Cyclone™ II Megafunctions Feature Functional Description

Version 1.0

May 27, 2005

By

Altera Corporation

Table of Contents:

1. Overview

This document described the changes to Quartus® II library megafunctions to support the Cyclone II device family. For most megafunctions, behavior remains unchanged compared to previous device families. No new megafunctions are planned for Cyclone II hardware support.

1.1 Summary of Changes vs. the Original Cyclone Family

Changes of interest lie in DSP-related megafunctions and RAM-based megafunctions. The changes listed in the following table are relative to the original Cyclone family, unless otherwise noted.

Category	Megafunctions	New features	New restrictions
RAM	Altsyncram	All Stratix® II M4K features supported: • Addressstall_a/b ports • Clock enable bypass parameters for port registers	None vs. Cyclone Architecture
	dcfifo	Reduced write-to-read latency vs. Cyclone architecture. Also improved performance and LE efficiency. Behavior is the same as the Stratix II architecture	None vs. Cyclone Architecture
DSP	altmult_accum altmult_add	For altmult_add, there is a new dynamic scan chain feature. For altmult_accum, there is a new wide sload port. Multipliers are now implemented in dedicated hardware. Stratix II rounding and saturation features are not supported in Cyclone II architecture.	• Single clock restriction (registers are still optional, but if present must use common clock) • Common asynchronous clear for all registers (affects all registers)
PLL	altpll	Same as Stratix II “Fast” PLL, but with fewer outputs, no dynamic phase control, and no reconfigurability	None vs. Cyclone Architecture
	altclkctrl	Clock buffer	Did not exist in Cyclone Architecture

2. Inferable Quartus II Megafunctions

Arithmetic or memory megafunctions are grouped here since they are of particular interest to synthesis tools. Unless otherwise noted, these megafunctions will have Clearbox support

2.1 Megafunctions with No Behavioral Changes Since Cyclone Device

Many megafunctions, including all LPM megafunctions, have no behavioral changes when compared with the original Cyclone family. For all of these functions, the port, parameter, and MegaWizard plug-in will remain the same.

2.1.1 altaccumulate

This is the accumulator megafunction. It is implemented using LEs. The Cyclone II behavior is identical to the original Cyclone behavior.

Parameter and port list

```
component altaccumulate
    generic (
        extra_latency        : natural := 0;
        lpm_representation   : string := "UNSIGNED";
        right_shift_distance : natural := 0; -- internal use only
        use_wys              : string := "ON";       -- no behavioral impact
        width_in             : natural;
        width_out            : natural;
        lpm_type             : string := "altaccumulate"
    );
    port(
        aclr          : in std_logic := '0';
        add_sub       : in std_logic := '1'; -- 1 means add, 0 means subtract
        cin           : in std_logic := '0';
        clken         : in std_logic := '1';
        clock         : in std_logic;
        cout          : out std_logic;
        data          : in std_logic_vector(width_in-1 downto 0);
        overflow      : out std_logic;
        result        : out std_logic_vector(width_out-1 downto 0);
        sign_data     : in std_logic := '0';  -- to dynamically change signs
        sload         : in std_logic := '0'
    );
end component;
```

Parameter definitions

right_shift_distance: 0 means normal accumulator behavior. Non-zero values will result in shift toward the LSB of the given number of bits on each clock cycle. This is especially useful as a building block for multi-cycle shift-and-accumulate multipliers. This parameter is for clearbox internal use and it is not exposed through the megawizard or the megafunction interface.

use_wys: For some device families, chooses an implementation based on LE WYSIWYGs, rather than the alternate carry_sum-based implementation. Ignored for Cyclone II architecture.

Port definitions

sign_data: 1 is signed 0 is unsigned. LPM_REPRESENTATION should be unused when sign_data is used.

cout: Carry-out of the accumulator. This is an unregistered output (i.e. it changes between clock edges). Useful for pipelining and chaining multiple accumulators.

2.1.2 altshift_taps

This is the RAM-based shift register with taps megafunction. 'Taps' means that data from the shift register is observed only at certain points only the shift-register chain. The tap points must be exactly evenly spaced (see **tap_distance** parameter). All device families with simple dual-port RAM are supported.

