# Stratix III LUTRAM Feature Functional Description

Version 3.0

March 9, 2009

by

Altera Corporation

# Table of Contents:

## 1. Overview

This document describes the support plan for LUTRAM in Stratix III.

Please refer to the WYSIWYG document for the exact description of the Stratix III LUTRAM primitive.

The LUTRAM is a new mode of the LUT available in Stratix III. Stratix III LUTRAM has the following features:

1. General-purpose Simple Dual-Port SRAM array and only exists in MLAB.
2. Can function as a simple dual-port SRAM, as a single port SRAM with external soft logic, or as a ROM by disabling the WE signal.
3. For ROM operation: every ALM consists of two 32x1 LUTRAM blocks storing 64 bits of memory per ALM; for RAM operation, every ALM consists of two 16x1 LUTRAM blocks storing 32 bits of memory per ALM.
4. For ROM operation, 64 memory bits per ALM and total 640 memory bits per MLAB; for RAM operation, 32 memory bits per ALM and total 320 bits per MLAB in RAM mode.
5. For ROM operation: 2 configuration modes, 64-address deep (64-deep) mode and 32-address deep (32-deep) mode. The 64-deep mode allows 1-bit data width per ALM and the 32-deep mode allows 2-bit data width per ALM. Maximum data width of an MLAB is 10 in 64-deep mode or 20 in 32-deep mode; for RAM operation, LUTRAM can only be configured into 16-address deep mode.
6. Write operation must be synchronous to LAB_CLK0.
7. Byte Enable is not supported for RAM operation
8. Read operation can be either combinational read or registered read.

This document is organized as follows. Some commonly used terminologies are defined in Section 2. Section 3 describes the basic read and write operations of LUTRAM. In Section 4, LUTRAM and Block RAM are compared. Section 5 describes user RAM modes and how they are supported with LUTRAM. In Section 6, impact on the user flow is described.

## 2. Glossary

**Block RAM**: M9K and M144K.

**ALM**: Adaptive Logic Module. Basic logic unit in Stratix III.

**LAB**: Logic Array Block. Each Stratix III LAB contains 10 ALMs.

**MLAB**: Logic Array Block that can be configured as dual port RAM. 50% of the LABs in Stratix III are MLABs.

**LUTRAM**: In contrast to Block RAM, LUTRAM is one configuration mode of MLAB. When configured as LUTRAM, it supports 64 deep x 10 wide or 32 deep x 20 wide ROM or 16 deep x 20 wide simple dual port memory.

# 3. LUTRAM Operation

LUTRAM is not a typical stand-alone embedded memory core in that the read access uses the LUT structure vs precharged-discharged bit lines. It is a general-purpose Simple Dual-Port SRAM array that only exists in MLABs**.**

LUTRAM write must be synchronized to LAB-wide clock LAB_CLK0. A pulse generator in the LCB of the MLAB is hard-wired to the LAB_CLK0 and generates a self-timed pulse-width signal, WEN, on every falling edge of LAB_CLK0.

The WEN write pulse can be disabled in a given clock cycle as follows. The user can disable LAB-wide clock signal LAB_CLK0 using the standard clock-enable. This will disable the write for all ALMs within the MLAB.

The read data path of LUTRAM is the same as the LUT data path, which doesn't require a clock and clock control signals. The read address decoding mechanism is inherited from the existing LUT.

# 4. LUTRAM vs Block RAM

This section lists the major differences between LUTRAM and Block RAM.

## 4.1 Operation Mode

LUTRAM natively supports only simple dual port mode. One needs to emulate single port and ROM mode user memories. LUTRAM does not support true dual port mode.

In contrast, Stratix III block RAM natively supports true dual port mode (thus single port mode) as well as simple dual port mode memories.

## 4.2 Write Operation

Write operation on LUTRAM is controlled by the clock input, the enable input, and optionally the byte enable inputs. The write pulse is generated off the falling edge of the clock. In contrast, for block RAM, the write pulse is triggered off the rising edge.

## 4.3 Read Operation

LUTRAM supports asynchronous read. The read address does not need to be registered and the read is always happening. In contrast, in block RAM, the read is triggered off the clock edge and all inputs are registered including the read address inputs.

## 4.4 Input/Output Registers

Most LUTRAM related registers come from other LABs. The only exception is the data input register, which is hardwired to use the registers from within the same LAB. In contrast, all registers (input as well as output) needed by the block RAM are included in the block.

Other than read address inputs, the other input registers will be checked according to the corresponding block RAM legality rules. This is for simplicity and the rules can be relaxed in the future if needed.

### 4.5 Mixed Port Read-during-write Behavior

If the read and write ports are driven by the same clock, in the case of address collision, block RAM has the option to give the data before the write, i.e. “old data”. In contrast, for LUTRAM, since it is always reading and the write triggers off the falling edge, the mixed port read-during-write behavior is unknown. There will be a window near the falling edge of the clock, during which the output is unknown. Prior to that window, “old data” is read out whereas after that window, “new data” is seen at the output.

Some extra soft logic is needed to get the “old data” behavior, which is going to be handled by the mega-function.

### 4.6 Same Port Read-during-write Behavior

In the case of block RAM, if the RAM is configured in true dual port mode, then on the same port, during the write cycle, the output can either show the “old data” or the “new data”.

Since LUTRAM does not natively support true dual port mode, single port user memories implemented in LUTRAM will get unknown data at the output during a write cycle – there will be a window near the falling edge of the clock, during which the output is unknown. Prior to that window, “old data” is read out whereas after that window, “new data” is seen at the output.

### 4.7 Mixed-Width Support

LUTRAM does not natively support mixed data with between read and write. A user memory that uses this feature needs to be emulated with multiple LUTRAM blocks.

### 4.8 Depth/Width Configurations

Block RAM support many different configurations. LUTRAM supports 32 words deep or 64 words deep mode for ROM operation and 16 words deep for RAM operation. Each MLAB can be either 32x20 or 64x10 for ROM and 16x20 for RAM.

### 4.9 Write Enable/Read Enable

In the case of LUTRAM, the clock enable input is used as the enable for the write operation. Write enable needs to be combined with the clock enable input.

There is no dedicated read enable for LUTRAM. Read enable needs to be emulated with clock enable on the read address registers.

