LCELL WYSIWYG Description for Stratix III

Version 2.0

March 9, 2009

by

Altera Corporation

Table of Contents:

1. Overview

It's assumed that the reader is familiar with the Stratix II LCELL WYSIWYGs

This is a summary of the LCELL WYSIWYG and atom support for Stratix III. The basic logic tile in Stratix III will continue to be called an ALM (Adaptive Logic Module). The ALM physically contains 2 combinational and 2 register blocks.

Stratix III LCELL primitives are similar to the Stratix II ones. Highlights of the differences are

- Asynchronous load has been removed from the Stratix III LAB registers
- One can not simultaneously get the sumout and the combout on the Stratix III combinational primitive. However, the adder circuitry can still be used to form the cin to cout path in case combout is used
- The 2 combinational blocks can work together to form a so-called LUT register block. LUT registers will still be modeled with the normal register primitive.

We 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.

Section 2 describes the combinational WYSIWYG block.

Section 3 describes both the dedicated register and LUT register. In Stratix III, we are modeling IO registers separately from the rest of the IO cell, similar to what we have been doing with lcell registers, thus registers used in LAB and I/O blocks are modeled with a common primitive. This register primitive is documented separately.

Section 4 describes how they are connected in an ALM, in particular, how the LUT register block is formed.

2. Combinational Logic Cell Primitive

```
Stratix III_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>;
```

2.1 Combinational Logic Cell Input Ports

Same as in the Stratix II case.

2.2 Combinational Logic Cell Output Ports

Same as in the Stratix II case *except that combout and sumout can not be connected at the same time*. Cin/Cout and sharein/shareout connections are still allowed even if combout is used.

2.3 Combinational Logic Cell Modes

Same as in the Stratix II case.

2.4 Combinational Logic Cell Block Diagram

Same as in the Stratix II case.

2.5 Common Combinational Logic Cell Usage

Same as in the Stratix II case: the most common usages of the combinational logic cell are normal, arithmetic, and shared arithmetic.

2.6 Exotic Combinational Logic Cell Usage

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

2.6.1 Extended LUT Mode

Same as in the Stratix II case.

2.6.2 Mixed Normal and Arithmetic Mode

Same as in the Stratix II case except that the sumout is not available. Note that the following example function does not use the sumout:

REG = MAX(X,Y)

2.7 Interpreting the LUT Mask

Same as in the Stratix II case.

2.8 Combinational Logic Cell Polarities and Default Values

Same as in the Stratix II case.

3. LAB Register Primitive

Registers in the LAB do not support asynchronous load. See the Stratix III register primitive documentation for more details on the primitive itself.

3.1 LUT Register

Registers created from a pair of combinational blocks in an ALM can support the use of asynchronous clear and synchronous clear.

If the synchronous clear is used, then the LUT register will be modeled as a Stratix III register primitive that has its sclear port connected. If the synchronous clear is not used, the hardware can implement the register plus an arbitrary two-input function as its data input port. This two-input function will be modeled using a Stratix III combinational lcell and the LUT register itself will be modeled using a Stratix III register primitive.

LUT registers do not support asynchronous load or synchronous load. LUT registers also must share the clock, clock enable, and the asynchronous clear signals with the top dedicated register in the same ALM.

See Section 4 for more details on how LUT registers are formed.

3.2 LAB Register Control Signal Choices

Same as in the Stratix II case *except that asynchronous load has been removed*. The synchronous control signal rules apply to dedicated registers in the LAB, not to LUT registers.

4. ALM and LUT Register Description

An ALM is a grouping of two combinational and two registered blocks 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 2.6.1.

A single ALM can also support one asynchronously clear-able register (called LUT register) that shares its clock/clock enable/asynchronous clear signal with the top dedicated register in the same ALM. In this mode, the hardware can implement a synchronous-clearable register. If synchronous clear is not used, then the same ALM can implement an arbitrary two-input function as the LUT register's data input in addition to the LUT register. The two inputs come from DC1 and F1 (see Figure 3). In the diagrams below, we will use synchronous-clearable register as an example of the LUT register.

