LCELL WYSIWYG Description for the Cyclone II Architecture

Version 1.2

May 27, 2005

By

Altera Corporation

Table of Contents:

1. OVERVIEW 2

2. COMBINATIONAL LOGIC CELL PRIMITIVE 2

2.1 Combinational Logic Cell Input Ports 2
2.2 Combinational Logic Cell Output Ports 3
2.3 Combinational Logic Cell Modes 3
2.4 Combinational Logic Cell Block Diagram 3
2.5 Combinational Logic Cell Usage 4
2.5.1 Normal Mode 4
2.5.2 Arithmetic Mode 5
2.6 Interpreting the LUT Mask 6
2.7 Combinational Logic Cell Polarities and Default Values 8

3. LOGIC CELL REGISTER PRIMITIVE 8

3.1 Register Logic Cell Input Ports 8
3.2 Register Logic Cell Output Ports 9
3.3 Register Logic Cell Modes 9
3.4 Register Logic Cell Polarities and Default Values 9
3.5 Register Logic Cell Control Signal Priority 9
3.6 Register Logic Cell Control Signal Choices 10
3.6.1 Clocks & clock-enables 10
3.6.2 Asynchronous clear 10
3.6.3 Synchronous load & clear 10
3.7 Register Logic Cell Block Diagram 11

4. CYCLONE II LOGIC CELL DESCRIPTION 11

1. Overview

This document describes the LCELL WYSIWYG and atom support for the Cyclone II architecture. The Cyclone II logic cell is a cost-optimized version of the Cyclone logic cell. The main differences are:

- inverta is removed
- The register does not have asynchronous load
- The carry circuitry has been simplified – there are no internal carry signals

Similar to the Stratix® II family, two primitives will represent the basic logic cell. One will represent the combinational block, and another will represent the register block. Each logic cell in the Cyclone II device family contains one combinational block and one register block.

We also use the same WYSIWYG philosophy as in the Stratix II family. Basically every input pin on a WYSIWYG must be able to be connected, and functionality must be directly indicated by parameters. Refer to the document *stratixii_le_wys.doc* for more details.

Refer to the document *stratix_wysuser_doc.doc* for a description of the Cyclone and Stratix logic cells.

Section 2 describes the combinational WYSIWYG block.

Section 3 describes the register WYSIWYG block.

Section 4 describes how they are connected in a logic cell.

2. Combinational Logic Cell Primitive

```
cycloneii_lcell_comb <lcell_name>
(
      .dataa(<data_a source>),
      .datab(<data_b source>),
      .datac(<data_c source>),
      .datad(<data_d source>),
      .cin(<carry in source>),

      .combout(<combinational output>),
      .cout(<carry output>)
);
defparam <lcell_name>.lut_mask = <lut mask>;
defparam <lcell_name>.sum_lutc_input = <sum lut input choice>;
```

2.1 Combinational 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.

.**dataa(**<*data_a source*>**), … .datad(**<*data_d source*>**):** are the LUT inputs for the logic cell.

Different LUT inputs have different speeds. The fitter will choose the final connections.

.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 GND, or inverted, but it can be directly tied to VCC. See section 2.5.2 for a description of how this port is used.

2.2 Combinational Logic Cell Output Ports

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

.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. See section 2.5.2 for a description of how this port is used.

2.3 Combinational Logic Cell Modes

<*sum_lutc_input*> is one of *{datac, cin}*. 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