### 4.10 Asynchronous Clear on Output Latch

The output is not latched in the case of LUTRAM so block RAMs that use this feature can not be converted to LUTRAMs.

### 4.11 Address Stall

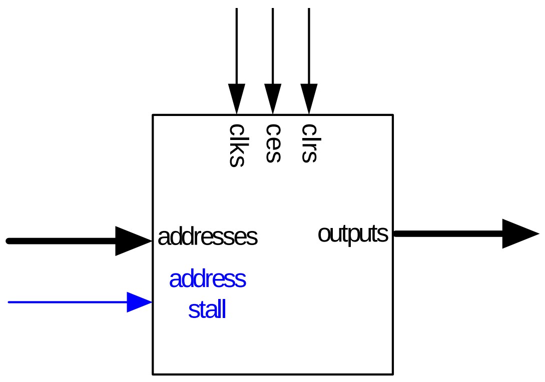
LUTRAM does not natively support address stall, extra soft logic (clock enable on the address register) is needed to emulate it. Block RAM has dedicated input for address stall support.

## 5. Operation Modes

This section describes the various operation modes in user's view and how they are supported in LUTRAM. Basically, there are 3 modes supported. They are ROM, single-port, dual-port. True dual port mode is not supported by LUTRAM.

### 5.1 ROM (Read-Only Memory)

ROM is a pre-initialized memory block that can only perform the read operation. The following picture illustrates its I/O interfaces. The addresses and the outputs can be separately registered.

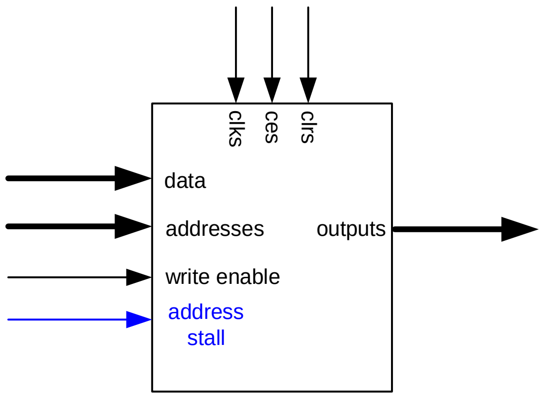


The followings show how the above functional ports map to the WYSIWYG.

| | |
|---|---|
| addresses | -> raddr plus external registers (in case of synchronous read) |
| address stall | -> soft logic in front of the read address register |
| outputs | -> dataout plus external registers (in case of registered output) |
| clks | -> clk plus clock on external registers |
| ces | -> ena plus clock enable on external registers |
| clrs | -> clr on external registers |

### 5.2 Single-Port

In a single-port RAM, you can only access one location of the RAM at one time. You can perform read/write operations on the memory block. The following picture shows its I/O interfaces. The inputs and outputs can be separately registered.

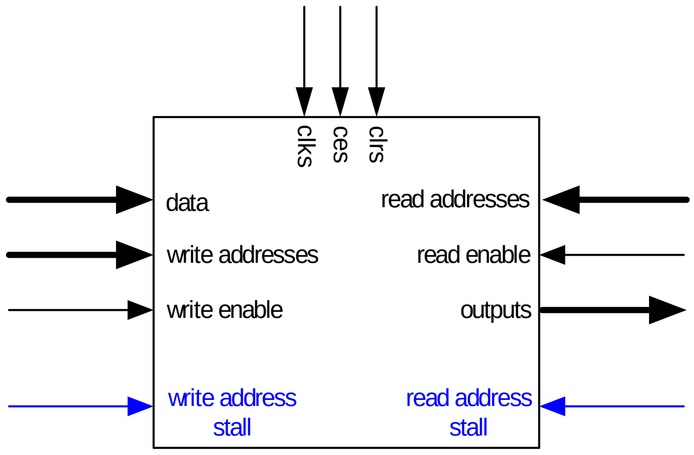


The followings show how the above functional ports map to the WYSIWYG.

| | |
|---|---|
| data | -> datain plus internal (w.r.t the MLAB) registers |
| addresses | -> waddr, raddr plus external registers |
| write enable | -> soft logic in front of the clock enable (ena) input |
| address stall | -> soft logic in front of the address register |
| outputs | -> dataout plus external registers (in case of registered output) |
| clks | -> clk plus clock on external registers |
| ces | -> ena plus clock enable on external registers |
| clrs | -> clr on external registers |

## 5.3 Simple Dual-Port

In simple dual-port mode, you can do 1 read and 1 write operation (1R1W) on different locations at the same time. The following picture shows the I/O interfaces. The read port and the write port can be separately registered.



The followings show how the above functional ports map to the WYSIWYG.

| | |
|---|---|
| data | -> datain plus internal (w.r.t the MLAB) registers |
| write addresses | -> waddr plus external registers |
| write enable | -> soft logic in front of the clock enable (ena) input |
| write address stall | -> soft logic in front of the write address register |
| read addresses | -> raddr plus external registers |
| read enable | -> soft logic in front of the read address register (convert to clock enable) |
| outputs | -> dataout plus external registers (in case of registered output) |
| read address stall | -> soft logic in front of the read address register |
| clks | -> clk plus clock on external registers |
| ces | -> ena plus clock enable on external registers |
| clrs | -> clr on external registers |