Figure 1 shows a logical representation of an ALM and the four blocks contained within it when the LUT register is not used. 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 1 – ALM Logical Representation

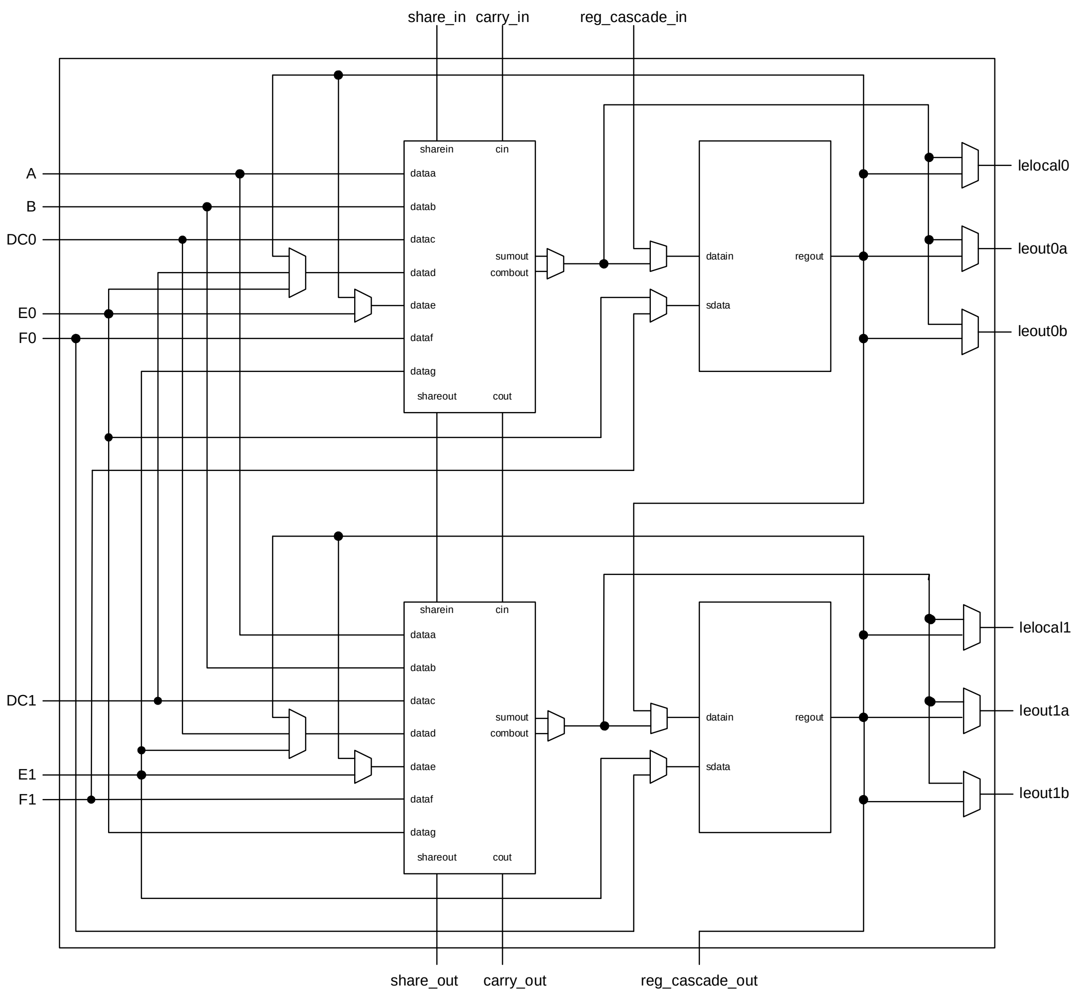


Figure 2 shows one way of constructing a synchronously-clearable LUT register from 2 combinational blocks and the dedicated feedback paths in the ALM. Dotted lines mean that those connections are not active for creating the LUT register.