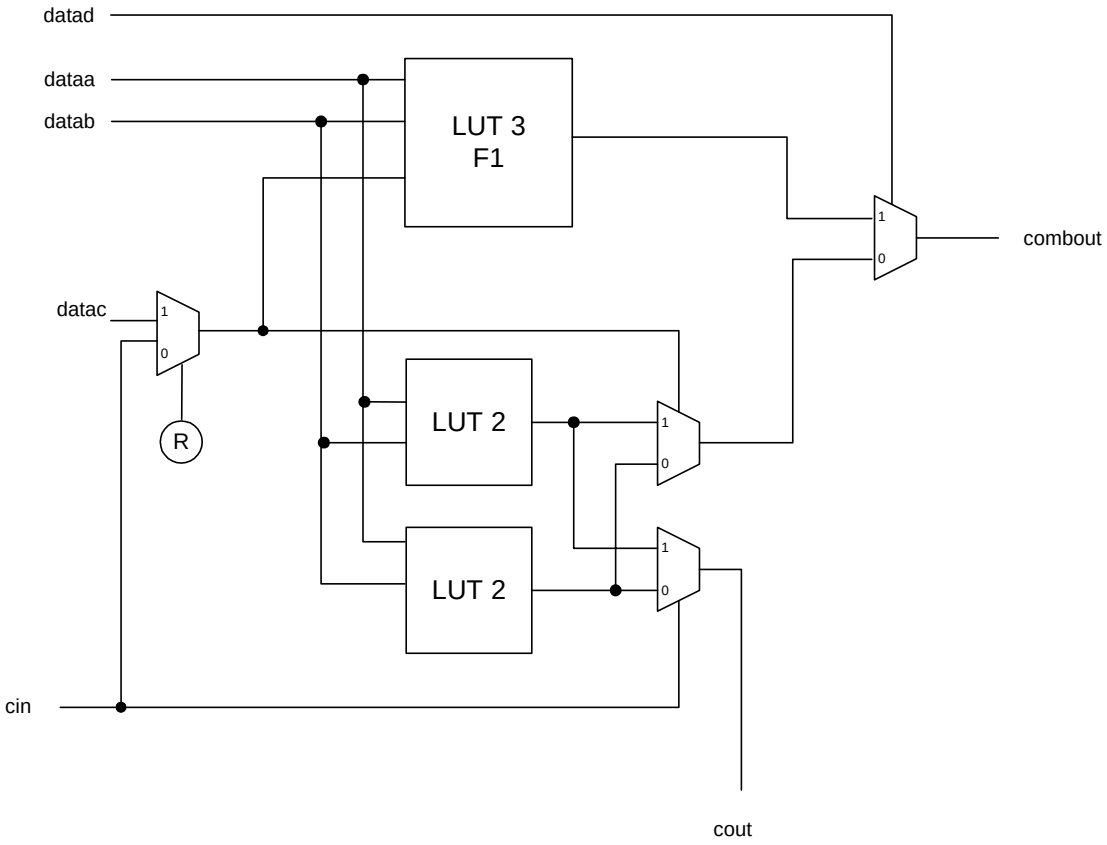
If the mode is cin, then the upper LUT (or the full 4-input LUT) is fed by the cin signal

This field is optional. It defaults to cin if the .cin port is connected; otherwise, it defaults to datac.

2.4 Combinational Logic Cell Block Diagram

Figure 1 shows the full functionality of the Cyclone II combinational logic cell. It has five inputs and two outputs. Its most common usages are described in section 2.5.

Figure 1 -- Combinational Logic Cell



2.5 Combinational Logic Cell Usage

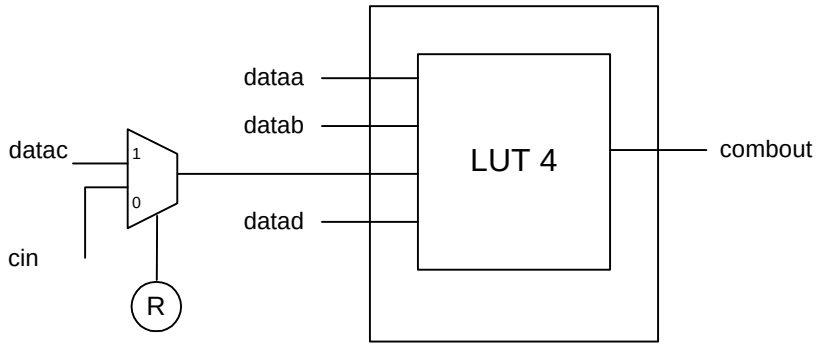
The combinational logic cell can be in normal or arithmetic mode.