Parameter and port list

```
component altshift_taps
    generic (
        number_of_taps    : natural;
        power_up_state    : string := "CLEARED"; -- "CLEARED" , "DONT_CARE"
        tap_distance      : natural;
        width             : natural;
        lpm_type          : string := "altshift_taps"
    );
    port(
        clken    : in std_logic := '1';
        clock    : in std_logic;
        shiftin  : in std_logic_vector(width-1 downto 0);
        shiftout : out std_logic_vector(width-1 downto 0);
        taps     : out std_logic_vector(width*number_of_taps-1 downto 0)
    );
end component;
```

Parameter definitions

number_of_taps: The number of evenly spaced points in the shift register where intermediate states are visible. In terms of the RAM-based implementation, this parameter corresponds to the number of RAM bits per word that will be used.

tap_distance: Spacing between taps in shift positions (i.e. clock cycles). In terms of the RAM-based implementation, this parameter corresponds to the number of RAM words that will be used.

width: The number of bits per position in the shift register. This can also be described as the number of parallel single-bit shift registers.

power_up_state: Controls whether the shift register powers up with zero contents ("CLEARED") or with unknown contents ("DONT_CARE"). The value "CLEARED", which is the default, forces the use of M512 or M4K RAM blocks in Cyclone II and original Cyclone devices. "DONT_CARE" allows the use of MegaRAM blocks.

2.1.2.1 Megafunction implementation

The altshift_taps megafunction implementation was changed in the Quartus II software version 3.0 to ensure that the read and write pointers are always in sync, even if some parts of a chip see Power-On-Reset for longer periods than other parts of the chip. Read and write addresses are now derived from a single counter. For the Cyclone II and Cyclone device families, the simple dual port RAM will return the OLD_DATA in Simple dual port mode for M4K blocks if read and write address are the same. This avoids the use of two counters as in the previous altshift_taps implementation. For pre-Stratix families, an adder circuit that powers up at 2 is used to feed the read address.

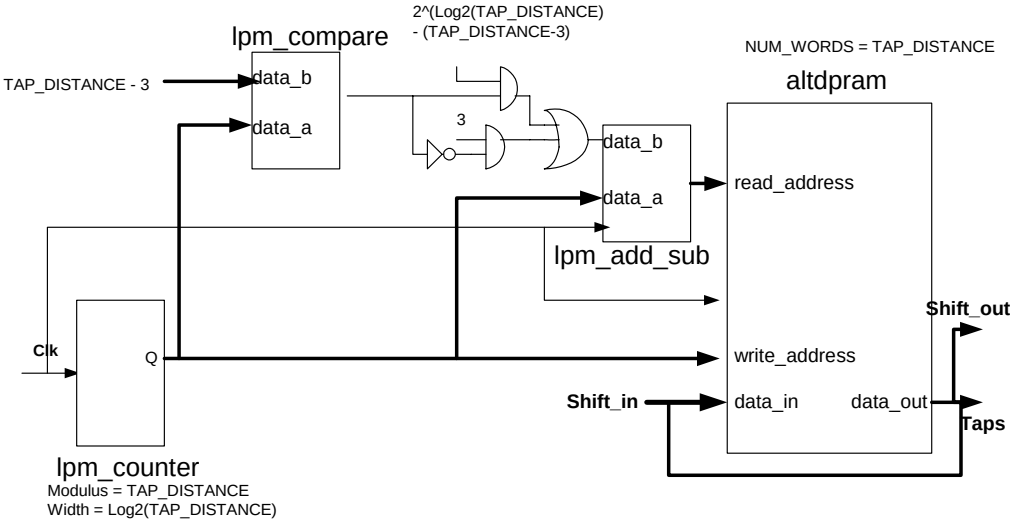


Figure 2: Implementation of altshift_taps for pre-Stratix families

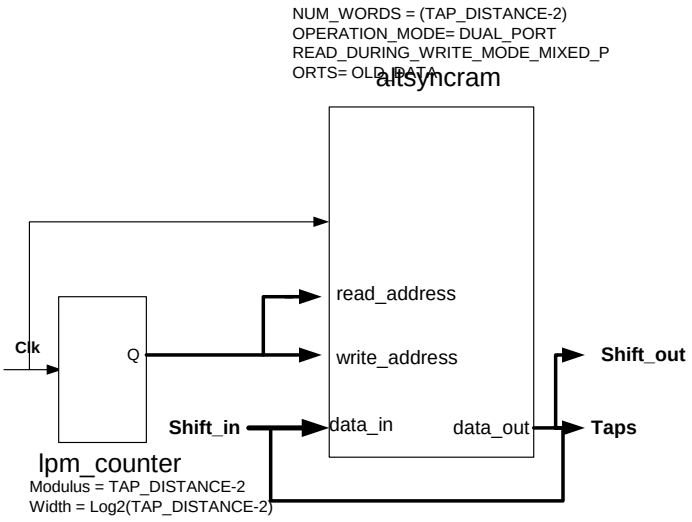


Figure 3: Implementation of altshift_taps for Cyclone II and Cyclone families

2.1.3 lpm_add_sub

This LE-based adder/subtractor megafunction consists of “+” and “-“ operators, which really means that this logic will be left to the synthesis tool.

2.1.4 lpm_compare

This LE-based comparator megafunction consists of HDL-native comparison operators such as “>” and “=“. This means that this logic will be left to the synthesis tool.