Figure 2. LUT Register from 2 Combinational Blocks

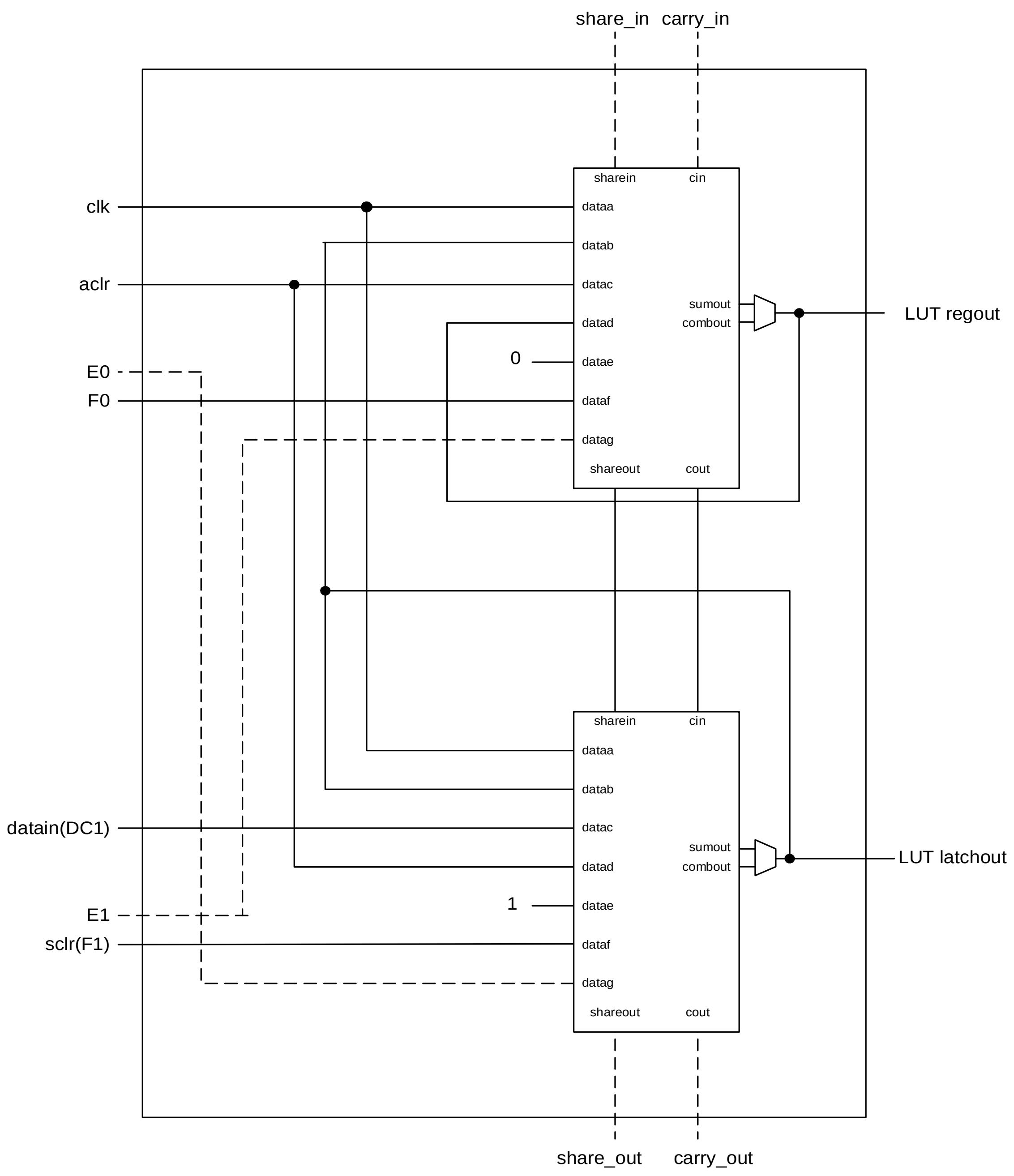


Figure 3 shows the logical representation of an ALM when the LUT register is used. Again the dotted lines are connections that are not active. Note that inputs A, B, and DC0 are not used in this mode.