2.5.1 Normal Mode

Normal mode implements the function of the 4 data inputs described by the LUT mask.

Combout = F(dataa, datab, datac or cin, datad)

Figure 2 -- Normal Mode of Combinational Logic Cell



2.5.2 Arithmetic Mode

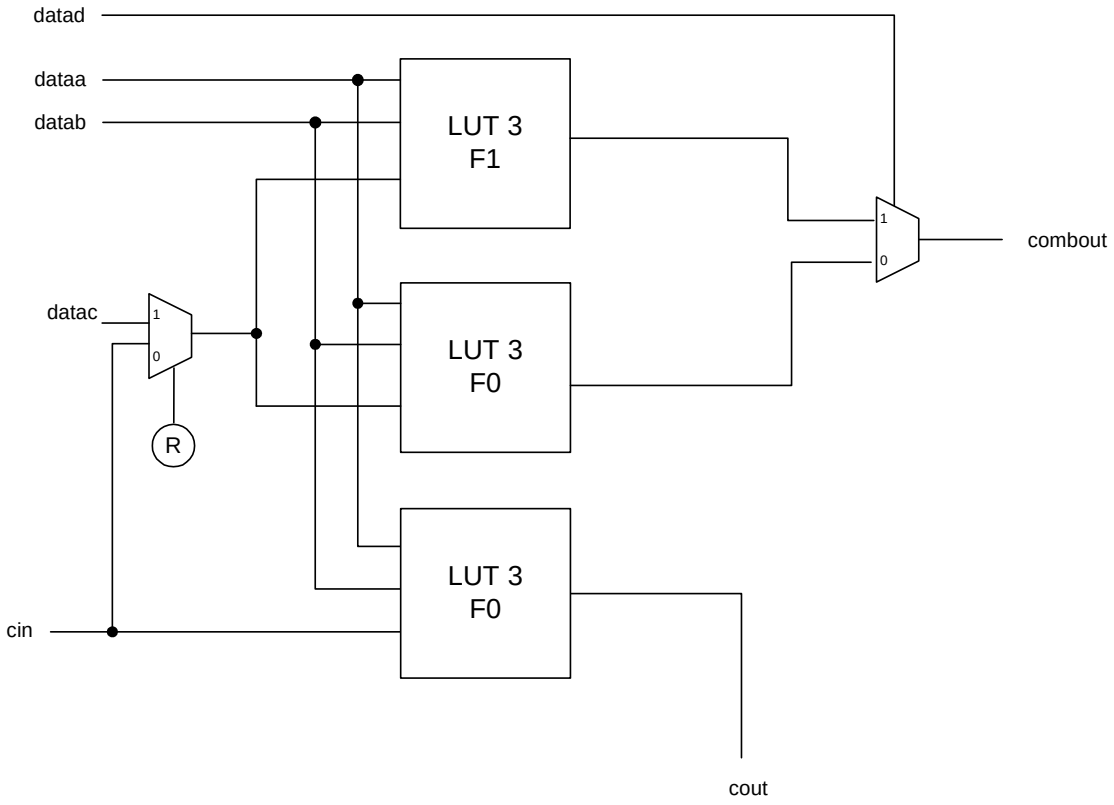
In this mode, in addition to the regular look-up table output (i.e. combout), there is a second look-up table for computing the carryout. Note that the two look-up tables are not completely independent. In fact, when you restrict datad to 0 in the combout function F, you get the carryout function.

Normally .datad is connected to VCC, and the regular look-up table output computes the sum as a function of .dataa, .datab, and .datac or .cin depending on the value of sum_lutc_input. The second look-up table computes the carry signal as a function of .dataa, .datab, and .cin.

Please see section 2.6 for more information on how to interpret the LUT mask.

