LCELL WYSIWYG Description for the Stratix II Family

Version 1.1

May 27, 2005

By

Altera Corporation

Table of Contents:

1. Overview

This is a summary of the LCELL WYSIWYG and atom support for the Stratix II family. Unlike the Stratix family and all previous Altera architectures, two primitives will represent the basic logic cell. One will represent the combinational block and another will represent the register block.

The basic logic tile in the Stratix II device is called an ALM (Adaptive Logic Module) and can contain 2 combinational and 2 register blocks.

Section 2 describes the WYSIWYG philosophy.

Section 3 describes combinational WYSIWYG block.

Section 4 describes the register WYSIWYG block.

Section 5 describes how they are connected in an ALM.

2. WYSIWYG Philosophy

There is a strong desire to have the same Stratix II WYSIWYG netlist be legal for both simulation and design entry. In order to achieve this goal several rules must be followed in the definition of the WYSIWYG primitives.

- No functionality of the WYSIWYG primitive may depend on whether an input is connected or not. Third party simulators cannot use connectivity to determine functionality so all functionality must be directly indicated by parameters.
- Every input pin on a WYSIWYG primitive must be able to be connected. Third-party simulators require all module inputs to be driven by at least a default value. This means that pins illegal in some mode must have a legal default value that can be ignored. For example, in the Stratix architecture, connecting the cin port when you are not in arithmetic mode was illegal. In the Stratix II architecture, you would at least have to allow the connection of GND.

3. Combinational Logic Cell Primitive

```
stratixii_lcell_comb <lcell_name>
(
      .dataa(<data_a source>),
      .datab(<data_b source>),
      .datac(<data_c source>),
      .datad(<data_d source>),
      .datae(<data_e source>),
      .dataf(<data_f source>),
      .datag(<data_g source>),
      .cin(<carry in source>),
      .sharein(<shared function input source>),

      .combout(<combinational output>),
      .sumout(<arithmetic sum output>),
      .cout(<carry output>),
```

```
        .shareout(<shared function output>)
    );
    defparam <lcell_name>.lut_mask = <lut mask>;
    defparam <lcell_name>.shared_arith = <on, off>;
    defparam <lcell_name>.extended_lut = <on, off>;
```

3.1 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*>**), … .datag(**<*data_g source*>**):** are the LUT inputs for the logic cell.

Different LUT inputs have different speeds. The fitter will choose the final connections. See section 5 for details of how the ALM inputs connect to the comb logic cell's data inputs.

.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, or inverted, but unlike the Stratix architecture, it can be directly tied to GND to start an arithmetic chain. See section 3.5.2 for a description of how this port is used.

.sharein(<*shared function input source*>**)** is a direct input to the fast adder circuitry in this logic cell. It must be fed by the .shareout port of the same logic cell that provides the .cout signal for the .cin port. If .sharein is connected then .cin must be connected. This connection cannot be inverted or directly tied to VCC. It can be tied to GND to start a chain. See section 3.5.3 for a description of how this port is used.

The cin and sharein ports can be tied off to ground at either the start of a LAB or the mid-point of a LAB. In order to start a carry chain in any other position a dummy atom must be created to generate the first cout.

3.2 Combinational Logic Cell Output Ports

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

.sumout (<*arithmetic sum output*>**)** is the sum output of the dedicated adder in 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 3.5.2 for a description of how this port is used.

.shareout(<*shared function output*>**)** is the shared function output of the logic cell. Note that if .shareout is connected to another logic cell the signal must go to the logic cell's .sharein port. Also if the .shareout is used the .cout must also be used and connected to the same logic cell as the .shareout signal. See section 3.5.3 for a description of how this port is used.

3.3 Combinational Logic Cell Modes

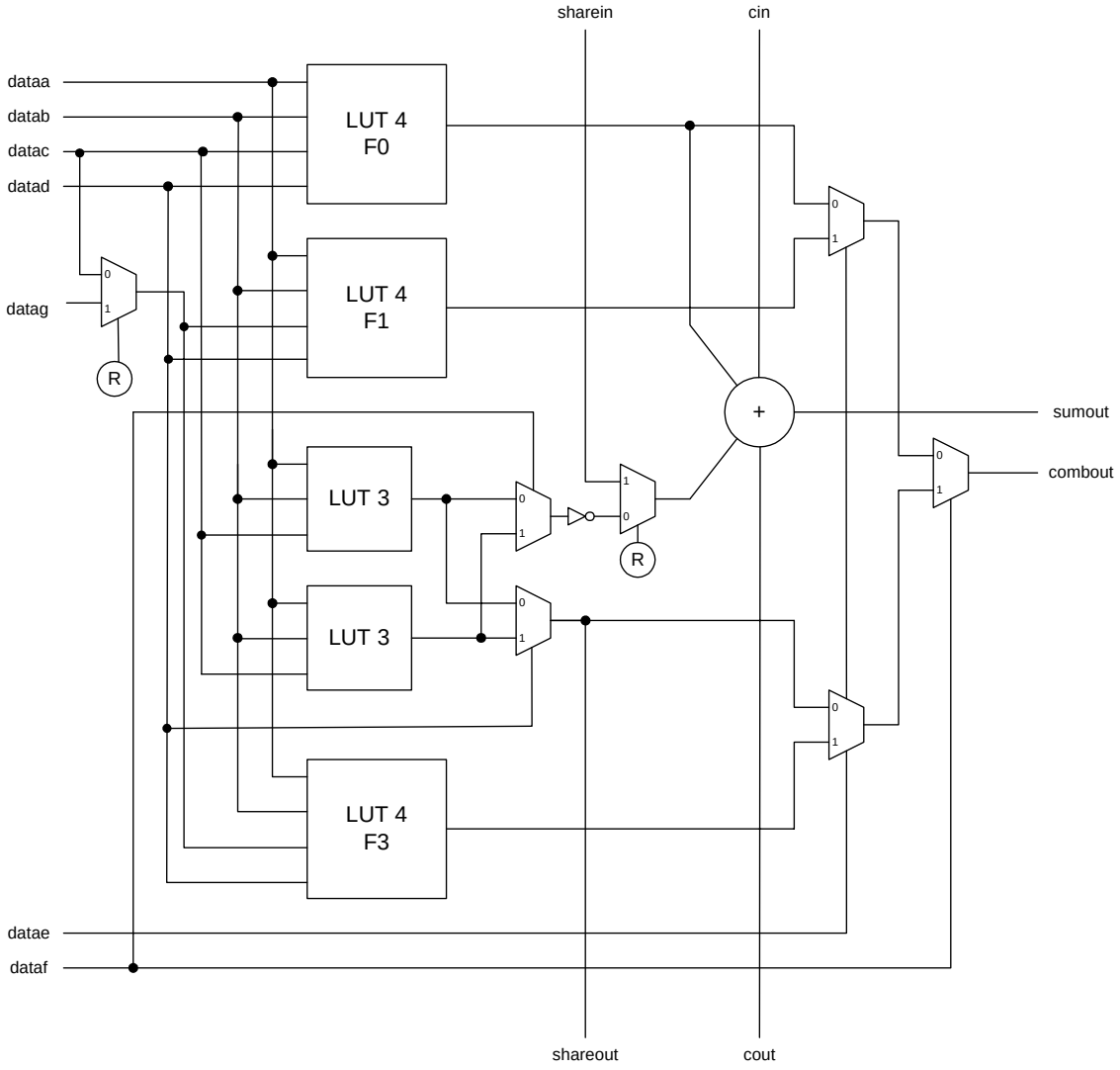
.shared_arith (<*on, off*>**)** controls where the second data input to the dedicated adder comes from. When set to on it comes from the sharein port. The sharein port must be connected to a valid shareout port or GND when in shared arithmetic mode. When this parameter is set to off the second data in signal to the dedicated adder comes from the F2 LUT. The default value for this parameter is off. See sections 3.5.2 and 3.5.3 for descriptions of the arithmetic and shared arithmetic functions.