2.1.5 lpm_counter

This LE-based counter megafunction will be implemented using Cyclone II WYSIWYG primitives.

2.1.6 lpm_divide

This LE-based divider megafunction remain the same as in original Cyclone architecture.

2.1.7 lpm_mult

This is the basic multiplier megafunction. It supports DSP Block-based implementations and LE-based implementations. The behavior is unchanged from previous device families, regardless of the implementation.

2.2 altmult_accum

This is the multiply-accumulate megafunction. From a behavioral perspective, it consists of a single multiplier feeding an accumulator. In the Cyclone II device family, only the multipliers are implemented in dedicated hardware. The accumulator part is always implemented in LEs. Compared to the Stratix architecture, only a single clock is supported, and if the aclr feature is used, it must affect all registers.

New features in the Cyclone II architecture include an input for upper data bits when loading the accumulator, and scanin ports for use with the dynamic scan-chains. Dynamic scan chains means that the input source for both A and B inputs can be selected dynamically. There are new parameters, INPUT_SOURCE_A and INPUT_SOURCE_B, with values of “DATAA”, “SCANA”, or “VARIABLE”, and “DATAB”, “SCANB”, or “VARIABLE” respectively, to select the multiplier input source. Stratix II rounding and saturation features are not supported.

2.2.1 MegaWizard plug-in

The MegaWizard plug-in will enforce the Cyclone II feature subset.