Combout	= F1(dataa, datab, datac or cin) & datad # F0(dataa, datab, datac or cin) & ! datad
Cout	= F0(dataa, datab, cin)

Figure 3 -- Arithmetic Mode of Combinational Logic Cell



2.6 Interpreting the LUT Mask

<*lut_mask*> designates the content of the LUT(s) in the logic cell. It can be specified as a 4 digit hexadecimal number, or a sixteen digit binary number, and represents the 16 bits of data in the LUT.

Note: The Quartus II compiler can take LUT masks in many forms, including hex or binary strings, or constants (which could also be in either hex radix or binary radix). In Verilog, for best compatibility with the simulation models and 3rd-party tools, the preferred way to assign a value to lut_mask would be as a constant (either hex radix or binary radix). For example, use:

```
defparam instance_name.lut_mask = 16'hFF00;    or
defparam instance_name.lut_mask = 16'b1111111100000000;
```

instead of

```
defparam instance_name.lut_mask = "FF00";    or
defparam instance_name.lut_mask = "1111111100000000";
```

The LUT mask uses positive logic, and is formatted as in Table 1, which shows how the data is ordered when the bits are ordered as: o_{15} o_{14} o_{13} … o_2 o_1 o_0

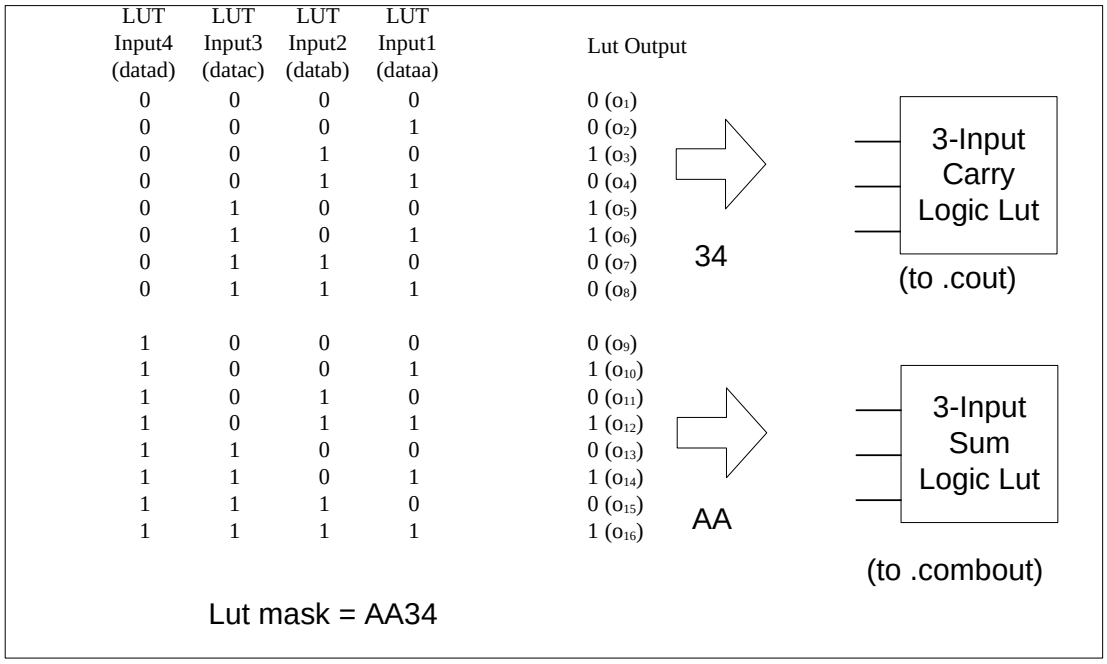
Table 1 -- LUT Mask Data Ordering when in normal mode.

LUT Input4 (datad)	*LUT Input3 (datac)*	*LUT Input2 (datab)*	*LUT Input1 (dataa)*	*LUT Output*
0	0	0	0	O_0
0	0	0	1	O_1
0	0	1	0	O_2
0	0	1	1	O_3
0	1	0	0	O_4
0	1	0	1	O_5
0	1	1	0	O_6
0	1	1	1	O_7
…	…	…	…	…
1	1	1	1	O_{15}

If one pictures a four-input LUT as a tree of 2to1 MUXes, then *LUT Input1* controls the first stage, and *LUT Input4* controls the last stage, whose output is *LUT Output*.