# 6. User Flow

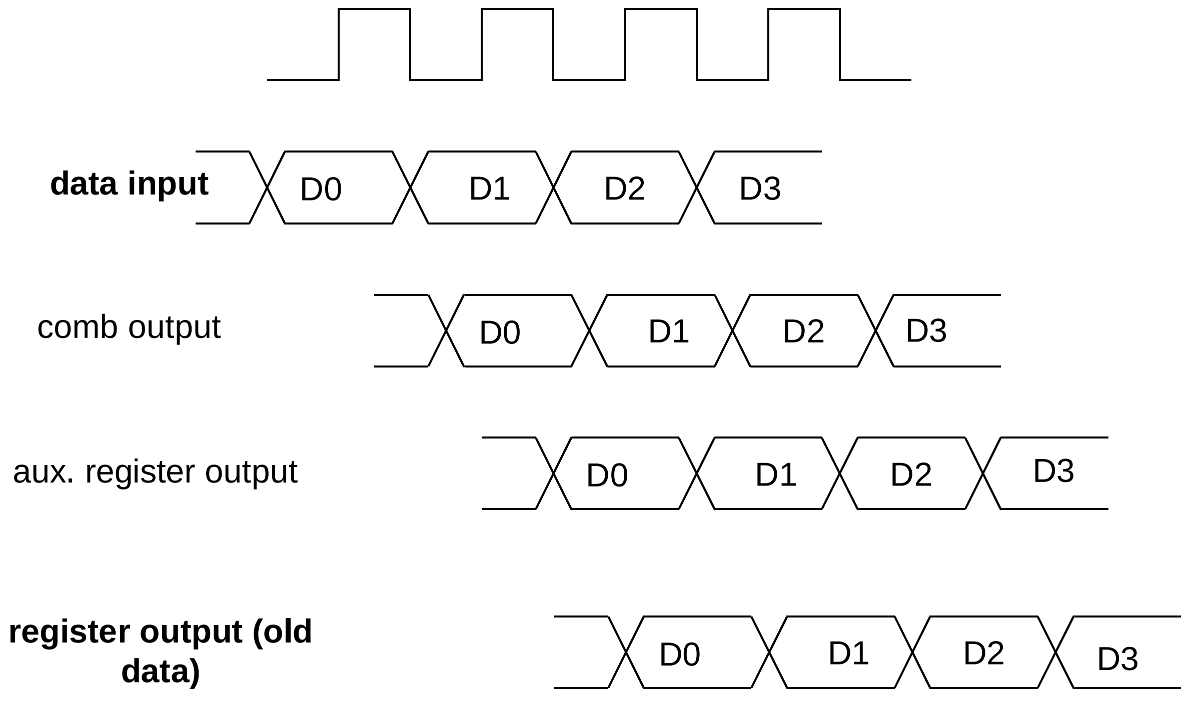
## 6.1 Mega-function

By default, the type should be left as AUTO, in which case, the megafunction instantiates block ram WYSIWYGs.  Fitter can then convert block rams to LUTRAMs when necessary.

If the LUTRAM type is chosen, then LUTRAM wysiwygs are instantiated, along with the associated registers.

### 6.1.1 Read Before Write ("Old Data") Behavior

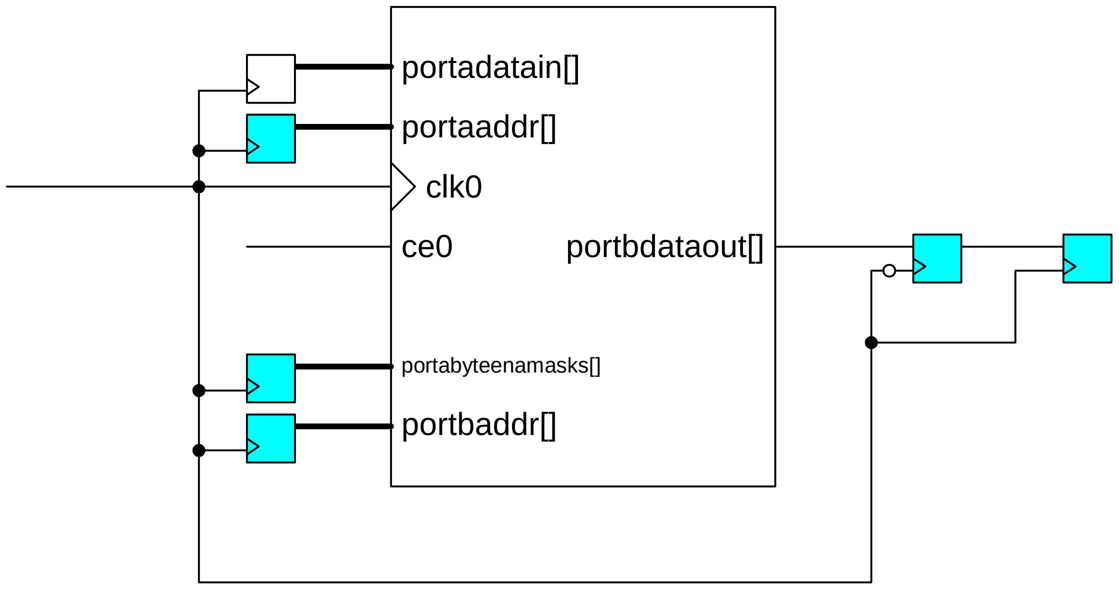
If the lutram uses a single clock to drive both read and write inputs as well as the data out registers, then we can achieve the old data behavior of block RAMs by inserting an aux. register into the netlist. MegaWizard is responsible for inserting this register. The wave form is as follows:



The netlist would look like this: note that all these registers will be part of the netlist.

□ Registers in the same MLAB

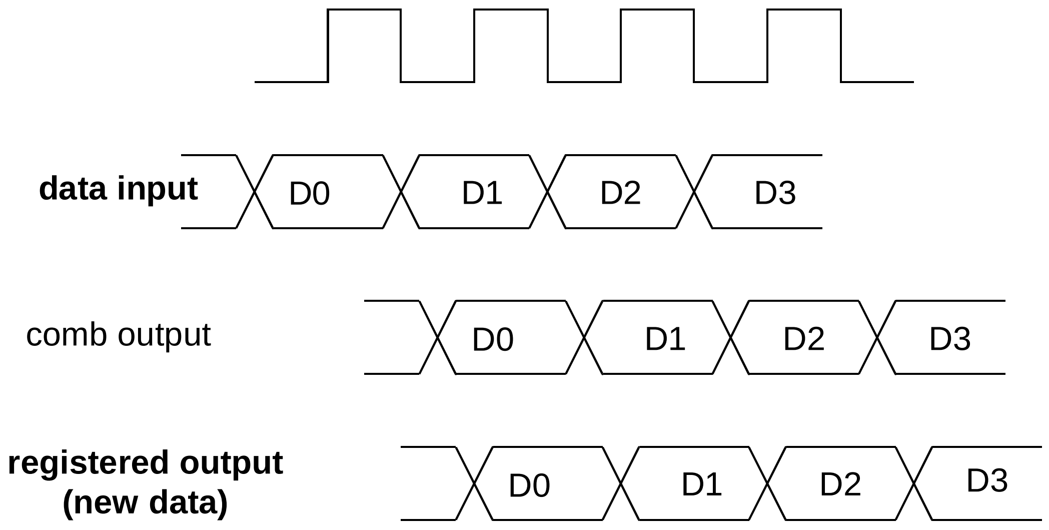
🟦 Registers external to the MLAB



LUTRAM example: read during write mode = "old data"

