddio_data_out(<internal ddio register output>),
      .dff_oe(<internal output enable register output>)
);
defparam <I/O name>.operation_mode = <operation mode>;
defparam <I/O name>.ddio_mode = <DDIO mode>;
defparam <I/O name>.open_drain_output = <open drain mode>;
defparam <I/O name>.bus_hold = <bus hold mode>;

defparam <I/O name>.output_register_mode = <output register mode>;
defparam <I/O name>.output_async_reset = <output register
      asynchronous reset mode>;
defparam <I/O name>.output_power_up = <output register power up
      mode>;
defparam <I/O name>.output_sync_reset = <output register
      synchronous reset mode>;
defparam <I/O name>.tie_off_output_clock_enable = <tie off output
      clock enable>;

defparam <I/O name>.oe_register_mode = <oe register mode>;
defparam <I/O name>.oe_async_reset = <oe register asynchronous
      reset mode>;
```

```
defparam <I/O name>.oe_power_up = <oe register power up mode>;
defparam <I/O name>.oe_sync_reset = <oe register synchronous reset
      mode>;
defparam <I/O name>.tie_off_oe_clock_enable = <tie off oe clock
      enable>;

defparam <I/O name>.input_register_mode = <input register mode>;
defparam <I/O name>.input_async_reset = <input register
      asynchronous reset mode>;
defparam <I/O name>.input_power_up = <input register power up
      mode>;
defparam <I/O name>.input_sync_reset = <input register synchronous
      reset mode>;

defparam <I/O name>.extend_oe_disable = <extend OE disable>;
defparam <I/O name>.created_from = <created from register name>;
```

2.2 I/O Input Ports

<*I/O name*> is the unique identifier for the I/O element. This is any identifier name which is legal for the given description language (e.g. Verilog, VHDL, AHDL, etc.). *This field is required.*

.**datain (**<*output source*>**)** is the data input to the I/O element.

.ddiodatain(<*DDIO output source*>**)** is the second data input to the I/O element when it is in DDIO "output" or "bidir" mode.

.oe(<*oe source*>**)** is the output enable signal for the I/O element. This should not be specified when the I/O element is in "input" mode. It defaults to VCC (permanently enabled) in output mode, and it should not be specifically set to VCC.

.outclk(<*output clock source*>**)** designates the output-clock input to the I/O element, which drives the clock input on the output and oe registers. If this signal is used, then at least one of the output and oe registers must be in "register" mode.

.outclkena(<*output clock-enable source*>**)** designates the clock-enable signal for the output and oe registers in the I/O element. Only allowed when the .outclk port is specified and the output or oe is in "register" mode.

.inclk(<*input clock source*>**)** designates the clock input to the input register(s) (there are two input registers used when in DDIO mode) in the I/O element. Only allowed when .regout is connected.

.inclkena(<*input clock-enable source*>**)** designates the clock-enable signal for the input register in the I/O element. Only allowed when the .inclk port on the logic element is specified and **.regout** is connected.

.areset(<*asynchronous set/reset source*>**)** designates the asynchronous set/reset signal for the I/O element. Each set of registers (input, output and OE) in the I/O element can independently choose to use the .areset signal as either a clear or a preset or neither.

Only allowed when at least one of the .outclk or .inclk ports on the logic element is specified, and when at least one of the input, output, or oe register's asynchronous reset mode is "preset" or "clear."

.sreset(<*synchronous set/reset source*>**)** designates the synchronous set/reset signal for the I/O element. Each set of registers (input, output and OE) in the I/O element can independently choose to use the .sreset signal as either a clear, a preset or neither. Only allowed when at least one of the .outclk or .inclk ports on the logic element is specified, and when at least one of the input, output or oe register's synchronous reset mode is "preset" or "clear."

2.3 I/O Output Ports

.combout(<*combinational output*>**)** is the combinatorial output of the I/O element. It always feeds directly from the .padio.

.regout(<*registered output*>**)** is the registered output of the I/O element. Only allowed when the .inclk port on the logic element is specified. It is always sourced from the input register.

.ddioregout(<*DDIO registered output*>**)** is the second registered output of the I/O element when in DDIO mode. Only allowed when the .inclk port on the logic element is specified and the ddio_mode is "input" or "bidir".

.dff_data_out(<*internal register output* >**)** specifies the name of the packed register when register packing occurs. This port should always be dangling. The only purpose is to save the register name such that timing assignments made to that register are not lost when a packing operation occurs.

.dff_ddio_data_out(<*internal DDIO register output* >**)** specifies the name of the packed register when ddio register packing occurs. This port should always be dangling. The only purpose is to save the register name such that timing assignments made to that register are not lost when a packing operation occurs.

.dff_oe(<*internal OE register output*>**)** specifies the name of the packed register when output enable register packing occurs. This port should always be dangling. The only purpose is to save the register name such that timing assignments made to that register are not lost when a packing operation occurs.

2.4 I/O Bidirectional Ports

.padio(<*pad I/O source/output*>**)** represents the physical pad of the I/O element. A bi-directional port, this signal should be connected directly to module inputs and outputs. Logic feeding out between these ports and the top-level module outputs is illegal.

Example 1 demonstrates how the .padio signal is used to connect up to input, output and bidir signals.

Example 1 -- Stratix .padio Example