By setting *LUT Input4* = 0 or 1, the 16-bit LUT mask can be naturally divided into two 8-bit regions, F0, and F1, each of which represents one three-input LUT. This is useful when the cell is in arithmetic mode. In this case, the 8 *LUT Output* bits corresponding to *LUT Input4* = 0 go to the F0 function and the 8 *LUT Output* bits corresponding to *LUT Input4* = 1 go to the F1 function. Figure 4 shows how the LUT mask AA34 is mapped onto the F0 and F1 LUTs.

Figure 4 -- How a 16-bit LUT mask is divided into the F0 & F1 LUTs



Care must be taken when generating LUT masks. A LUT mask must be designated for each device primitive instantiation.

2.7 Combinational 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 Combinational Logic Cell Inputs

Signal	*Polarity*	*Programmable Inversion*
.dataa, .datab, .datac, .datad	Positive Logic	Yes (By LUT mask manipulation)
.cin	Active High	No

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

3. Logic Cell Register Primitive

```
cycloneii_lcell_ff <lcell_name>
(
      .datain(<register data source>),
      .clk(<clock source>),
      .aclr(<asynchronous clear source>),
      .sclr(<synchronous clear source>),
      .sload(<synchronous load source>),
      .sdata(<register synch data source>),
      .ena(<clock enable source>),

      .regout(<registered output0>)
);
```

3.1 Register 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.

.datain(<*register data source*>**):** designates the data input to the register.

.clk(<*clock source*>**):** designates the clock input to the register. This port is required if any of the following ports are used: .datain, .sclr, .sload, .sdata, .ena. It defaults to GND if unspecified.

.aclr(<*asynchronous clear source*>**)** designates the asynchronous clear signal for the logic cell. Defaults to GND when not specified.

.sclr(<*synchronous clear source*>**),** designates the synchronous clear signal for the logic cell. Defaults to GND if unspecified. Only one sclr value is allowed in a single logic array block (LAB), 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.

.sload(<*synchronous load source*>**),** designates the synchronous load signal for the logic cell. When .sload is high then the register is synchronously loaded from .sdata. Hence .sdata must be connected if .sload is connected. Defaults to GND when unspecified. Only one

sload value is allowed in a single logic array block (LAB), so sloads should not be used for low-fanout signals.

.sdata(<*synchronous load data source*>**),** designates the synchronous data signal for the logic cell. When .sload is high then the register is synchronously loaded from .sdata. Hence .sdata must be connected if .sload is connected. The signal defaults to GND when unspecified.

.ena(<*clock enable source*>**),** designates the clock enable signal for the logic cell. Defaults to VCC when unspecified.

3.2 Register Logic Cell Output Ports

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

3.3 Register Logic Cell Modes

There are no modes.

3.4 Register Logic Cell Polarities and Default Values

Table 3 describes the polarity and programmable inversion ability of all the logic cell inputs. Signals which can be programably inverted in a logic cell can be provided in either polarity.

Table 3 -- Polarity of Register Logic Cell Inputs

Signal	*Polarity*	*Programmable Inversion*
.clk	Rising Edge	Yes
.datain	Positive Logic	No
.aclr	Active High	Yes
.sclr	Active High	Yes
.sload	Active High	Yes
.sdata	Positive Logic	No
.ena	Active High	Yes

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

3.5 Register Logic Cell Control Signal Priority

There are several different signals that can cause the register to change its stored value. If more than one of these signals is active (logic 1), the highest priority signal determines the stored register state. The priority of these signals is described in Table 4. If none of these signals are active on a given cycle, the normal datain input to the register sets the stored register value at the rising clock edge.