.extended_lut (<*on, off*>**)** controls whether the combinational output represents a 7-input function. When set to on, datag gets used in addition to dataa through dataf, creating some 7-input functions. When this parameter is set to off the combinational output represents arbitrary functions of 6-inputs. The default value for this parameter is off. See section 3.6.1 for more details on this mode.

3.4 Combinational Logic Cell Block Diagram

Figure 1 shows the full functionality of the Stratix II combinational logic cell. It has nine inputs and four outputs. Its most common usages are described in section 3.5.

Figure 1 -- Combinational Logic Cell



3.5 Common Combinational Logic Cell Usage

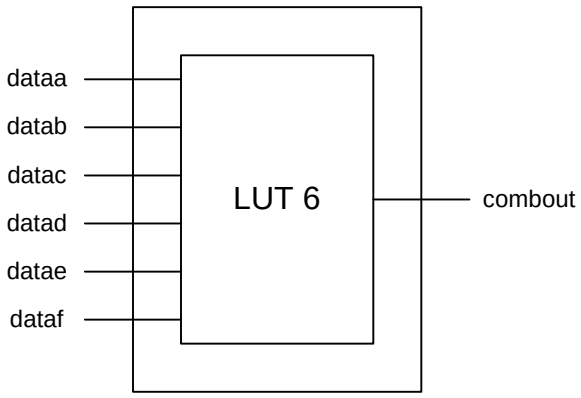
The most common usages of the combinational logic cell are normal, arithmetic, and shared arithmetic.

3.5.1 Normal Mode

Normal mode implements the function of the 6 data inputs described by the LUT mask.

Combout = F(dataa, datab, datac, datad, datae, dataf)

Figure 2 -- Normal Mode of Combinational Logic Cell



3.5.2 Arithmetic Mode

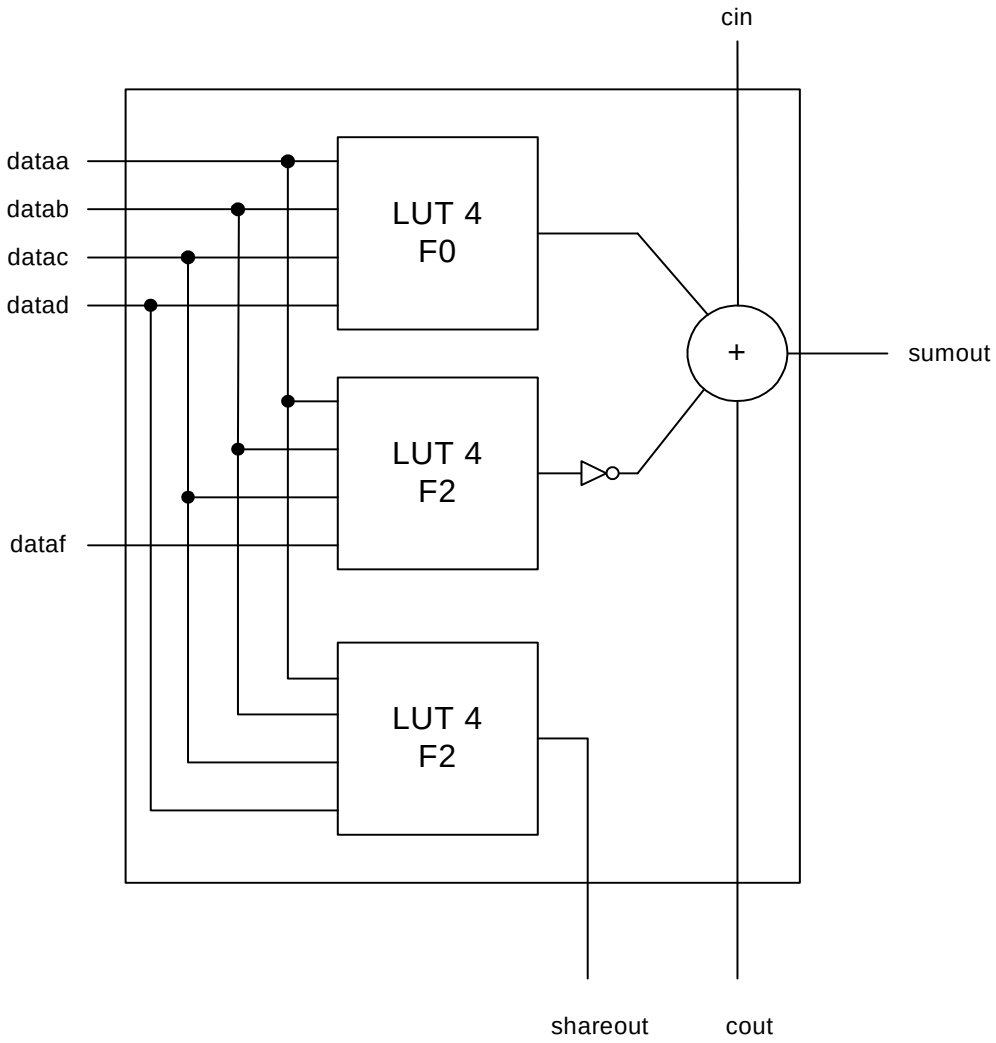
Arithmetic mode implements an addition of two functions of 4 data inputs and the cin. Normally a sum and a carry out are generated. If the next cell on the carry chain is in shared arithmetic mode (see section 3.5.3), this cell also generates a shareout, which feeds the sharein of the next cell. **However, the switch from arithmetic mode to shared arithmetic mode can only happen at an ALM boundary**. See section 5 for the ALM description.