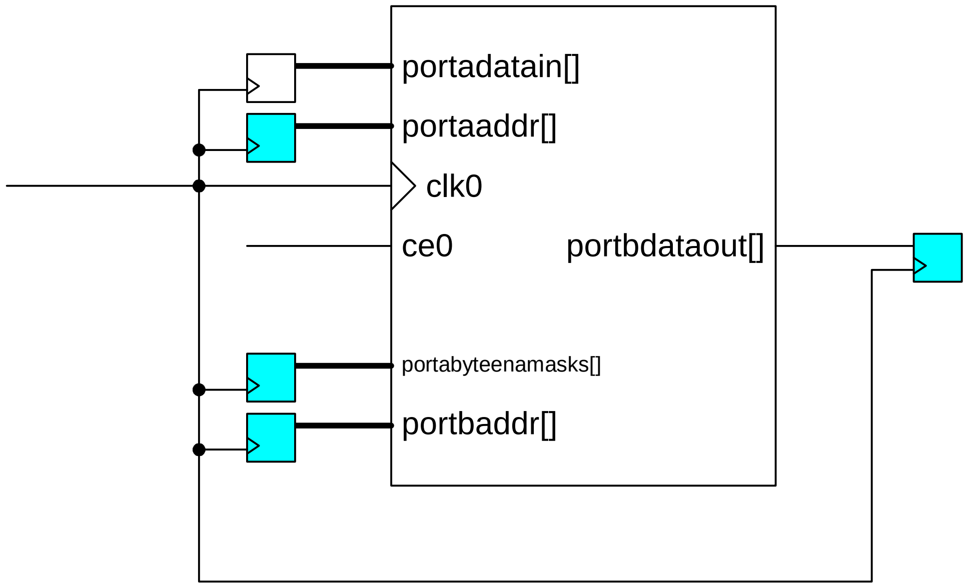
### 6.1.2 Read After Write ("New Data") Behavior

If the lutram uses a single clock to drive both read and write inputs as well as the data out registers, then we can achieve the new data behavior of block RAMs without inserting an aux. register into the netlist. See the timing model section on how this behavior is modeled to guarantee the functionality. The wave form is as follows:



The netlist would look like this: note that all these registers will be part of the netlist.

| | |
|---|---|
| ⬜ | Registers in the same MLAB |
| 🟦 | Registers external to the MLAB |



LUTRAM example: read during write mode = "new data"

## 6.2 Inferencing

The user can instantiate a LUTRAM block via the megawizard/megafunction. Alternatively, the user can rely on synthesis tools to infer a LUTRAM.

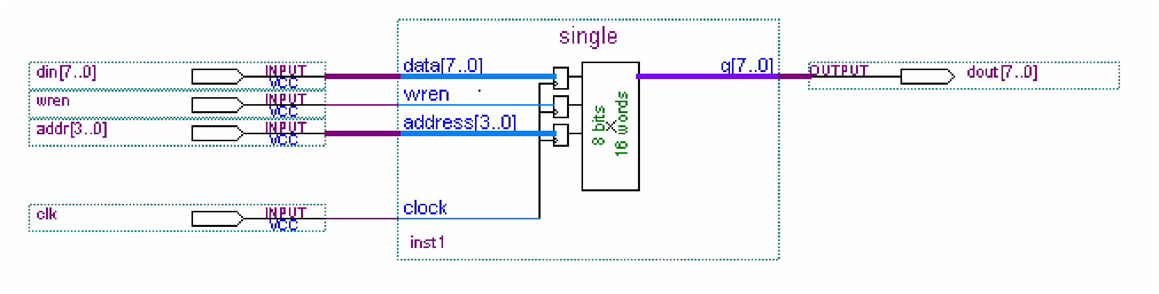
### 6.2.1 ROM

```
// Purely combinational – should not get inferred as ROM mode LUTRAM
module ROM(addr, dout);

input [3:0] addr;
output [7:0] dout;
reg [7:0] dout;

always @(addr)
        case (addr)
                4'b0000: dout = 8'b01100011;
                4'b0001: dout = 8'b11111111;
                4'b0010: dout = 8'b00001111;
                4'b0011: dout = 8'b11110000;
                4'b0100: dout = 8'b10000000;
                4'b0101: dout = 8'b00000001;
                4'b0110: dout = 8'b00001000;
                4'b0111: dout = 8'b00110000;
                4'b1000: dout = 8'b00110011;
                4'b1001: dout = 8'b11001100;
                4'b1010: dout = 8'b11010011;
                4'b1011: dout = 8'b10010001;
                4'b1100: dout = 8'b11000011;
                4'b1101: dout = 8'b10100101;
                4'b1110: dout = 8'b10101010;
                4'b1111: dout = 8'b00000000;
        endcase
endmodule
```

### 6.2.2 Single-Port RAM



```
// Single-Port RAM with unregistered output and registered addresses
// Inferred memory will have its type marked as AUTO
module single (clk, din, addr, wren, dout);

input clk, wren;
input [3:0] addr;
input [7:0] din;
output [7:0] dout;

reg [7:0] dout;
reg [7:0] mem[15:0];

always @ (posedge clk) begin
        if (wren)
                mem[addr] = din;
        dout = mem[addr];
end
endmodule
```

### 6.2.3 Simple Dual-Port

**Example 1: Single clock, read/write addresses are the same**

```
// Dual-port RAM with unregistered output.  Asynchronous read.
// Same read and write address.  Not supported by block RAM or LUTRAM.
// Will not infer a memory block
module dual1 (clk, din, addr, wren, dout);

input clk, wren;
input [3:0] addr;
input [7:0] din;
output [7:0] dout;

reg [7:0] mem[15:0];

always @ (posedge clk) begin
        if (wren)
                mem[addr] <= din;

end

assign dout = mem[addr];

endmodule
```