```
module foo ( input_signal, output_signal, bidir_signal )
```

```
            input input_signal;
            output output_signal;
            inout bidir_signal;
            stratix_io input_io (.padio(input_signal), ...);
                  defparam input_io.operation_mode = “input”;
                  ...
            stratix_io bidir_io (.padio(bidir_signal), ...)
                  defparam bidir_io.operation_mode = “bidir”;
                  ...
            stratix_io output_io (.padio(output_signal), ...)
                  defparam output_io.operation_mode = “output”;
                  ...
      end module
```

2.5 I/O Modes

<*operation_mode*> is one of *{input, output, bidir}*. Determines the directionality of the I/O element. *This field is required.*

<*ddio_mode*> is one of *{input, output, bidir or none}*. Determines the functionality of the DDIO circuitry in the I/O element. This field is optional and defaults to none.

If the DDIO mode is “input” or “bidir” then the I/O element is setup to receive data on both clock edges, but pass it to the core only on the rising edge of the clock. The following must be true:

- The .inclk port must be specified.
- The .combout port cannot be connected.
- The I/O element can be in “input” or “bidir” *operation_mode*.
- The .ddioregout port passes the data received on the falling edge of the input clock cycle and the .regout port passes the data received on the rising edge of the input clock cycle.

If the DDIO mode is “output” or “bidir” then the I/O element is setup to transmit data on both clock edges, but the data is latched from the core only on the rising edge of the clock. The following must be true:

- The .outclk port must be specified.
- The *output register mode* must be “register”.
- The I/O element can be in “output” or “bidir” *operation_mode*.
- On the rising edge of the output clock cycle, the .padio port transmits the data that was received at the .datain port. On the falling edge of the output clock cycle, the .padio port transmits the data that was received at the .ddiodatain port on the rising edge of the output clock cycle.

Note that it is possible for an I/O element in *operation_mode* = “bidir” to use DDIO on its output, its input, or both. For example, an I/O with *operation_mode* = “bidir” and

ddio_mode = “input” uses the standard output circuitry to send one value per clock, and the DDIO input circuitry to receive two values per clock.

<*open_drain_mode*> is one of *{true, false}*. This field is optional and defaults to *false*. This field can only be set to *true* when the I/O is in output or bidir mode but not input mode.

<*bus_hold_mode*> is one of *{true, false}*. This field is optional and defaults to *false*.

<*output_register_mode*> is one of *{register or none}*. This field is optional, and defaults to none. This determines if the .datain signal should be registered or not. The .outclk port is required if this mode is *register*. If the *ddio_mode* is *output* or *bidir*, then this mode must be *register*.

<*output_async_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .areset port clears, presets, or has no effect on the output register(s). The .areset port is required if this mode is *clear* or *preset*.