2.2.2 Parameter and port list

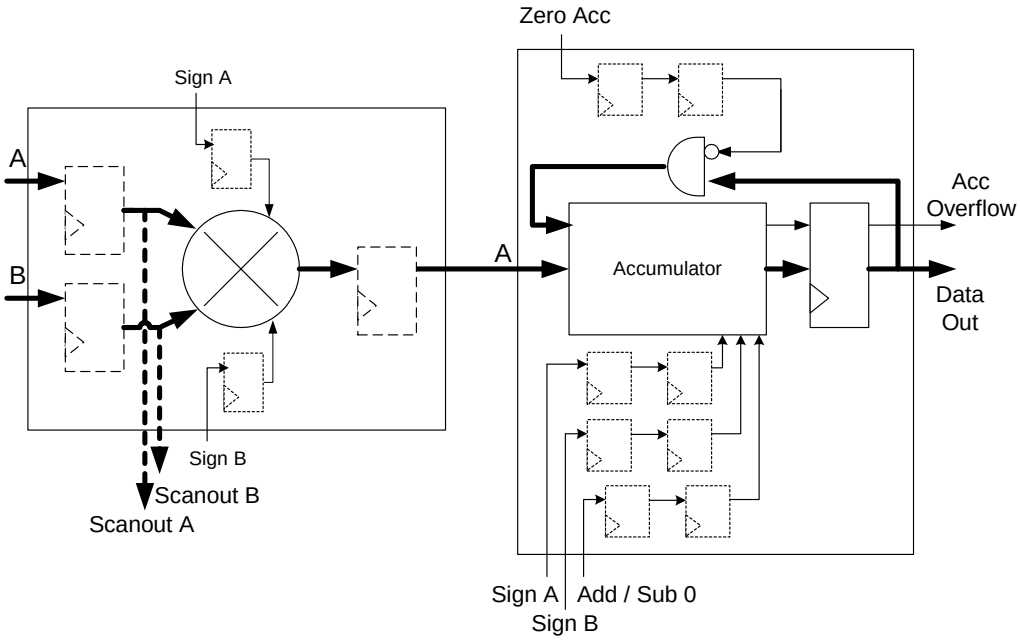


Figure 2-1: This diagram taken from the Stratix MAC Feature FD gives a good view of the parameter options.

```
component altmult_accum
    generic (
       accum_direction                : string := "add";
       accum_round_aclr               : string := "aclr3";
       accum_round_pipeline_aclr      : string := "aclr3";
       accum_round_pipeline_reg       : string := "clock0";
       accum_round_reg                : string := "clock0";
       accum_saturation_aclr          : string := "aclr3";
       accum_saturation_pipeline_aclr : string := "aclr3";
       accum_saturation_pipeline_reg  : string := "clock0";
       accum_saturation_reg           : string := "clock0";
       accum_sload_aclr               : string := "aclr3";
       accum_sload_pipeline_aclr      : string := "aclr3";
       accum_sload_pipeline_reg       : string := "clock0";
       accum_sload_reg                : string := "clock0";
       accum_sload_upper_data_aclr    : string := "aclr3";
       accum_sload_upper_data_pipeline_aclr   : string := "aclr3";
       accum_sload_upper_data_pipeline_reg    : string := "clock0";
       accum_sload_upper_data_reg     : string := "clock0";
       accumulator_rounding           : string := "no";
       accumulator_saturation         : string := "no";
       addnsub_aclr                   : string := "aclr3";
       addnsub_pipeline_aclr          : string := "aclr3";
       addnsub_pipeline_reg           : string := "clock0";
       addnsub_reg                    : string := "clock0";
       dedicated_multiplier_circuitry : string := "auto";
       dsp_block_balancing            : string := "UNUSED";
       extra_accumulator_latency      : natural := 0;
       extra_multiplier_latency       : natural := 0;
       input_aclr_a                   : string := "aclr3";
```

```
        input_aclr_b                : string := "aclr3";
        input_reg_a                 : string := "clock0";
        input_reg_b                 : string := "clock0";
        input_source_a              : string := "dataa";
        input_source_b              : string := "datab";
        mult_round_aclr             : string := "aclr3";
        mult_round_reg              : string := "clock0";
        mult_saturation_aclr        : string := "aclr3";
        mult_saturation_reg         : string := "clock0";
        multiplier_aclr             : string := "aclr3";
        multiplier_reg              : string := "clock0";
        multiplier_rounding         : string := "no";
        multiplier_saturation       : string := "no";
        output_aclr                 : string := "aclr3";
        output_reg                  : string := "clock0";
        port_accum_is_saturated     : string := "unused";
        port_mult_is_saturated      : string := "unused";
        representation_a            : string := "UNSIGNED";
        representation_b            : string := "UNSIGNED";
        sign_aclr_a                 : string := "aclr3";
        sign_aclr_b                 : string := "aclr3";
        sign_pipeline_aclr_a        : string := "aclr3";
        sign_pipeline_aclr_b        : string := "aclr3";
        sign_pipeline_reg_a         : string := "clock0";
        sign_pipeline_reg_b         : string := "clock0";
        sign_reg_a                  : string := "clock0";
        sign_reg_b                  : string := "clock0";
        width_a                     : natural;
        width_b                     : natural;
        width_result                : natural;
        width_upper_data            : natural := 1;
        lpm_type                    : string := "altmult_accum"
    );
    port(
        accum_is_saturated     : out std_logic;
        accum_round            : in std_logic := '0';
        accum_saturation       : in std_logic := '0';
        accum_sload            : in std_logic := '0';
        accum_sload_upper_data : in std_logic_vector(width_upper_data-1
                                     downto 0) := (others => '0');
        aclr0                  : in std_logic := '0';
        aclr1                  : in std_logic := '0';
        aclr2                  : in std_logic := '0';
        aclr3                  : in std_logic := '0';
        addnsub                : in std_logic := '1';
        clock0                 : in std_logic := '1';
        clock1                 : in std_logic := '1';
        clock2                 : in std_logic := '1';
        clock3                 : in std_logic := '1';
        dataa                  : in std_logic_vector(width_a-1 downto 0)
                                     := (others => '0');
        datab                  : in std_logic_vector(width_b-1 downto 0)
                                     := (others => '0');
        ena0                   : in std_logic := '1';
        ena1                   : in std_logic := '1';
        ena2                   : in std_logic := '1';
        ena3                   : in std_logic := '1';
```

```
        mult_is_saturated      : out std_logic;
        mult_round             : in std_logic := '0';
        mult_saturation        : in std_logic := '0';
        overflow               : out std_logic;
        result          : out std_logic_vector(width_result-1 downto 0);
        scanina         : in std_logic_vector(width_a-1 downto 0)
                                    := (others => '0');
        scaninb         : in std_logic_vector(width_b-1 downto 0)
                                    := (others => '0');
        scanouta        : out std_logic_vector(width_a-1 downto 0);
        scanoutb        : out std_logic_vector(width_b-1 downto 0);
        signa           : in std_logic := '0';
        signb           : in std_logic := '0';
        sourcea         : in std_logic := '0';
        sourceb         : in std_logic := '0'
    );
end component;
```

2.2.3 Parameter definitions

WIDTH_A: Width of the dataa input bus.

WIDTH_B: Width of the datab input bus.

WIDTH_RESULT: Width of the result output bus.

WIDTH_UPPER_DATA: Width of the accum_sload_upper_data input bus.

INPUT_SOURCE_A: Selects the input source for the dataa input. Default is “DATAA” for Stratix compatibility. Legal values are “DATAA”, “SCANA”, and “VARIABLE”. “VARIABLE” means that the source will be dynamically selected by the *sourcea* input.

INPUT_SOURCE_B: Selects the input source for the datab input. Default is “DATAB” for Stratix compatibility. Legal values are “DATAB”, “SCANB”, and “VARIABLE”. “VARIABLE” means that the source will be dynamically selected by the *sourceb* input.