For example if aclr and sload were both asserted on the same clock cycle the aclr port would take precedence and clear the register since it is has the first priority.

Table 4 -- Register Logic Cell Control Signal Priority

Signal	*Priority*
.aclr	1
.ena	2
.sclr	3
.sload	4

3.6 Register Logic Cell Control Signal Choices

The register control signal flexibility in the Cyclone II architecture is approximately the same as the Cyclone architecture. Every LAB has the following register control signals available: 2 clocks, 2 clock-enables, 2 asynchronous clears, 1 synchronous clear, and 1 synchronous load.

Along with the global network there are 4 normal routing connections in the LAB for these 8 control signals.

3.6.1 Clocks & clock-enables

Same as in the Cyclone family, all combinations of the 2 clock-enable and 2 clock signals can be used to create 2 LAB clocks. Every register in the LAB may then choose from one of these 2 clocks. The clocks can be routed on the dedicated global clock network as well as using normal routing resources, while the clock-enables must be routed using normal routing resources. An always-enabled clock uses up one of the 2 LAB clock lines, but does not use one of the 4 normal routing paths for control signals to generate the VCC enable.

3.6.2 Asynchronous clear

Every LAB has 2 asynchronous clears. Any one of the two signals can be tied disabled at the LAB boundary for registers that do not need it. What this means is that a LAB may contain registers with at most 2 different combinations of asynchronous clears.

Same as in the Cyclone family, only one of the asynchronous clear signals could be routed using the global routing network; the other one must be routed using normal routing resources.

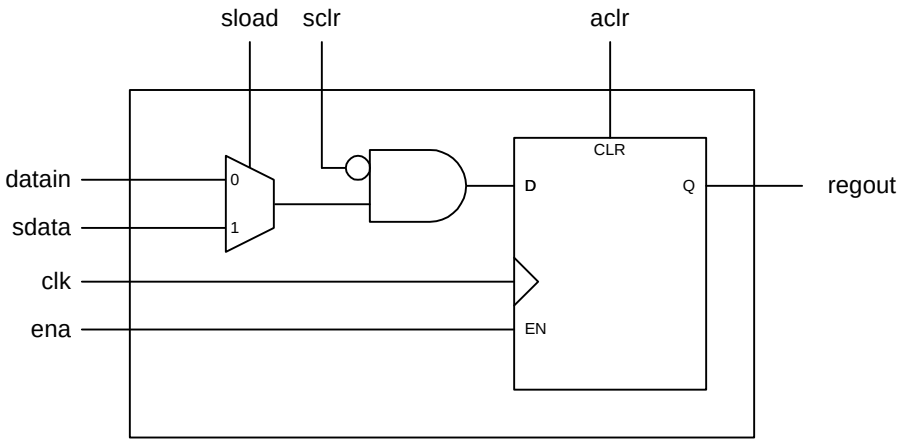
3.6.3 Synchronous load & clear

The synchronous load and clear restrictions are identical to the Cyclone architecture. There is 1 synchronous load and 1 synchronous clear signal available per LAB. Each register in a LAB can either use or not use the 2 synchronous signals as a group.

The synchronous load circuitry is also used for “lonely register” packing by connecting the LAB wide synchronous load to VCC which means the logic cell is constantly feeding datac into the D input of the register. Figure 8 of Section 4 illustrates how this is physically implemented. If a synchronous load signal is used in a LAB, “lonely registers” may not be placed in that LAB. Also if a synchronous clear signal is used in a LAB any “lonely register” placed in that LAB would have to use that synchronous clear signal.

3.7 Register Logic Cell Block Diagram

Figure 5 -- Register Logic Cell



4. Cyclone II Logic Cell Description

A Cyclone II logic cell is a grouping of one combinational and one registered logic cell.

Figure 6 shows a logical representation of a Cyclone II logic cell. This diagram shows clearly how the logic cell inputs and outputs are connected to the combinational and register logic cells and how they are connected to each other.