Note that the last data input to the second function is actually the dataf signal and not the datae signal, whereas the last data input to the shareout function is the datad signal. Also note that the functions used for the equation are the first and third functions described in the LUT mask. See section 3.7 for more information about how to interpret the LUT mask.

Cout, Sumout = F0(dataa, datab, datac, datad) + ! F2(dataa, datab, datac, dataf) + cin

Shareout = F2(dataa, datab, datac, datad)

Figure 3 -- Arithmetic Mode of Combinational Logic Cell



3.5.3 Shared Arithmetic Mode

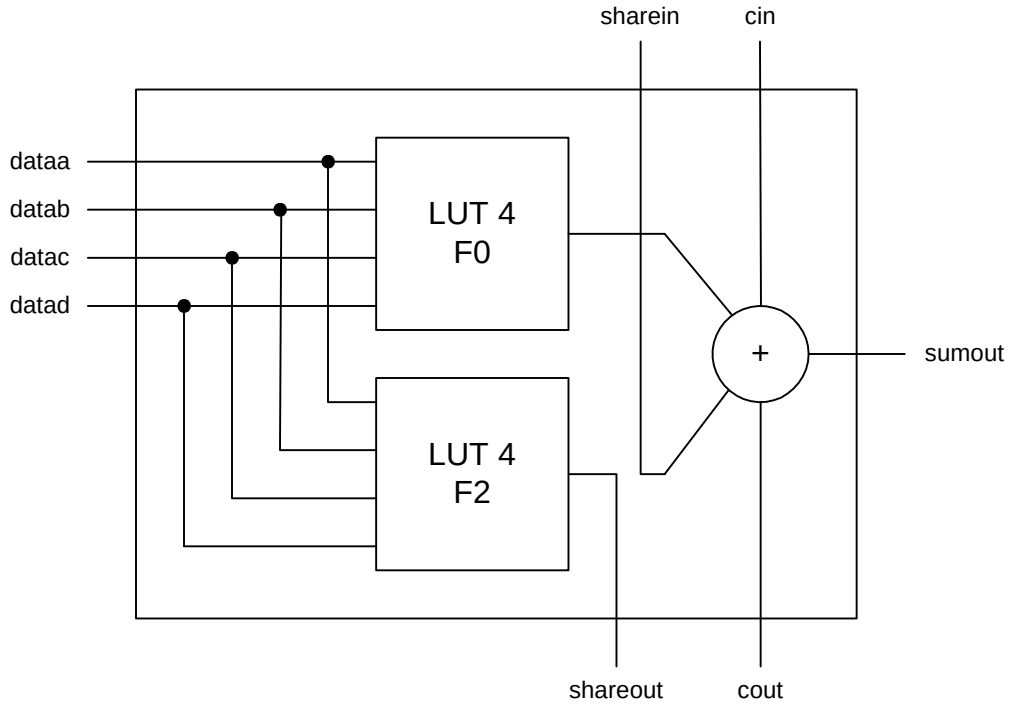
Shared arithmetic mode implements an addition of a function of 4 data inputs, the sharein and the cin. A sum and a carry out are generated. Additionally, a shareout signal may be generated as a function of the same 4 inputs used in the addition function. If the next cell on the chain is in arithmetic mode (see section 3.5.2), the shareout signal is not used. **However, the switch from shared arithmetic mode to arithmetic mode can only happen at an ALM boundary**. See section 5 for the ALM description

Note that the functions used for the equation are the first and third functions described in the LUT mask. See section 3.7 for more information about how to interpret the LUT mask.