**Example 2: Read/write addresses are different**

```
// Dual-port RAM with unregistered output.  Asynchronous read.
// Different read and write address.  Not supported by block RAM or LUTRAM
// Will not infer a memory block
module dual2 (clk, din, raddr, waddr, wren, dout);

input clk, wren;
input [3:0] raddr, waddr;
input [7:0] din;
output [7:0] dout;

reg [7:0] mem[15:0];

always @ (posedge clk) begin
        if (wren)
                mem[waddr] <= din;

end

assign dout = mem[raddr];

endmodule
```

**Example 3: Dual clock.  Read/write addresses are different**

```
// Dual-port RAM with unregistered output
// Separate read/write clocks with separate read/write addresses
// Not supported by block ram.  Inferred memory will have its type
// marked as LUTRAM
module dual3 (rclk, rce, wclk, din, raddr, waddr, wren, dout);

input rclk, rce, wclk, wren;
input [3:0] raddr;
input [3:0] waddr;
input [7:0] din;
```

```
output [7:0] dout;

reg [3:0] raddr_reg;
reg [7:0] mem[15:0];

always @ (posedge wclk) begin
        if (wren)
                mem[waddr] <= din;
end

always @ (posedge rclk) begin
        if (rce)
                raddr_reg <= raddr;
end

assign dout = mem[raddr_reg];

endmodule
```

## 6.3 Legality Checking

LUTRAM legality checking is mostly a subset of the legality checking done on the block ram. However, one major difference is that the registers are external to the LUTRAM wysiwyg/atom and they can fan out to other places. For Stratix III, the following rules are enforced by the legality checker:

1. If write is allowed, LUTRAM write address bus width must be the same as the address width parameter
2. LUTRAM read address bus width must be the same as the address width parameter
3. If data in, write address, or byte enable inputs are used, they must be driven by registers
4. These registers must use the same clock signal as the one used by the LUTRAM primitive
5. These registers can not use asynchronous load or asynchronous clear inputs
6. Data in registers can not use the synchronous load input
7. If read during write is set to "new data", then the read address inputs must be driven by registers that use the same clock & the data out must be registered with the same clock
8. If read during write is set to "old data", then the data out must be registered with the same clock or the inverted clock
9. Quartus will use multiple LUTRAMs (depth stitch) to support LUTRAMs that are in RAM mode and use more than 4 address lines so the legality checker will error out if it sees a LUTRAM in RAM mode with more than 4 address lines
10. Quartus will use soft logic to emulate the byte enable function so it will error out if it sees a LUTRAM that uses the byte enable feature

## 6.4 Timing Model

### 6.4.1 Write Model

LUTRAM writes must be synchronized to LAB-wide clock LAB_CLK0. Figure 1 shows the waveform of the LUTRAM write signals.

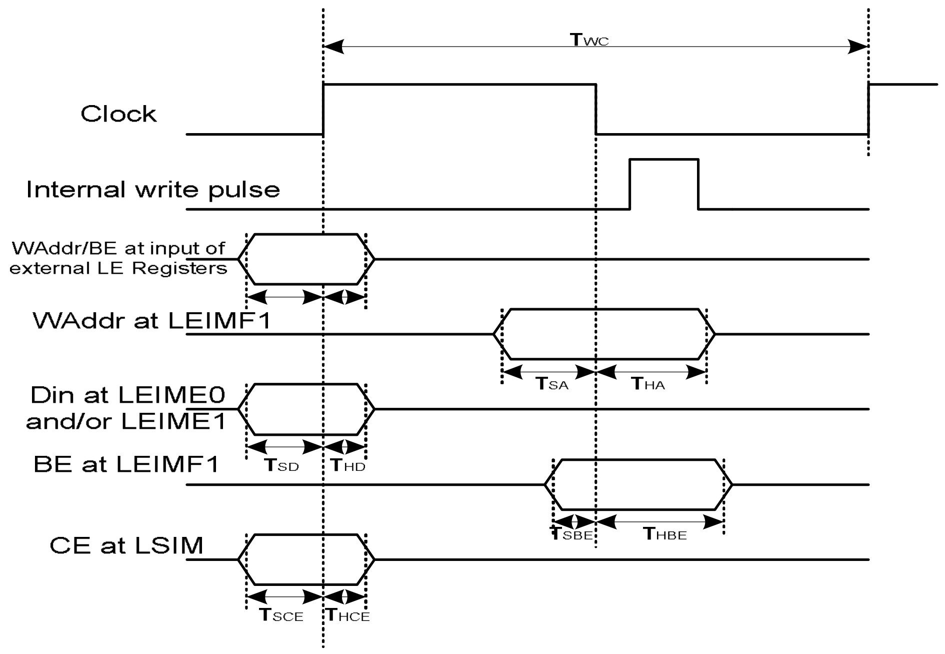


**Figure 1: Waveform for LUTRAM Write Signals**

Figure 2 shows the timing model implementation to correctly analyze the LUTRAM write timing. Any constructs (and their corresponding inputs and outputs) filled in grey are virtual and not implemented on chip. They are only created in the software timing graph to correctly model the hardware timing.

The virtual negative edge triggered BUR_WADDR and BUR_BE flip-flops with their respective observable oterms tell the timing analyzer that the write address and byte enable signals coming from external LE registers must be setup and held to the falling edge of the clock. The WADDR generation register must exist outside the MLAB (most likely in the adjacent LAB). The Write Pulse Generation logic internal to the MLAB does not latch the WADDR data during the low-phase of the clock. Therefore, the external WADDR logic must guarantee adequate hold time to the falling edge of WEN. We model this hold requirement by appropriately annotating the micro thold of the BUR_WADDR and BUR_BE virtual negative edge triggered registers. This micro thold will be the addition of the delay from the falling edge of the clock to the rising edge of the internal write pulse and the pulse width of the internal write pulse. This method will also make it easy to propagate changes in the write pulse width to the timing analysis.

The virtual positive edge triggered BUR_ENA flip flop tells the timing analyzer that the CE signal (i.e. wren in megafunction) must be setup and held to the positive edge of the clock signal. There is no virtual DATAIN flip flop or edge from the DATAIN register since the DATAIN register is always placed in the same BLE as the LUTRAM. The timing for the DATAIN register into the LUTRAM MEMORY is therefore guaranteed by design.