Figure 3. ALM: LUT Register Mode

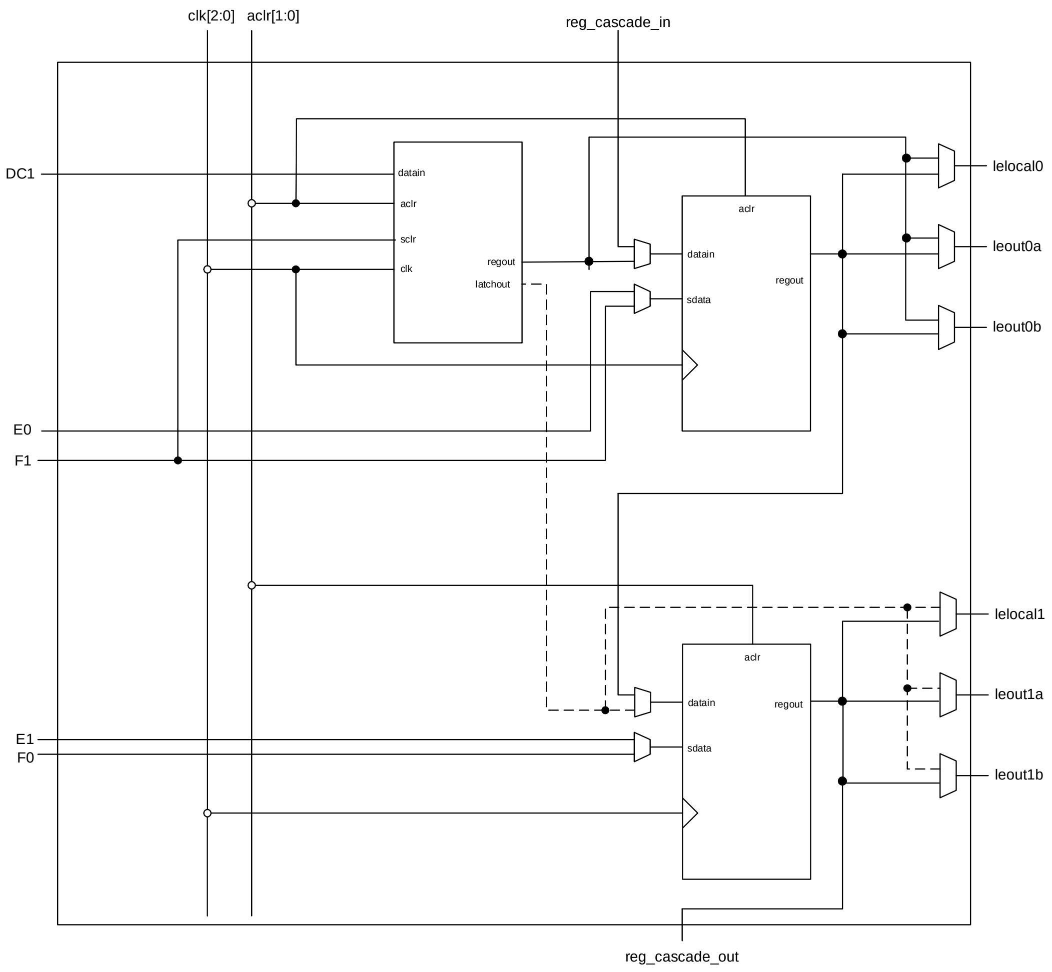
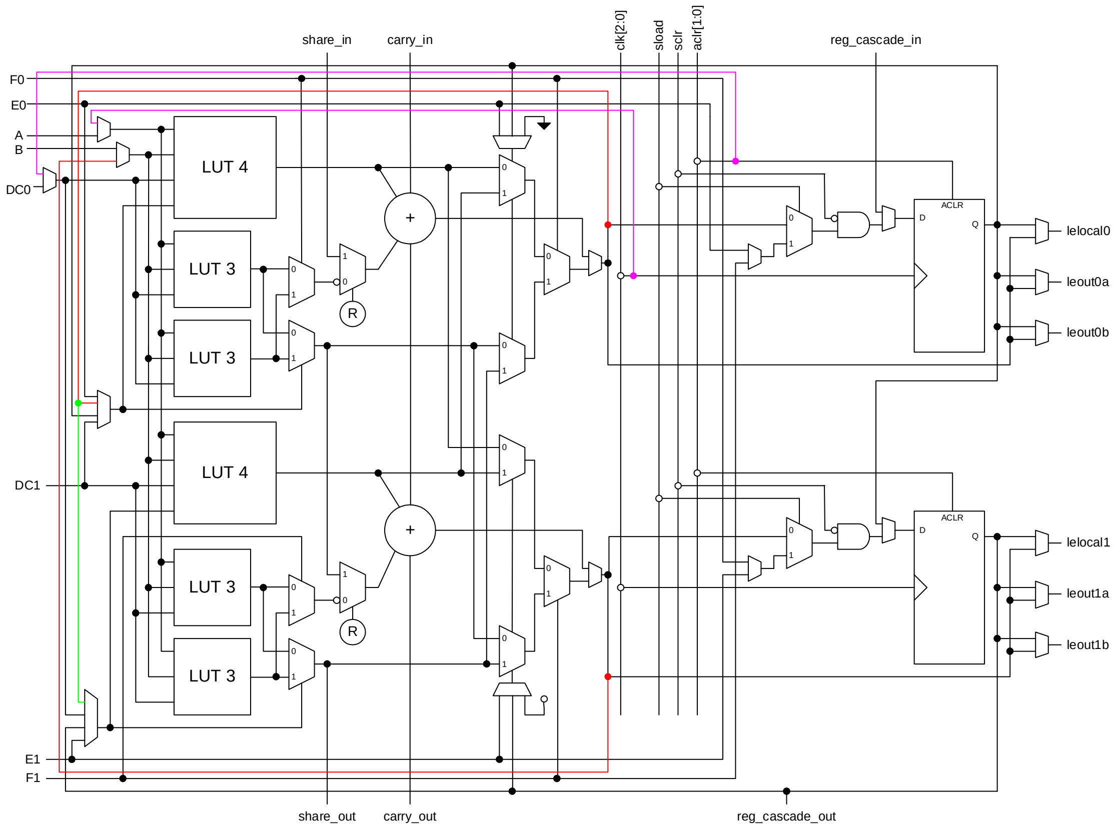


Figure 4 gives an accurate description of the physical implementation of an ALM. In this diagram, the asynchronous clear and clock paths to the combinational elements are highlighted, so are the other dedicated feedback paths used for the LUT register functionality.

It is clear from the picture that how we combine the two combinational elements to form the LUT register and how the LUT register shares its clock, clock enable, and asynchronous clear sources with the top dedicated register. The dedicated path from the top combinational block to the bottom one (at the D-stage, marked as green), is only available in LABs which do not support memory operation, i.e. non-MLABs.

Figure 4 -- ALM Physical Representation


share_in
carry_in
clk[2:0]
sload
sclr
aclr[1:0]
reg_cascade_in
F0
E0
A
B
DC0
LUT 4
LUT 3
LUT 3
+
R
ACLR
D
Q
lelocal0
leout0a
leout0b
DC1
LUT 4
LUT 3
LUT 3
+
R
ACLR
D
Q
lelocal1
leout1a
leout1b
E1
F1
share_out
carry_out
reg_cascade_out