INPUT_REG_A: Clock source for the dataa registers. Legal values are “UNREGISTERED” and “CLOCK0”

INPUT_ACLR_A: Asynchronous clear source for the dataa registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

INPUT_REG_B: Clock source for the datab registers. Legal values are “UNREGISTERED”, “CLOCK0”, “CLOCK1”, and “CLOCK2”.

INPUT_ACLR_B: Clear source for the datab input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ADDNSUB_REG: Clock source for the register on the addnsub input port. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_ACLR: Asynchronous clear source for the addnsub input port. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ADDNSUB_PIPELINE_REG: Clock source for the second register on the addnsub input port. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_PIPELINE_ACLR: Asynchronous clear source for the second register on the addnsub input port. Legal Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_DIRECTION: Used to specify whether the accumulator will be incrementing or decrementing. When set to “ADD”, the accumulator will add the product to the present accumulator value. When set to “SUB”, the accumulator will subtract the product from the present accumulator value. When this parameter is set to “UNUSED”, the choice of whether to add or subtract is controlled dynamically through the addnsub port. It is illegal to set the ACCUM_DIRECTION to “ADD” or “SUB” and connect the addnsub port.

ACCUM_SLOAD_REG: Clock source for the accum_sload port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

ACCUM_SLOAD_ACLR: Asynchronous clear source for the accum_sload input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_SLOAD_PIPELINE_REG: Clock source for the second stage registers on the accum_sload input port. Legal values are “UNREGISTERED” and “CLOCK0”

ACCUM_SLOAD_PIPELINE_ACLR: Asynchronous clear source for the second stage registers on the accum_sload input port. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULT_SATURATION_REG: Clock source for the mult_saturation port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

MULT_ SATURATION _ACLR: Asynchronous clear source for the mult_saturation input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULT_ROUND_REG: Clock source for the mult_round port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

MULT_ ROUND _ACLR: Asynchronous clear source for the mult_round input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_SATURATION_REG: Clock source for the accum_saturation port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

ACCUM_ SATURATION _ACLR: Asynchronous clear source for the accum_saturation input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_ SATURATION _PIPELINE_REG: Clock source for the second stage registers on the accum_saturation input port. Legal values are “UNREGISTERED” and “CLOCK0”

ACCUM_ SATURATION _PIPELINE_ACLR: Asynchronous clear source for the second stage registers on the accum_saturation input port. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_ROUND_REG: Clock source for the accum_round port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

ACCUM_ ROUND _ACLR: Asynchronous clear source for the accum_round input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_ ROUND _PIPELINE_REG: Clock source for the second stage registers on the accum_round input port. Legal values are “UNREGISTERED” and “CLOCK0”

ACCUM_ ROUND _PIPELINE_ACLR: Asynchronous clear source for the second stage registers on the accum_round input port. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_SLOAD_UPPER_DATA_REG: Clock source for the accum_sload_upper_data port input registers. Legal values are “UNREGISTERED” and “CLOCK0”.

ACCUM_SLOAD_ UPPER_DATA_ACLR: Asynchronous clear source for the accum_sload_upper_data input registers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ACCUM_SLOAD_ UPPER_DATA_PIPELINE_REG: Clock source for the second stage registers on the accum_sload_upper_data input port. Legal values are “UNREGISTERED” and “CLOCK0”

ACCUM_SLOAD_ UPPER_DATA_PIPELINE_ACLR: Asynchronous clear source for the second stage registers on the accum_sload_upper_data input port. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

REPRESENTATION_A: For specifying the number representation of the A input. Legal values are “UNSIGNED”, “SIGNED”, and “UNUSED”. A value of “UNSIGNED” causes the accumulator to interpret the A input as an unsigned. A value of ”SIGNED” causes the accumulator to interpret the A input as a signed two’s complement. A value of “UNUSED” allows for dynamic control of the representation through the signa port.

SIGN_REG_A: Clock source for first register on the signa signal. Legal values are “UNREGISTERED” and “CLOCK0”.

SIGN_ACLR_A: Asynchronous clear signal for the first register on the signa signal. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

SIGN_PIPELINE_REG_A: Clock source for the second register on the signa signal. Legal values are “UNREGISTERED” and “CLOCK0”.

SIGN_PIPELINE_ACLR_A: Asynchronous clear source the second register on the signa signal. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

REPRESENTATION_B: For specifying the number representation of the B input. Legal values are “UNSIGNED”, “SIGNED”, and “UNUSED”. A value of “UNSIGNED” causes the accumulator to interpret the B input as an unsigned. A value of ”SIGNED” causes the accumulator to interpret the B input as a signed two’s complement. A value of “UNUSED” allows for dynamic control of the representation through the signb port.

SIGN_REG_B: Clock source for first register on the signb signal. Legal values are “UNREGISTERED” and “CLOCK0”.