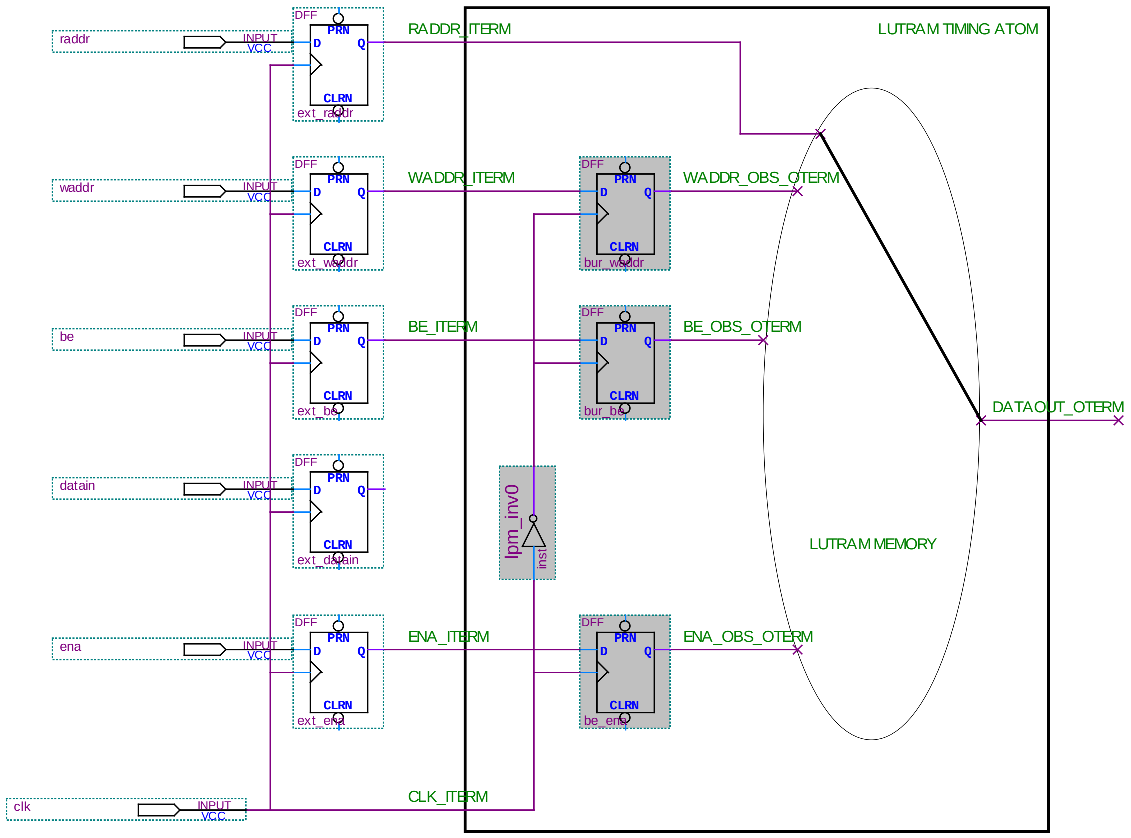


**Figure 2: Timing Model for LUTRAM Write Signals**

The WADDR, BE, ENA and DATAIN are used only to model the write signals in the LUTRAM. From a timing analysis perspective, they do not affect the read model. Therefore, the WADDR, BE and ENA observable oterms are left unconnected in the timing graph.

## 6.4.2 Read Model

Three different data paths can be used for LUTRAM reads: registered output in 64x1 mode, combinatorial read in 64x1 mode and combinatorial read in 32x1 mode. Figure 3 shows the waveform for the LUTRAM read signals.

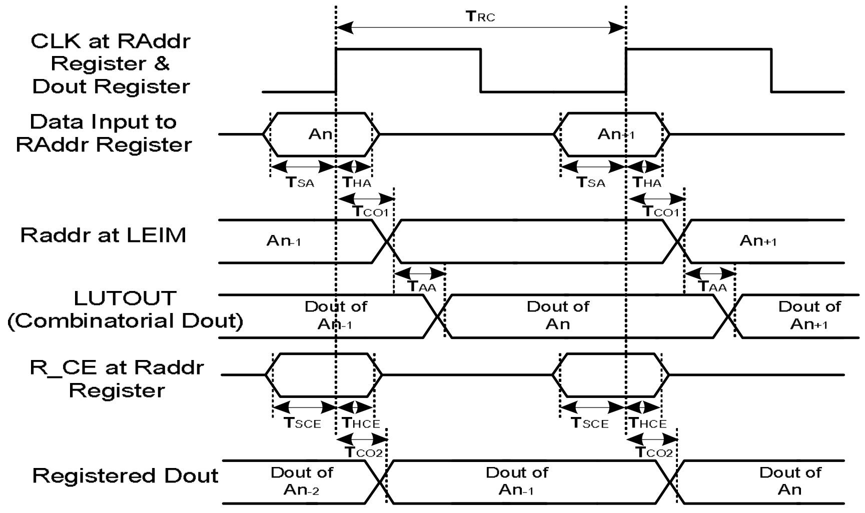


**Figure 3: Waveform for LUTRAM read signals**

Figure 4 shows the timing model implementation to correctly analyze the LUTRAM read timing. The read path through the LUTRAM TIMING ATOM is asynchronous. The edge connecting the RADDR_ITERM to the DATAOUT_OTERM will have a delay from the input to the output of the LUT.

In 64x1 mode of the ALE in an MLAB, we can use one of the ALE registers as the output register in the read data path. The other ALE register is used in the write data path. In 32x2, both ALE registers are used in the write data path. The output register must be placed as close as possible to the MLAB (e.g. in the adjacent LAB). The delay through the combinatorial logic and routing (Tcomb) must be minimized in synthesis and/or placement so that the data can meet the setup time into the output register.