Figure 6 – Cyclone II Logic Cell Logical Representation

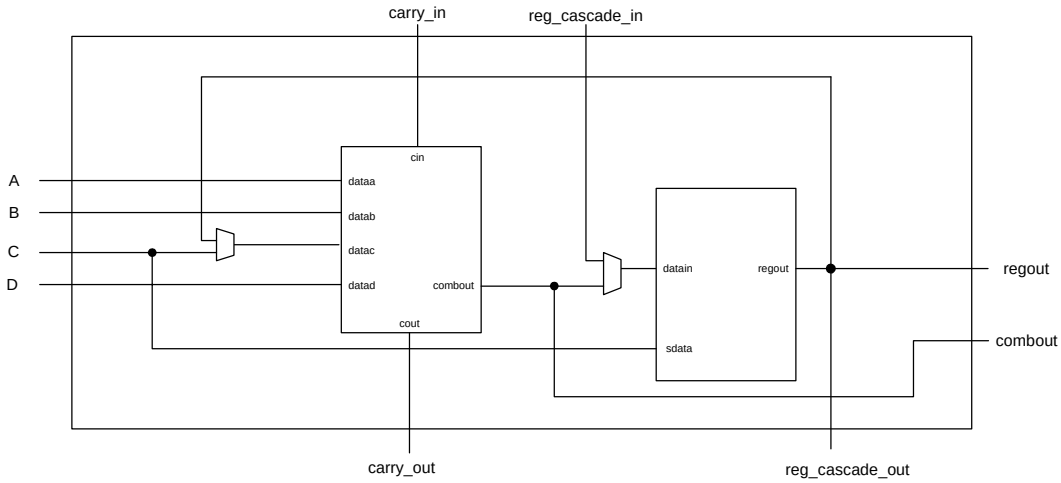


Figure 7 gives an accurate description of the physical implementation of a Cyclone II logic cell and Figure 8 illustrates the lonely register packing feature.

Figure 7 – Cyclone II Logic Cell Physical Representation

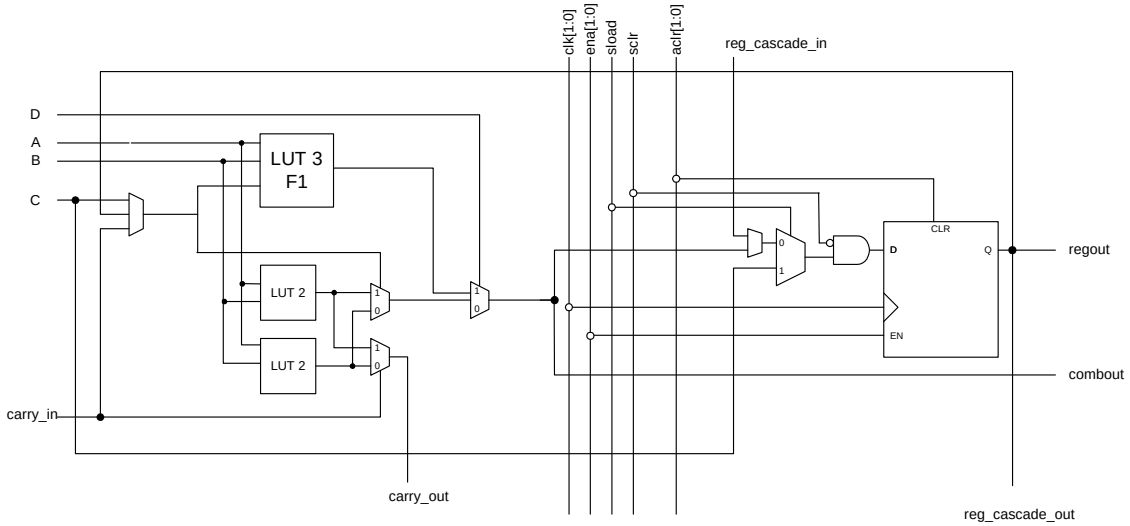


Figure 8 – Lonely Register Packing in Cyclone II Devices