Cout, Sumout = F0(dataa, datab, datac, datad) + sharein + cin

Shareout = F2(dataa, datab, datac, datad)

Figure 4 -- Shared Arithmetic Mode of Combinational Logic Cell



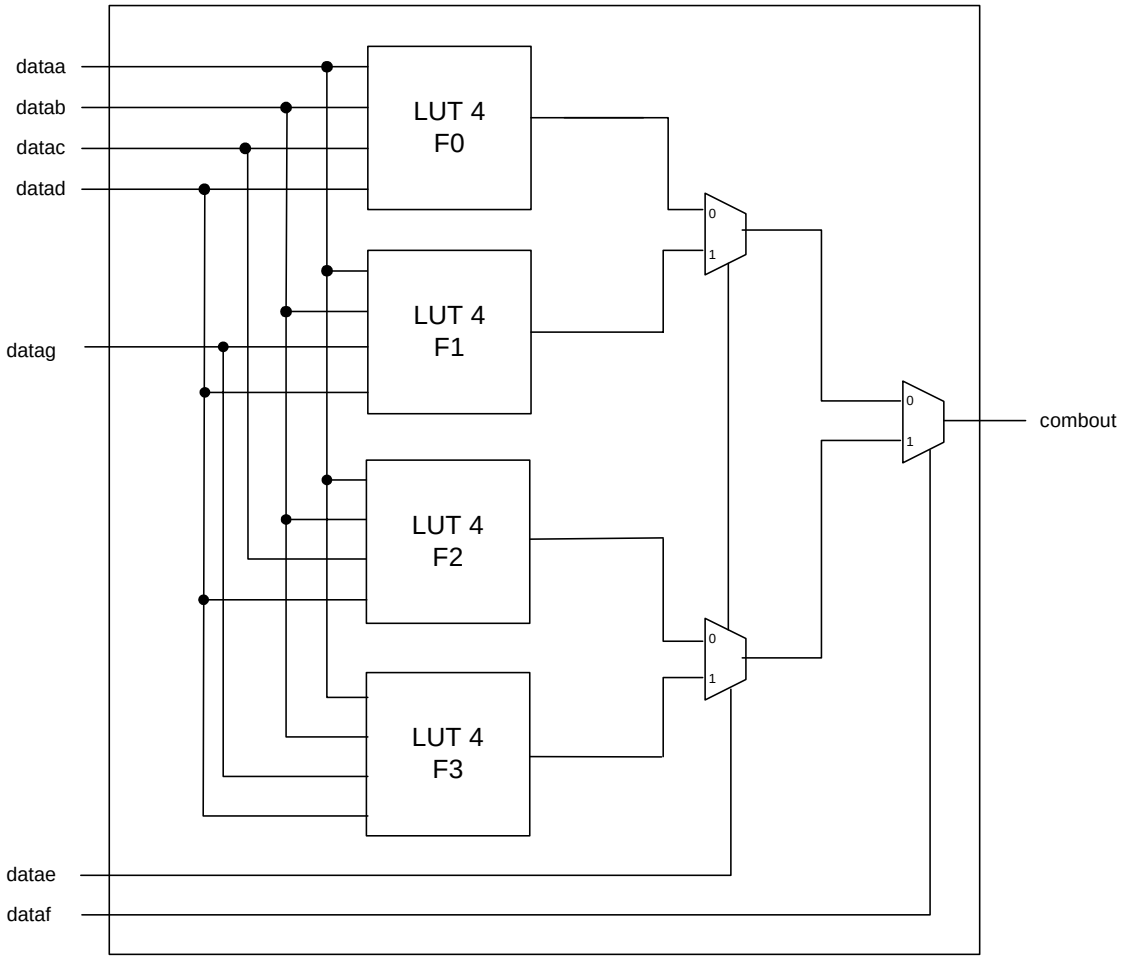
3.6 Exotic Combinational Logic Cell Usage

In this section, we describe two modes of the Stratix II combinational logic cell that are useful in certain situations.

3.6.1 Extended LUT Mode

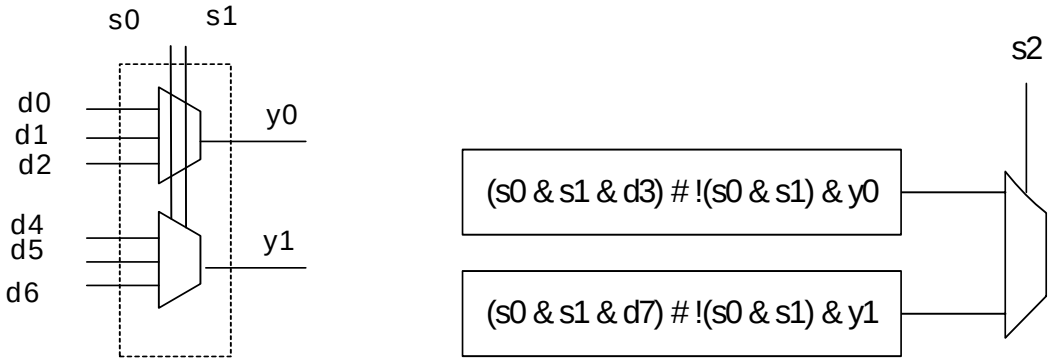
Extended LUT mode implements some functions of 7 data inputs. In this mode, the first and the third 4-input functions represented by the LUT mask use data inputs dataa, datab, datac, and datad; whereas the second and the fourth 4-input functions use data inputs dataa, datab, datag, and datad. See section 3.7 for more information about how to interpret the LUT mask.