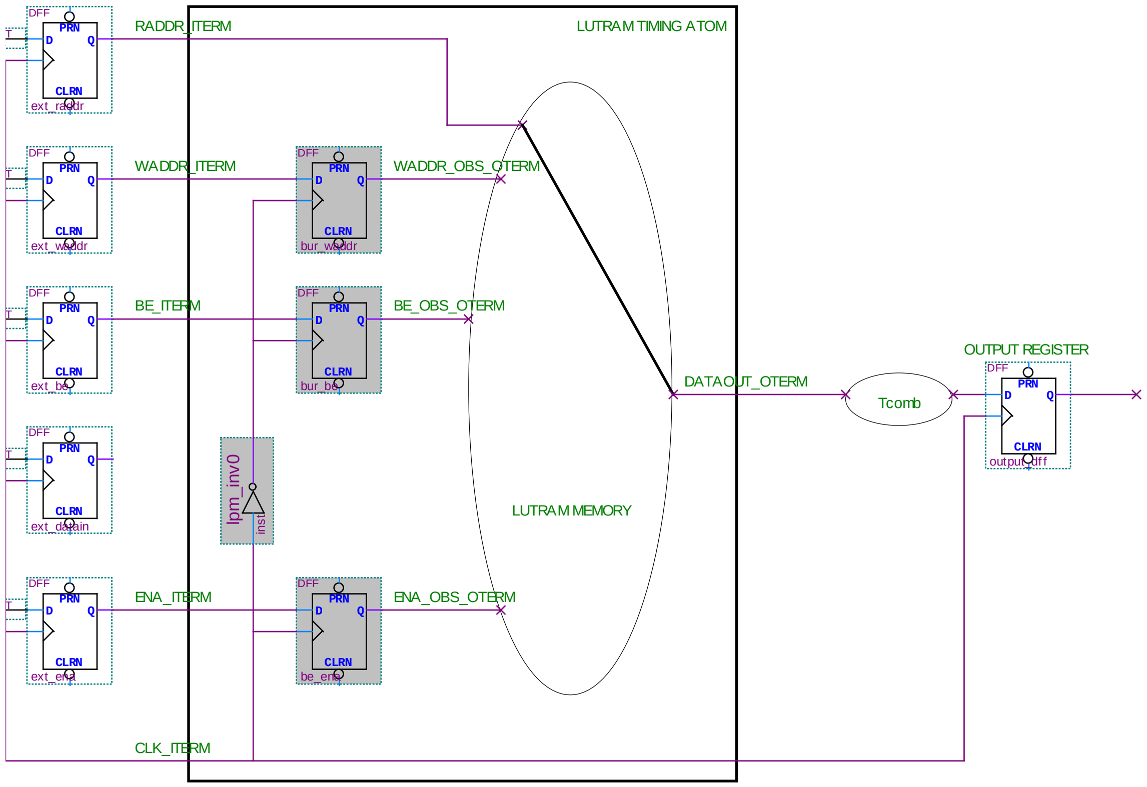


**Figure 4: Timing Model for LUTRAM read signals**

The LUTRAM allows read and write operations at the same clock cycle. However, the user must decide the read and write priority if read address and write address of the LUTRAM is identical within a given cycle. Different choice of priority requires different implementations in the timing model.

### 6.4.3 Read-before-Write (old data)

If the user wants to read old data while writing new data to the same address location in the same cycle (Read-before-Write mode), this requires the insertion of an additional auxiliary register in the data path after the LUTRAM combinatorial output. The auxiliary register is implemented on chip and should be clocked by the falling edge of the clock. This auxiliary register will capture the old data on the falling edge, so that the output register will capture this old data on the subsequent rising edge of the clock. Figure 5 shows the timing model implementation to correctly analyze the LUTRAM read-before-write (old data) timing.

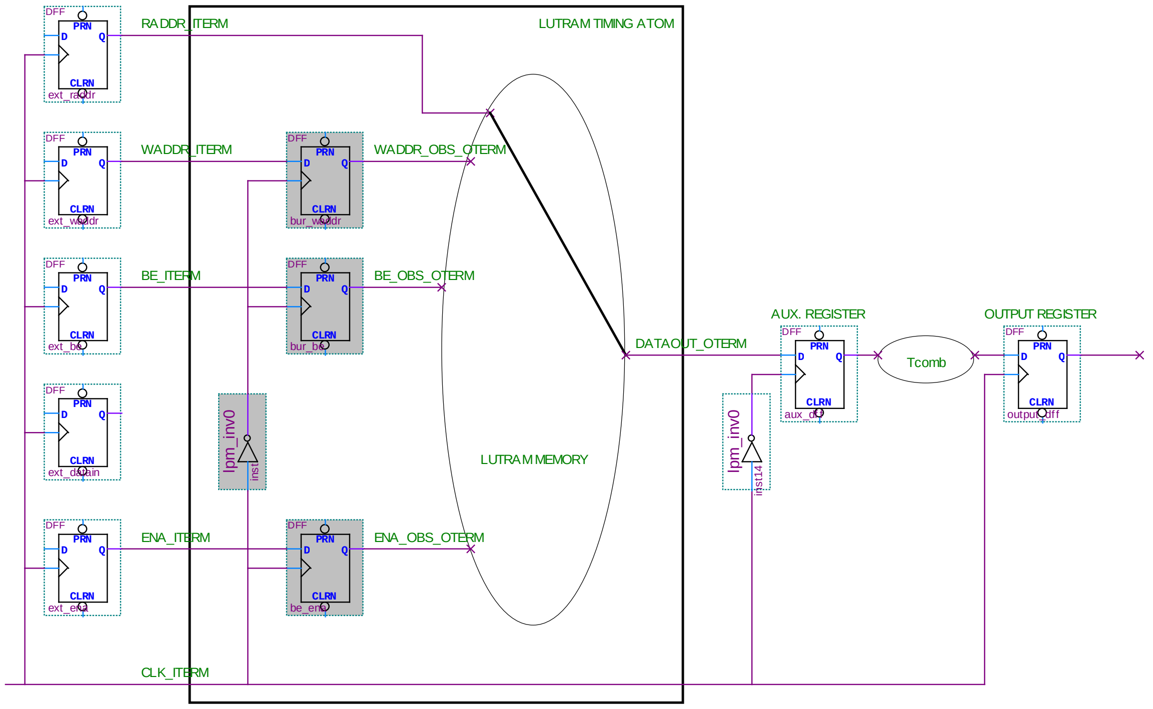


**Figure 5: Read-before-Write Timing Model**

In 64x1 mode of the ALE in an MLAB, we can use one of the ALE registers as the auxiliary register in the read data path. The other ALE register is used in the write data path. Since each register in the ALE can be clocked by two different lab wide clocks, we can use the LAB-wide Control Block to send an inverted clock to the auxiliary register of the ALEs in the same MLAB. For registered output 64x1 mode, the output register must be placed as close as possible to the MLAB (e.g. in the adjacent LAB). The delay through the combinatorial logic and routing (Tcomb) must be minimized in synthesis and/or placement so that the old data from the auxiliary register can meet the setup time into the output register.