SIGN_ACLR_B: Asynchronous clear signal for the first register on the signb signal. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

SIGN_PIPELINE_REG_B: Clock source for the second register on the signb signal. Legal values are “UNREGISTERED” and “CLOCK0”.

SIGN_PIPELINE_ACLR_B: Asynchronous clear source the second register on the signb signal. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULTIPLIER_REG: Clock source for the register immediately after the multiplier stage. Legal values are “UNREGISTERED” and “CLOCK0”.

MULTIPLIER_ACLR: Asynchronous clear source for the register immediately after the multiplier stage. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

OUTPUT_REG: Clock source for the registers on the outputs. Legal value is “CLOCK0”.

OUTPUT_ACLR: Asynchronous clear source for the registers on the outputs. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

EXTRA_MULTIPLIER_LATENCY: Used to add latency to the multiplier portion of the circuit. If the MULTIPLIER_REG parameter specifies a clock, then that clock will be used to add the latency. If the MULTIPLIER_REG parameter is set to UNREGISTERED then clock0 will be used to add the pipelining. The extra registers are added before the multiplier output register.

EXTRA_ACCUMULATOR_LATENCY: Used to add latency to the accumulator portion of the circuit. This latency will be from the same clock as specified in the OUTPUT_REG parameter.

DEDICATED_MULTIPLIER_CIRCUITRY: Specifies whether or not to use the DSP block to implement the circuit. Legal values are "YES" , "NO", and "AUTO". A value of "YES" causes the circuit to be implemented using the DSP block.

MULTIPLIER_ROUNDING: Enable rounding at the output of the multiplier stage. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is "NO" in Cyclone II devices.

MULTIPLIER_SATURATION: Enable saturation handling at the output of the multiplier stage. Useful only for signed multiplication. Saturation at the output of the multiplier can only occur if dataa and datab are both the maximum negative number. In this case, the product is a positive that is too large to be represented in the given number of bits. Turning on saturation handling will give a maximum positive result for this case. Legal value is "NO" in Cyclone II devices.

ACCUMULATOR_ROUNDING: Enable rounding in the accumulator. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is "NO" in Cyclone II devices.

ACCUMULATOR _SATURATION: Enable saturation handling in the accumulator. Saturation should only be used with input that conforms to signed 1.(x) fractional number representation, such as signed 1.15 or 1.30 representation. Turning on saturation handling will give a maximum positive or negative result in overflow situations. Legal value is "NO" in Cyclone II devices.

PORT_MULT_IS_SATURATED: Indicates that the *mult_is_saturated* output should be used or unused. Legal value is "UNUSED" in Cyclone II devices. If the value is "UNUSED", the *mult_is_saturated* output will be stuck at GND.

PORT_ACCUM_IS_SATURATED: Indicates that the *accum_is_saturated* output should be used or unused. Legal value is "UNUSED" in Cyclone II devices. If the value is "UNUSED", the *mult_is_saturated* output will be stuck at GND.

INTENDED_DEVICE_FAMILY: Defaults to "Stratix". Normally, this parameter would only be used for behavioral models, since megafunctions look at the Quartus II-defined device family. In this case, it is used to select Stratix-compatible assumptions about port and parameter priority resolution. In the original Stratix version of altmult_accum, dynamic control inputs such as signa, when connected, took precedence over the value of conflicting parameters, in this case REPRESENTATION_A. Going forward, we prefer that all behavioral information be taken from parameter and port values, and not from connectivity information, since this requires non-standard Verilog and VHDL instantiations. The MegaWizard plug-in for altmult_accum will always write a value for this parameter starting in the Quartus II software version 4.0. The megafunction will check to see if a value is assigned, and if none is assigned, it will revert to pre-Quartus II 4.0 port/parameter priority rules.

2.2.4 Port definitions

dataa: Bus for one of the two multiplier arguments.

datab: Bus for one of the two multiplier arguments.

addnsub: Allows for dynamic control of whether the adder will perform an add or a subtract. A high value causes an add; a low value causes a subtract.

accum_sload: This port is for causing the present value of the accumulator to go to zero. For example, if the accum_sload signal is high, then the next value stored on the outputs will be the multiplier output (assuming MULTIPLIER_REG is "UNREGISTERED"; otherwise the value in the accumulator will be the output of the multiplier registers). For accum_sload to behave as a true synchronous load, the accumulator must be in addition mode, either via the ACCUM_DIRECTION parameter, or by setting the addnsub input to '1' for addition. In Stratix II devices, the accum_sload_upper_data[] port is also added to the accumulator value. Combining the accum_sload_upper_data[] port with the multiplier output allows all bits of the accumulator to be loaded with a new value.

signa, signb: These are for dynamically specifying the number representation of the dataa and datab ports. A high value on signa/b causes the multiplier to treat dataa/b as a signed two's complement. A low value on signa/b causes the multiplier to treat dataa/b as an unsigned.