Figure 5 – Extended Lut Mode of Combinational Logic Cell



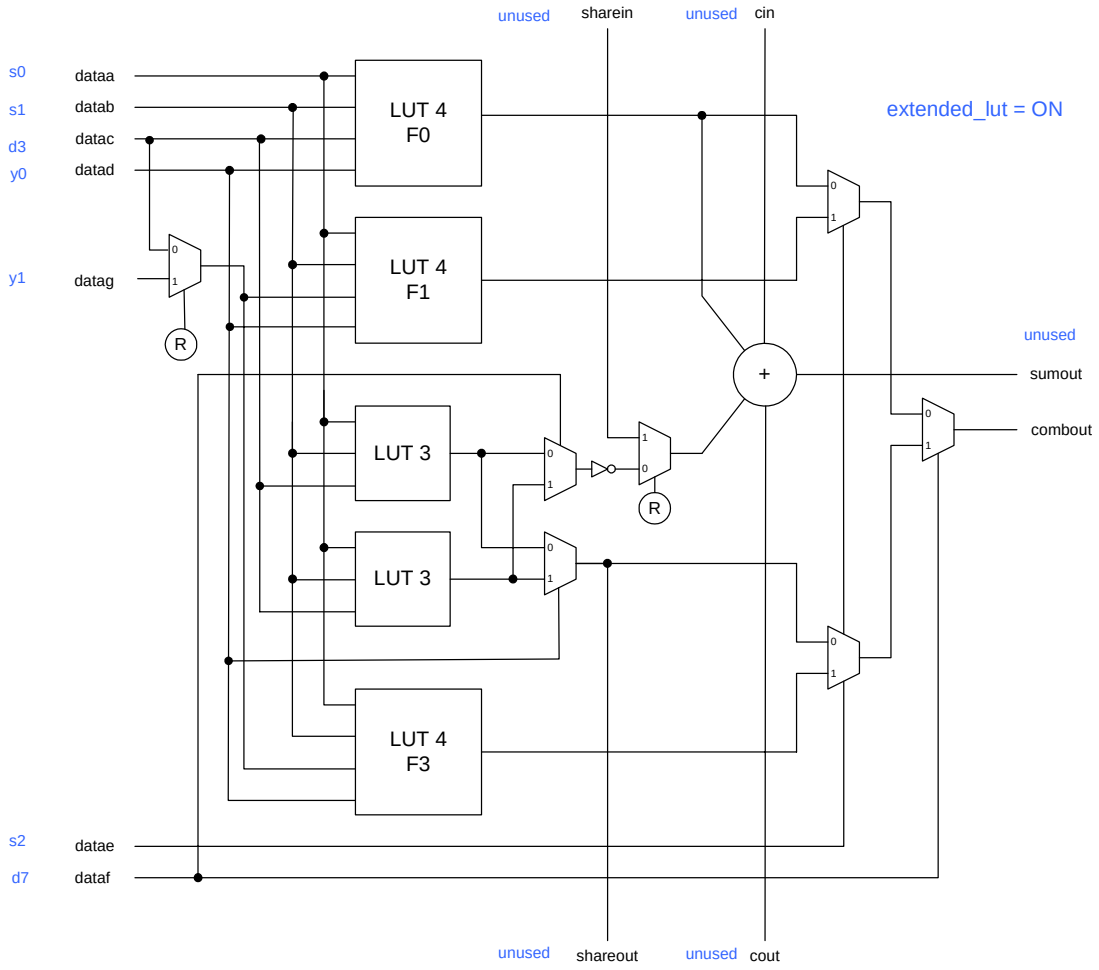
One important application of this mode is to implement 8-to-1 MUXes with two ALMs: let s0, s1, s2 be the select, d0, …, d7 be the data. Then the MUX can be decomposed into two stages as described in Figure 6.

Figure 6 -- Decomposing 8-to-1 MUXes



The first stage fits in one ALM since it represents two 5-input functions that share two inputs (see the section 5 for more details). The second stage, which is a 7-input function, can be implemented in one combinational logic cell in extended LUT mode. See Figure 7 for details.

Figure 7 – The Second Stage of 8-to-1 MUXes



3.6.2 Mixed Normal and Arithmetic Mode

The Stratix II logic cell can actually support using the normal combinational function and the arithmetic function at the same time, provided the LUT mask usage is compatible. The usefulness of this capability is not generally known, although the following function:

REG = MAX(X,Y)

Can be efficiently implemented in a single carry chain by using the carry chain to implement the function X < Y. Passing X through as the combout function. Connecting Y as the adatasdata signal to the register. Finally connecting the last cout as the sload for all the registers. Figure 8 and Figure 9 demonstrate how this function is described in the Stratix II architecture.

Figure 8 -- Combinational Logic Cell Implementing Combout = X & Cout = X < Y

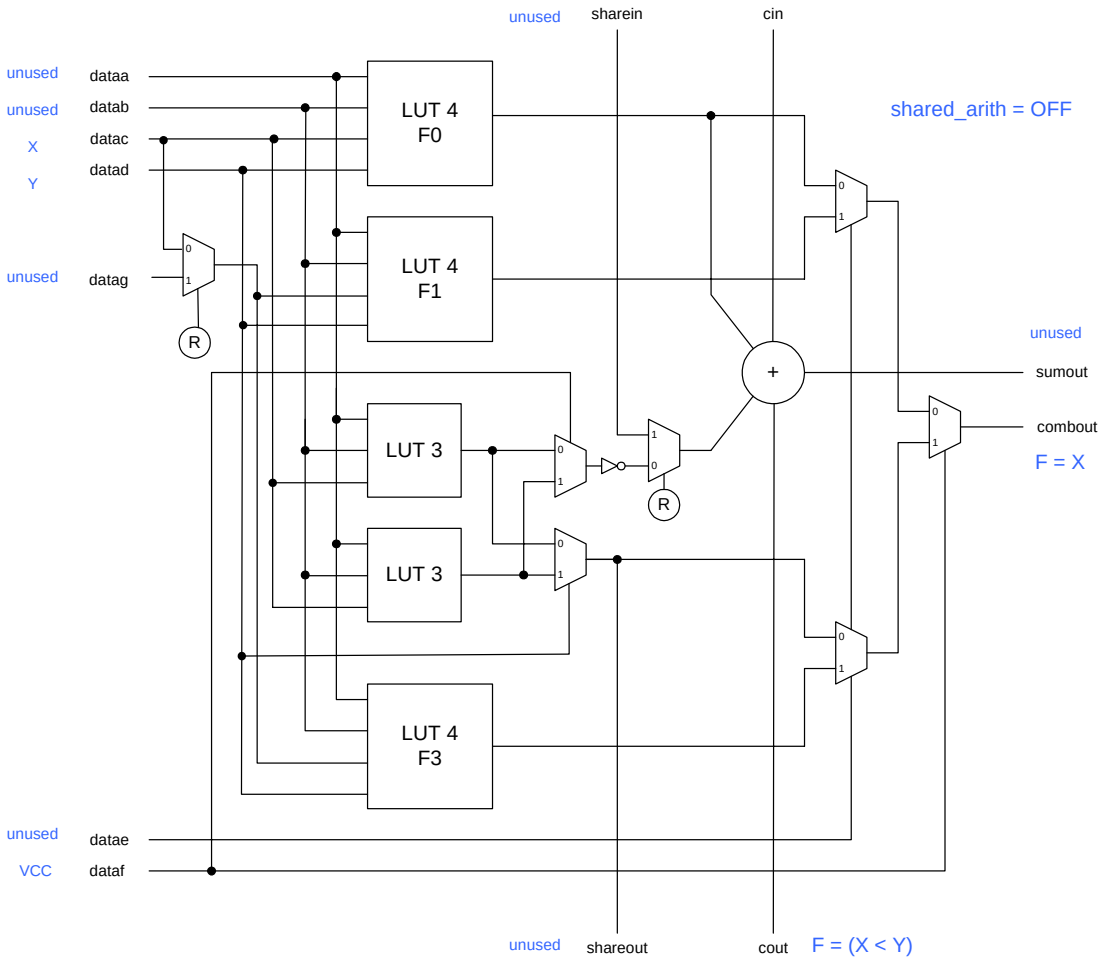
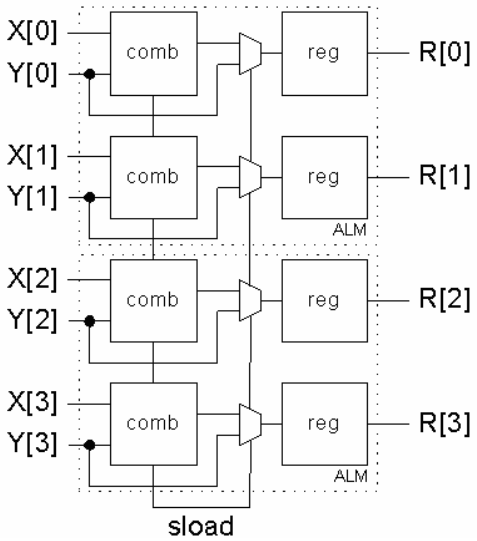