In 32x2 mode of the ALE in an MLAB, we cannot use any of the MLAB registers as auxiliary registers in the read data path since they are all used in the write data path. The auxiliary registers must be placed outside the MLAB but as close as possible (e.g. in the adjacent LAB) to meet setup time on the falling edge of the clock.

The timing graph in software will have the additional auxiliary register edge and node between the DATA_OUT_OTERM and the output register. Adding this auxiliary register limits the read access time to half the cycle time. The frequency capability of the MLAB will be limited by the half cycle read access time to setup the auxiliary register and/or the setup constraint to the output register.

### 6.4.4 Read-after-Write (new data)

If user wants to read new data while writing new data to the same address location in the same cycle (Read-after-Write mode), no additional registers need to be added on chip. However, a soft virtual register, triggered on the falling edge of the clock, will need to be added to the timing graph in software. This virtual register has no logical implications on the circuit operation. It is used to tell the timing analyzer that new data is only available after the write access time triggered on the falling edge of the clock. This virtual register will share the same name as the lutram timing atom's DATAOUT_OTERM. Figure 6 shows the timing model implementation to correctly analyze the LUTRAM read-after-write (new data) timing.

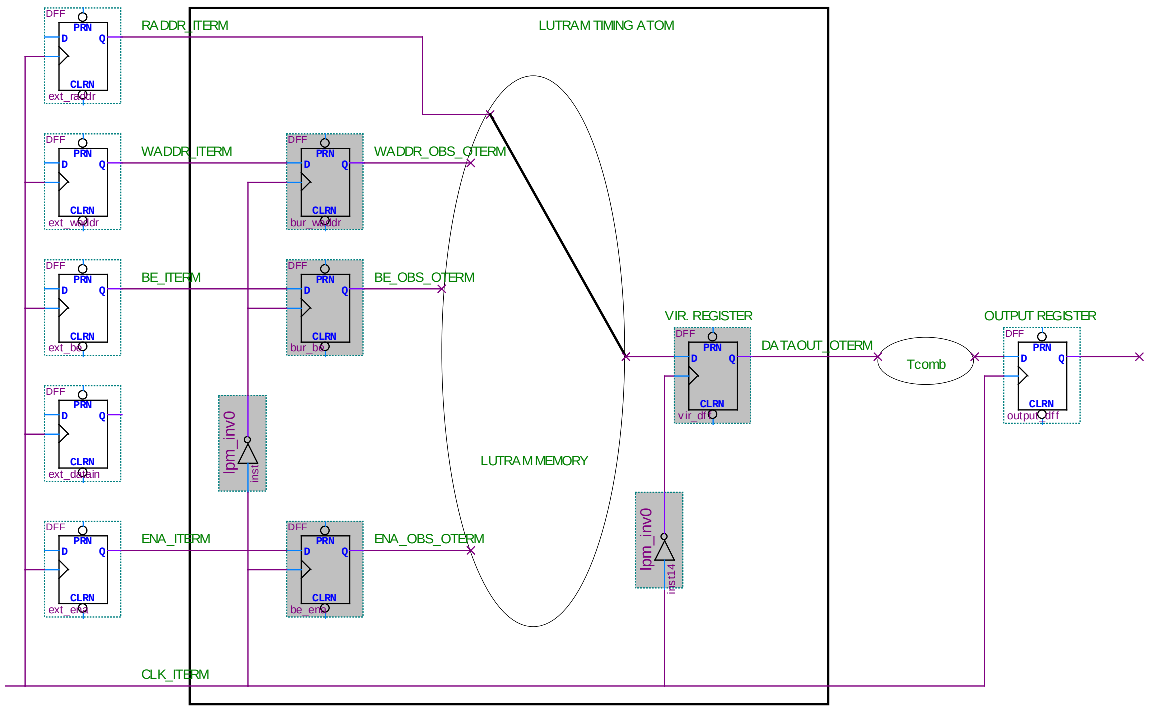


**Figure 6: Read-after-Write Timing Model**

The output register can be placed in the same MLAB for 64x1 output registered mode. For 32x2 mode, the output register should be placed and close as possible to the MLAB (e.g. in the adjacent lab. The delay Tcomb should again be minimized to meet the setup constraint to the output register. Supported LUTRAM frequencies in this mode will be limited by the time it takes to allow the half cycle write access (triggered by the falling edge of the clock), the subsequent read access and meeting the setup time into the output register.

Figure 7 shows the read-during-write waveforms.

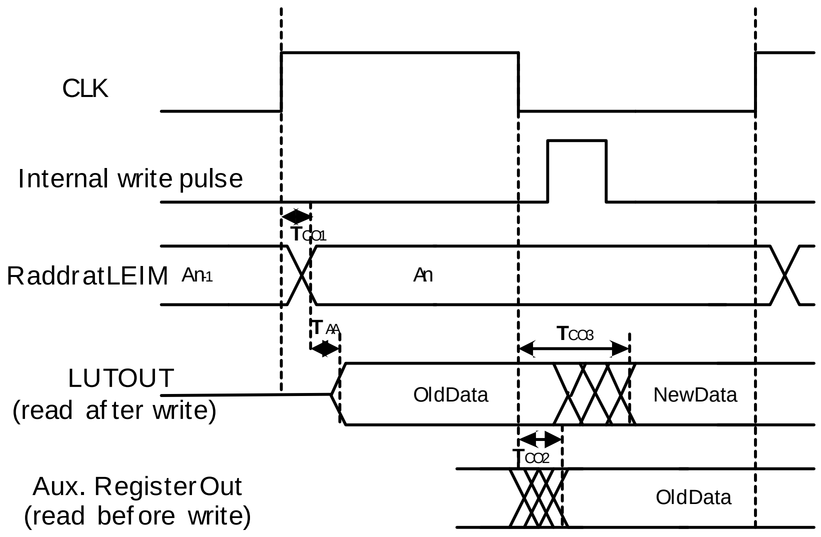


**Figure 7: Read-during-Write Waveform**

If the read during write mode is set to don’t-care, then no additional registers are needed in the netlist or timing graph.