<*output_power_up*> is one of *{high, low}* and describes the power-up condition of the output register(s). This field is optional and defaults to low.

<*output_sync_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .sreset port clears, presets, or has no effect on the output register(s). The .sreset port is required if this mode is *clear* or *preset*.

<*tie off output clock enable*> is one of *{true, false}* and determines if the clock enable for the output register, controlled by **.outclkena**, should be tied off (has no effect on the register). This field is optional and defaults to false.

<*oe_register_mode*> is one of *{register or none}*. This field is optional, and defaults to none. This determines if the .oe signal (output enable source) is registered or not. The .outclk port is required if this mode is *register*.

<*oe_async_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .areset port clears, presets, or has no effect on the oe register. The .areset port is required if this mode is *clear* or *preset*.

<*oe_power_up*> is one of *{high, low}* and describes the power-up condition of the oe register. This field is optional and defaults to low.

<*oe_sync_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .sreset port clears, presets, or has no effect on the oe register. The .sreset port is required if this mode is *clear* or *preset*.

<*tie off oe clock enable*> is one of *{true, false}* and determines if the clock enable for the oe register, controlled by **.outclkena** should be tied off (has no effect on the register). This field is optional and defaults to false.

<*input_register_mode*> is one of *{register or none}*. This field is optional, and defaults to none. This field should be set to *register* if and only if this I/O element uses its input register(s), i.e., the .regout port(the regout and ddioregout ports). The .inclk port is required if this mode is *register*. If the *ddio_mode* is *input* or *bidir*, then this mode must be *register*.

<*input_async_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .areset port clears, presets, or has no effect on the input register(s). The .areset port is required if this mode is *clear* or *preset*.

<*input_power_up*> is one of *{high, low}* and described the power-up condition of the input register(s). This field is optional and defaults to low.

<*input_sync_reset*> is one of *{clear, preset or none}*. This field is optional, and defaults to none. This determines if the .sreset port clears, presets, or has no effect on the input register(s). The .sreset port is required if this mode is *clear* or *preset*.

<*extend OE disable*> is one of *{true, false}* and describes whether the second OE register should be used. When the second OE register is used, the output drive is held at high impedance for an extra ½ clock cycle when OE goes low. This field is optional and defaults to false.

Note: An I/O register can use only one of asynchronous clear or preset. If an I/O register uses asynchronous clear the power-up state must be low, if an I/O register uses asynchronous preset the power-up state must be high. If neither asynchronous preset nor clear is used the power-up state can be high or low.