Figure 9 -- Chain of Combinational & Register Logic Cells Implementing REG = MAX(X,Y)



3.7 Interpreting the LUT Mask

<*lut_mask*> designates the content of the LUT(s) in the logic cell. It can be specified as a sixteen digit hexadecimal number, or a sixty-four digit binary number, and represents the 64 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_{63} o_{62} o_{61} … o_2 o_1 o_0

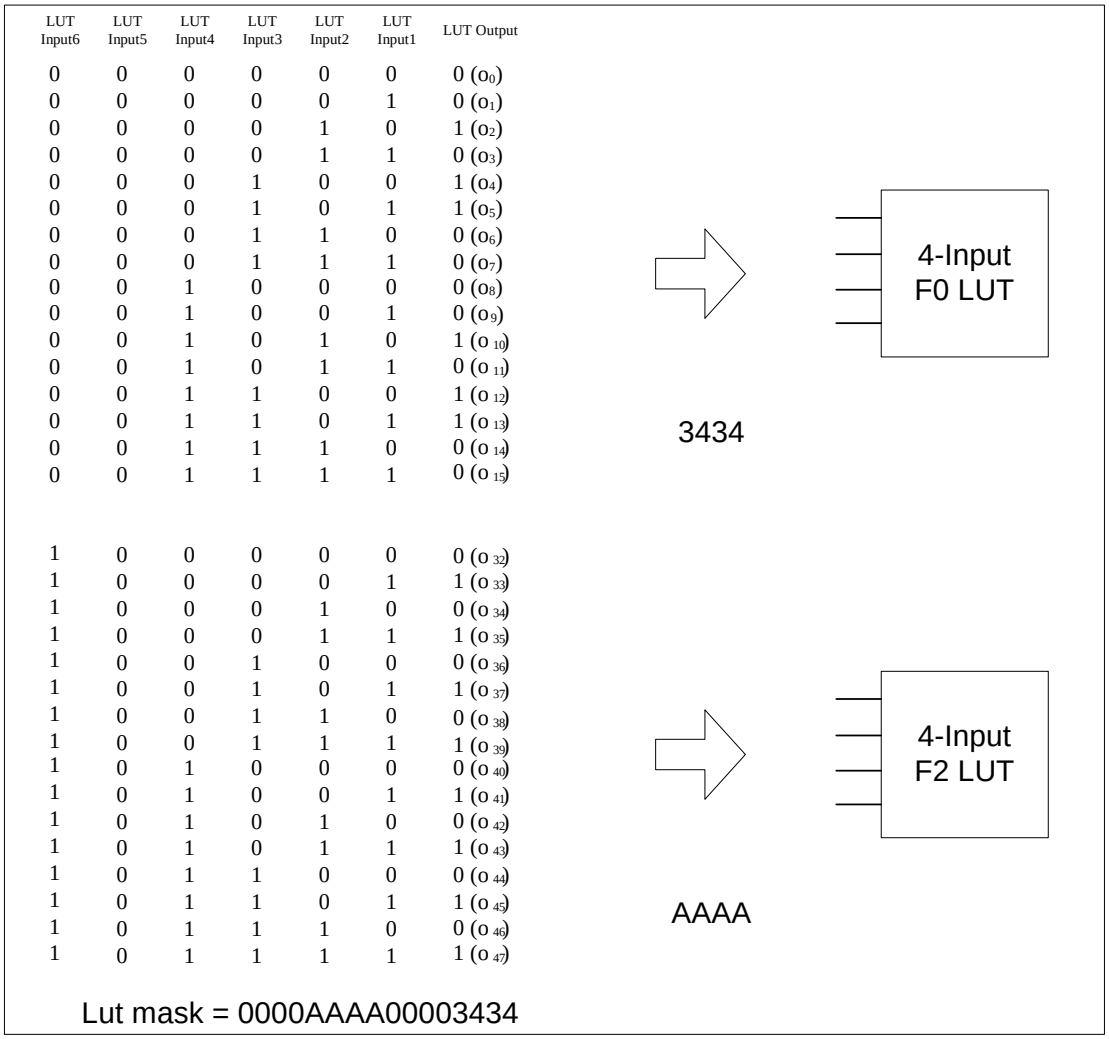
Table 1 -- LUT Mask Data Ordering when in normal mode.