scanina: The optional input to the A scan chain when INPUT_SOURCE_A is "SCANA" or "VARIABLE".

scaninb: The optional input to the B scan chain when INPUT_SOURCE_B is "SCANB" or "VARIABLE".

mult_round: Enables multiplier stage rounding when MULTIPLIER_ROUNDING = "VARIABLE".

mult_saturate: Enables multiplier stage saturation handling when MULTIPLIER_SATURATION = "VARIABLE".

accum_round: Enables accumulator rounding when ACCUMULATOR_ROUNDING = "VARIABLE".

accum_saturate: Enables accumulator saturation handling when ACCUMULATOR _SATURATION = "VARIABLE".

accum_sload_upper_data: The input for accumulator upper data bits during a synchronous load, for Stratix II devices only. This port defaults to all '0' to give Stratix-compatible behavior.

clock0, clock1, clock2, clock3: These are the four clock inputs.

ena0, ena1, ena2, ena3: ena0, ena1, ena2, ena3 are the respective clock enables for clock0, clock1, clock2, and clock3.

aclr0, aclr1, aclr2, alcr3: These are the four asynchronous clear sources.

result: The accumulator output.

overflow: Overflow flag for the accumulator adder.

scanouta: The output of the A scan chain.

scanoutb: The output of the B scan chain.

mult_is_saturated: Indicates that saturation handling has occured. Must be explicitly enabled with the parameter PORT_MULT_IS_SATURATED = "USED".

accum_is_saturated: Indicates that saturation handling has occured. Must be explicitly enabled with the parameter PORT_ACCUM_IS_SATURATED = "USED".

2.3 altmult_add

This is the multiply-add megafunction. From a behavioral perspective, it consists of 1 or more multipliers feeding a parallel adder. In the Cyclone II device family, only the multipliers are

implemented in dedicated hardware. The adder part is always implemented in LEs. Compared to Stratix devices, only a single clock is supported, and if the aclr feature is used, it must affect all registers.

New in the Cyclone II family are scanin ports for use with the dynamic scan-chains. Dynamic scan chains means that the input source for both A and B inputs can be selected dynamically. The parameters INPUT_SOURCE_A0/1/2/3 and INPUT_SOURCE_B0/1/2/3, will support a new value of “VARIABLE” for Stratix II and Cyclone II devices, in addition to the previously support values of “DATAA” or “SCANA”, and “DATAB” or “SCANB” respectively, to select the multiplier input source. Stratix II rounding and saturation features are not supported.

2.3.1 MegaWizard plug-in

MegaWizard plug-in updates will be needed to support new Stratix II features.