An IO register can use both an asynchronous present or clear and a synchronous preset or clear. The setting of the synchronous preset or clear has no effect on the power-up state of the register.

<*created_from*> specifies the name of the register from which the registered output of this IO was created. When register packing occurs on a register that drives multiple outputs, the register will be duplicated and packed into each of the outputs. This setting allows the original register name to be saved since each duplicate will be assigned a different name.

2.6 I/O Polarities and Default Values

Table 3 – Polarity of I/O Inputs

Signal	*Polarity*	*Programmable Inversion*
.datain	--	Yes
.ddiodatain	--	Yes
.oe	Active high	Yes
.outclk	Rising edge	Yes
.outclkena	Active high	Yes
.inclk	Rising edge	Yes
.inclkena	Active high	Yes
.areset	Active high	Yes
.sreset	Active high	Yes

If not explicitly set in the device primitive instantiation, signals default to unconnected on the I/O element. Output enable will default to GND when the I/O element is in “input” mode, and VCC when it is in “output” mode. It should be left unconnected in “input” mode, and it should not be set to VCC in “output “ mode.

All Stratix I/O element inputs have programmable inversion, and hence can be provided in either polarity.

2.7 Basic I/O Mode Diagrams

The diagrams below show the basic configurations of the I/O element, showing all possible registers in each configuration. OE and output registers may be removed by changing the appropriate register mode to “none”, while the input register is bypassed by using the .combout port instead of the .regout port.

Any register can use one of asynchronous preset or clear (but not both) by setting the register’s asynchronous reset mode. The asynchronous preset or clear is fed by the .areset port. Similarly, any register can use one of synchronous preset or clear by setting the register’s synchronous reset mode. The synchronous preset or clear is fed by the .sreset port.

Any diagram showing a TRI primitive may have an OPNDRN primitive fed by, or in place of, the TRI primitve by setting open_drain_output = “true.”

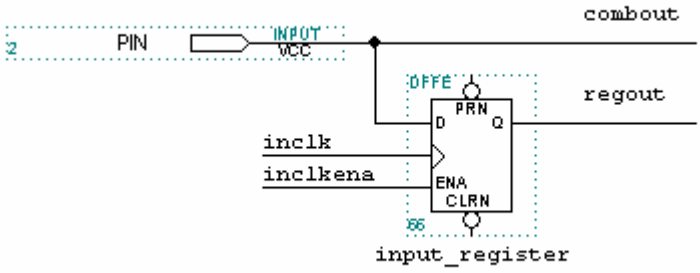


Figure 7: operation_mode = ”input”; either or both of **.combout** and **.regout** can be used. If .**regout** is used, input_register_mode must be set to “register”.

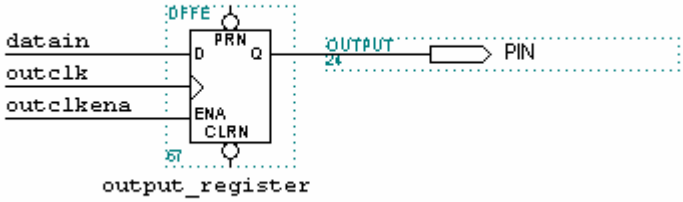


Figure 8: operation_mode = “output”; output_register_mode = “register.” This is the non tri-stated version of the output mode (.oe is disconnected).

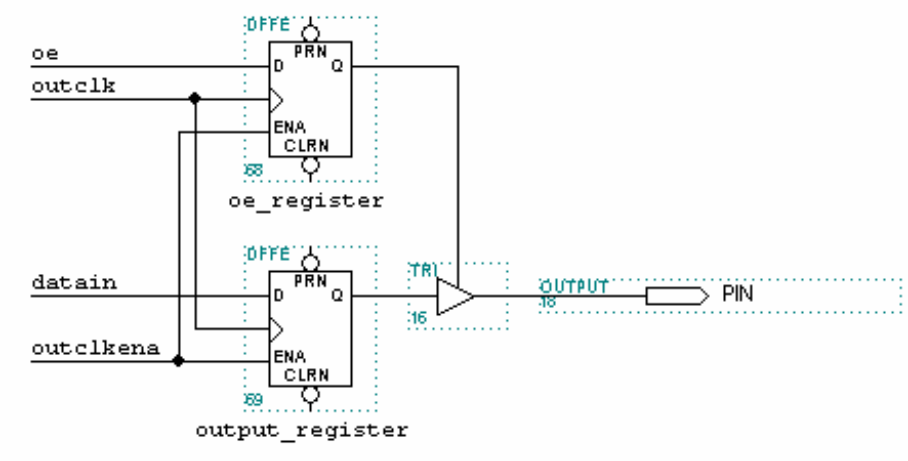


Figure 9: operation_mode = “output”; output_register_mode = “register”; oe_register_mode = “register.” This is the tri-state version of the output mode (**.oe** is connected).

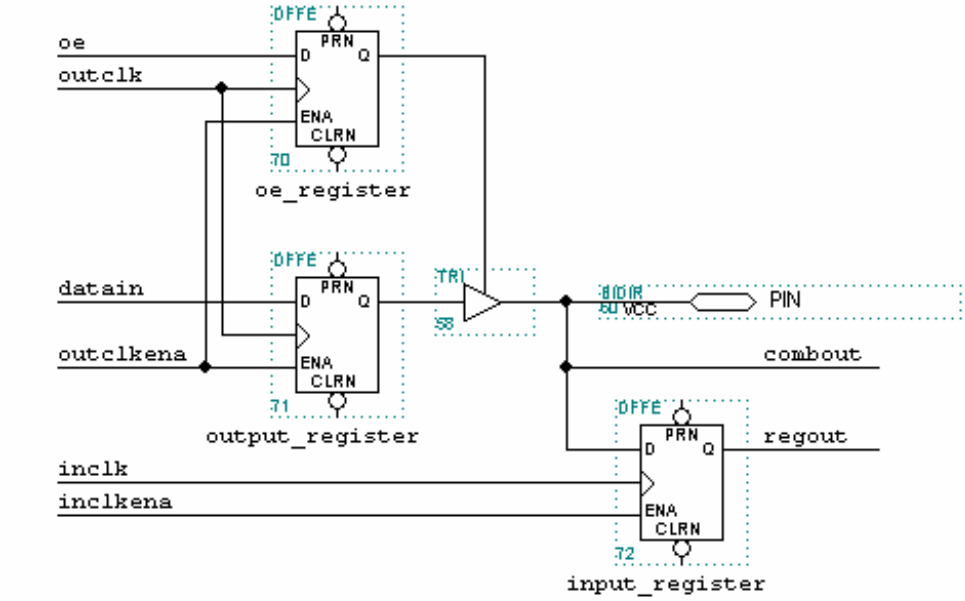


Figure 10: operation_mode = “bidir”; output_register_mode = “register”; oe_register_mode = “register”; either or both of .**combout** and .**regout** can be used. If .**regout** is used, input_register_mode must be set to “register”.

Figure 11: operation_mode = “output”; output_register_mode = “register”; ddio_mode = “output.”

2.8 Stratix DLL Extensions

Please see stratix_dll_wysuser_doc.doc for details on the DLL element, plus enhancements in the I/O element to support it.