LUT Input6	*LUT Input5*	*LUT Input4*	*LUT Input3*	*LUT Input2*	*LUT Input1*	*LUT Output*
0	0	0	0	0	0	O_0
0	0	0	0	0	1	O_1
0	0	0	0	1	0	O_2
0	0	0	0	1	1	O_3
0	0	0	1	0	0	O_4
0	0	0	1	0	1	O_5
0	0	0	1	1	0	O_6
0	0	0	1	1	1	O_7
…	…	…	…	…	…	…
1	1	1	1	1	1	O_{63}

If one pictures a six-input LUT as a tree of 2to1 MUXes, then *LUT Input1* controls the first stage, and *LUT Input6* controls the last stage, whose output is *LUT Output*.

By setting (*LUT Input6, LUT Input5*) = (0, 0), (0, 1), (1, 0), or (1, 1), the 64-bit LUT mask can be naturally divided into four 16-bit regions, F0, F1, F2, and F3, each of which represents one four-input LUT. This is useful when the cell is in arithmetic or shared arithmetic mode since two four-input LUTs implementing the F0 and F2 functions are used. In this case, the 16 *LUT Output* bits corresponding to *LUT Input6* = 0 and *LUT Input5* = 0 go to the F0 function and the 16 *LUT Output* bits corresponding to *LUT Input6* = 1 and *LUT Input5* = 0 go to the F2 function. Figure 10 shows how the LUT mask 0000AAAA00003434 is mapped onto the F0 and F2 LUTs.

Figure 10 -- How a 64-bit LUT mask is divided into the F0 & F2 LUTs



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

See section 5 for details of how the LUT mask may be shared in ALM packing.

3.8 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, .datae, .dataf, .datag	Positive Logic	Yes (By LUT mask manipulation)
.cin	Active High	No
.sharein	Active High	No

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

4. Logic Cell Register Primitive

```
stratixii_lcell_ff <lcell_name>
(
      .datain(<register data source>),
      .clk(<clock source>),
      .aclr(<asynchronous clear source>),
      .aload(<asynchronous load source>),
      .sclr(<synchronous clear source>),
      .sload(<synchronous load source>),
      .adatasdata(<register asynch/synch data source>),
      .ena(<clock enable source>),

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

4.1 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, .adatasdata, .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.

.aload(<*asynchronous load source*>**),** designates the asynchronous load signal for the logic cell. When asserted the value of the .adatasdata port is applied to the register. Hence .adatasdata must be connected if .aload is connected. 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 .adatasdata. Hence .adatasdata 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.

.adatasdata(<*asynchronous/synchronous load data source*>**),** designates the asynchronous or synchronous data signal for the logic cell. When .aload is high then the register is asynchronously loaded from .adatasdata. When .sload is high then the register is synchronously loaded from .adatasdata. Hence .adatasdata must be connected if .aload or .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.

4.2 Register Logic Cell Output Ports

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

4.3 Register Logic Cell Modes

There are no modes.

4.4 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
.aload	Active High	Yes
.sclr	Active High	Yes
.sload	Active High	Yes
.adatasdata	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.

4.5 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
.aload	2
.ena	3
.sclr	4
.sload	5

4.6 Register Logic Cell Control Signal Choices

The register control signal flexibility in the Stratix II architecture is approximately the same as the Stratix architecture. Every LAB has the following register control signals available: 2 clocks, 3 clock-enables, 2 asynchronous clears, 1 asynchronous load, 1 synchronous clear and 1 synchronous load. This biggest difference from the Stratix family is the addition of 1 clock-enable and the flexibility in combinations of global and non-global sources for these control signals.

Along with the global network there are 6 normal routing connections in the LAB for these 9 control signals.

4.6.1 Clocks & clock-enables

All combinations of the 3 clock-enable and 2 clock signals can be used to create 3 LAB clocks. Every register in the LAB may then choose from one of these 3 clocks. In the Stratix architecture, 2 clock and clock-enable signals are used to create 2 LAB clocks. The clocks can be routed on the dedicated global clock network, while the clock-enables must be routed using normal routing resources. An always-enabled clock uses up one of the 3 LAB clock lines, but does not use one of the 6 normal routing paths for control signals.

It may be necessary to limit the creation of many low-fanout clock-enables since this may negatively impact LAB utilization. Low fanout clock-enables can be re-mapped to normal logic on the data path. Never convert them to logic gated clocks since this will still hurt LAB utilization and create unstable clocking environments.

4.6.2 Asynchronous load & clear

Every LAB has 2 asynchronous clears and 1 asynchronous load. The asynchronous load is paired with the one of the asynchronous clears. Each register can choose to either be controlled by an asynchronous load and an asynchronous clear or just an asynchronous clear. Any of the 3 asynchronous control 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 controls. This is the exact same restriction as the Stratix device.

The asynchronous clear signals may be routed using either the normal routing resources or the dedicated global resources. The asynchronous load my only be routed using the normal routing

resources. This is an improvement to the Stratix architecture where only 1 of the asynchronous clears could be routed using the global routing network.

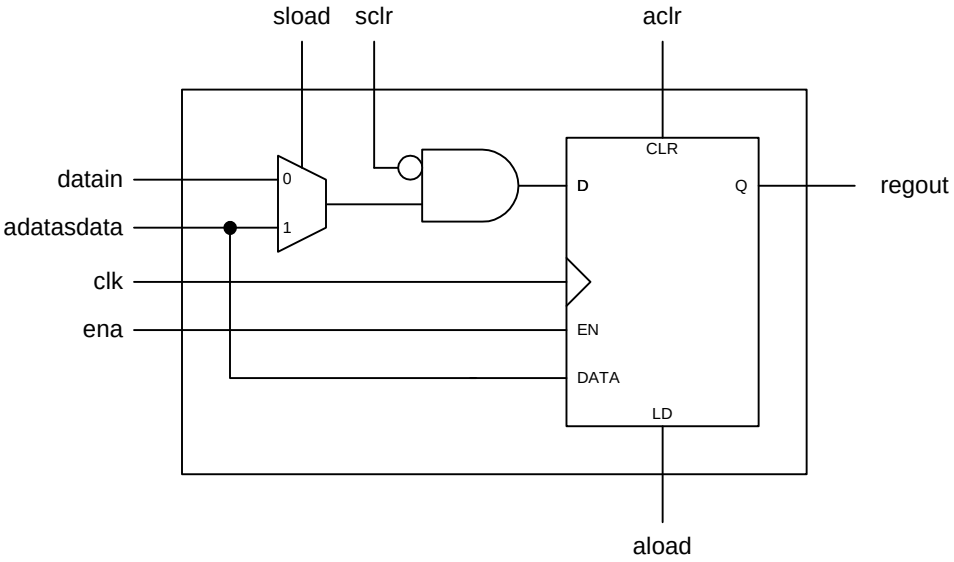
4.6.3 Synchronous load & clear

The synchronous load and clear restrictions are identical to the Stratix 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. So 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.

Because of these restrictions it is recommended that synchronous control signals only be used with arithmetic chains.

4.7 Register Logic Cell Block Diagram

Figure 11 -- Register Logic Cell

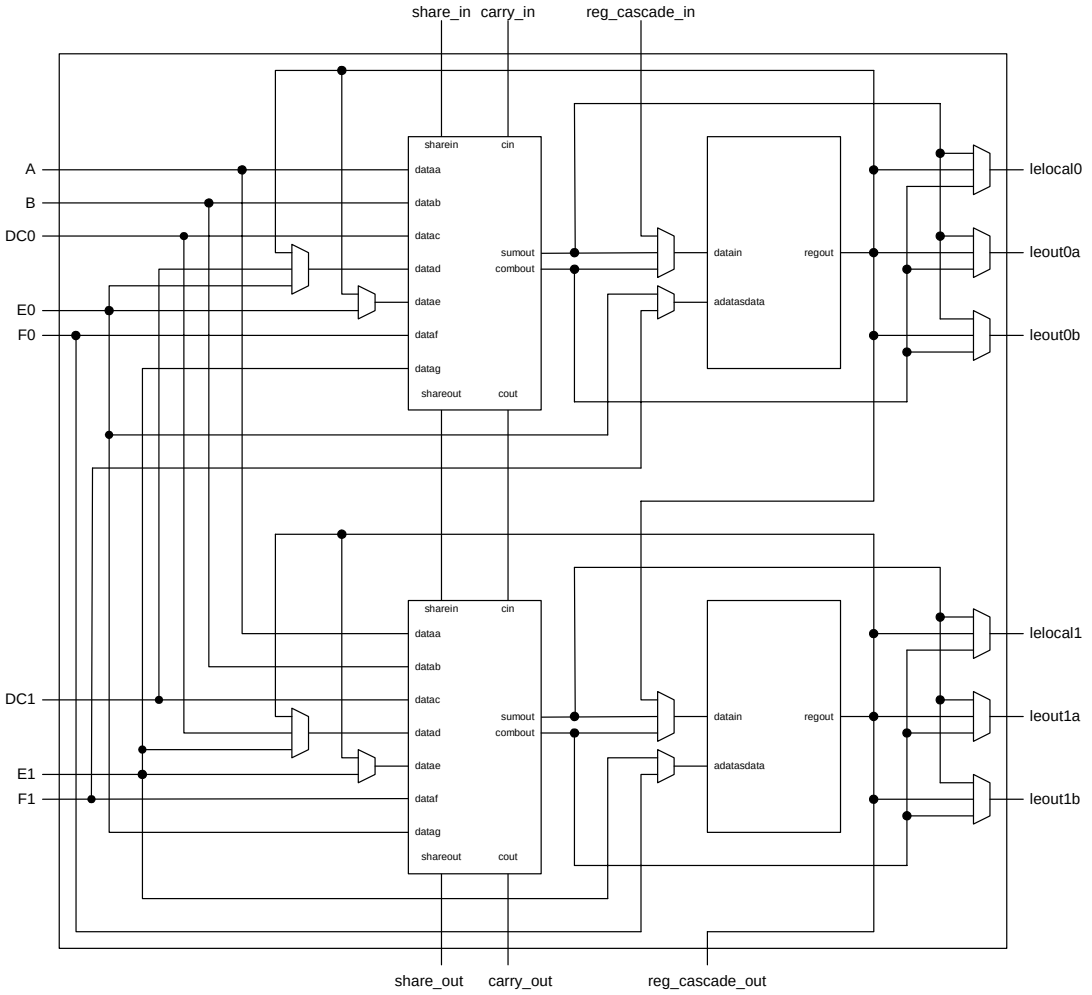


5. ALM Description

An ALM is a grouping of two combinational and two registered logic cells that share data inputs and LUT mask information. A single ALM can support two independent 4-input functions, or two independent 5-input functions that share two inputs, or two related 6-input functions that share four inputs, or the subset of 7-input functions as described in section 3.6.1.

Figure 12 shows a logical representation of an ALM and the four logic cells contained within it. This diagram shows clearly how the ALM inputs and outputs are connected to the logic cells and how they are connected to each other.

Figure 12 – ALM Logical Representation



One thing that is not clear from this picture is what is meant by related 6-input functions. Figure 13 gives an accurate description of the physical implementation of an ALM. It is clear from this picture that an ALM contains only 64-bits of LUT mask that must be shared by the two combinational logic cells. It should also be clear from this picture that the logic interpretation of the LUT mask must be translated to the physical interpretation differently depending on whether the LUT mask is being used as a 6 or more input function or 5 or fewer input function. This translation will be done in the assembler.

Figure 13 -- ALM Physical Representation