2.3.2 Parameter and port list

```
component altmult_add
    generic (
       adder1_rounding                      : string := "NO";
       adder3_rounding                      : string := "NO";
       addnsub1_round_aclr                  : string := "ACLR3";
       addnsub1_round_pipeline_aclr         : string := "ACLR3";
       addnsub1_round_pipeline_register     : string := "CLOCK0";
       addnsub1_round_register              : string := "CLOCK0";
       addnsub3_round_aclr                  : string := "ACLR3";
       addnsub3_round_pipeline_aclr         : string := "ACLR3";
       addnsub3_round_pipeline_register     : string := "CLOCK0";
       addnsub3_round_register              : string := "CLOCK0";
       addnsub_multiplier_aclr1             : string := "ACLR3";
       addnsub_multiplier_aclr3             : string := "ACLR3";
       addnsub_multiplier_pipeline_aclr1    : string := "ACLR3";
       addnsub_multiplier_pipeline_aclr3    : string := "ACLR3";
       addnsub_multiplier_pipeline_register1 : string := "CLOCK0";
       addnsub_multiplier_pipeline_register3 : string := "CLOCK0";
       addnsub_multiplier_register1         : string := "CLOCK0";
       addnsub_multiplier_register3         : string := "CLOCK0";
       dedicated_multiplier_circuitry       : string := "AUTO";
       dsp_block_balancing                  : string := "Auto";
       input_aclr_a0          : string := "ACLR3";
       input_aclr_a1          : string := "ACLR3";
       input_aclr_a2          : string := "ACLR3";
       input_aclr_a3          : string := "ACLR3";
       input_aclr_b0          : string := "ACLR3";
       input_aclr_b1          : string := "ACLR3";
       input_aclr_b2          : string := "ACLR3";
       input_aclr_b3          : string := "ACLR3";
       input_register_a0      : string := "CLOCK0";
       input_register_a1      : string := "CLOCK0";
       input_register_a2      : string := "CLOCK0";
       input_register_a3      : string := "CLOCK0";
       input_register_b0      : string := "CLOCK0";
       input_register_b1      : string := "CLOCK0";
       input_register_b2      : string := "CLOCK0";
       input_register_b3            : string := "CLOCK0";
       input_source_a0              : string := "DATAA";
       input_source_a1              : string := "DATAA";
```

```
    input_source_a2              : string := "DATAA";
    input_source_a3              : string := "DATAA";
    input_source_b0              : string := "DATAB";
    input_source_b1              : string := "DATAB";
    input_source_b2              : string := "DATAB";
    input_source_b3              : string := "DATAB";
    mult01_round_aclr            : string := "ACLR3";
    mult01_round_register        : string := "CLOCK0";
    mult01_saturation_aclr       : string := "ACLR2";
    mult01_saturation_register   : string := "CLOCK0";
    mult23_round_aclr            : string := "ACLR3";
    mult23_round_register        : string := "CLOCK0";
    mult23_saturation_aclr       : string := "ACLR3";
    mult23_saturation_register   : string := "CLOCK0";
    multiplier01_rounding        : string := "NO";
    multiplier01_saturation      : string := "NO";
    multiplier1_direction        : string := "ADD";
    multiplier23_rounding        : string := "NO";
    multiplier23_saturation      : string := "NO";
    multiplier3_direction        : string := "ADD";
    multiplier_aclr0             : string := "ACLR3";
    multiplier_aclr1             : string := "ACLR3";
    multiplier_aclr2             : string := "ACLR3";
    multiplier_aclr3             : string := "ACLR3";
    multiplier_register0         : string := "CLOCK0";
    multiplier_register1         : string := "CLOCK0";
    multiplier_register2         : string := "CLOCK0";
    multiplier_register3         : string := "CLOCK0";
    number_of_multipliers        : natural;
    output_aclr                  : string := "ACLR3";
    output_register              : string := "CLOCK0";
    port_mult0_is_saturated      : string := "UNUSED";
    port_mult1_is_saturated      : string := "UNUSED";
    port_mult2_is_saturated      : string := "UNUSED";
    port_mult3_is_saturated      : string := "UNUSED";
    representation_a             : string := "UNSIGNED";
    representation_b             : string := "UNSIGNED";
    signed_aclr_a                : string := "ACLR3";
    signed_aclr_b                : string := "ACLR3";
    signed_pipeline_aclr_a       : string := "ACLR3";
    signed_pipeline_aclr_b       : string := "ACLR3";
    signed_pipeline_register_a   : string := "CLOCK0";
    signed_pipeline_register_b   : string := "CLOCK0";
    signed_register_a            : string := "CLOCK0";
    signed_register_b            : string := "CLOCK0";
    width_a                      : natural;
    width_b                      : natural;
    width_result                 : natural;
    lpm_type                     : string := "altmult_add"
);
port(
    aclr0               : in std_logic := '0';
    aclr1               : in std_logic := '0';
    aclr2               : in std_logic := '0';
    aclr3               : in std_logic := '0';
    addnsub1            : in std_logic := '0';
    addnsub1_round      : in std_logic := '0';
```

```
        addnsub3              : in std_logic := '0';
        addnsub3_round        : in std_logic := '0';
        clock0                : in std_logic := '1';
        clock1                : in std_logic := '1';
        clock2                : in std_logic := '1';
        clock3                : in std_logic := '1';
        dataa          : in std_logic_vector(number_of_multipliers
                              *width_a-1 downto 0) := (others => '1');
        datab          : in std_logic_vector(number_of_multipliers
                              *width_b-1 downto 0) := (others => '1');
        ena0                  : in std_logic := '1';
        ena1                  : in std_logic := '1';
        ena2                  : in std_logic := '1';
        ena3                  : in std_logic := '1';
        mult01_round          : in std_logic := '0';
        mult01_saturation     : in std_logic := '0';
        mult0_is_saturated    : out std_logic;
        mult1_is_saturated    : out std_logic;
        mult23_round          : in std_logic := '0';
        mult23_saturation     : in std_logic := '0';
        mult2_is_saturated    : out std_logic;
        mult3_is_saturated    : out std_logic;
        result         : out std_logic_vector(width_result-1 downto 0);
        scanina        : in std_logic_vector(width_a-1 downto 0)
                              := (others => '1');
        scaninb        : in std_logic_vector(width_b-1 downto 0)
                              := (others => '1');
        scanouta       : out std_logic_vector(width_a-1 downto 0);
        scanoutb       : out std_logic_vector(width_b-1 downto 0);
        signa          : in std_logic := '0';
        signb          : in std_logic := '0';
        sourcea        : in std_logic_vector(number_of_multipliers-1
                              downto 0) := (others => '0');
        sourceb        : in std_logic_vector(number_of_multipliers-1
                              downto 0) := (others => '0')
    );
end component;
```

2.3.3 Parameter definitions

NUMBER_OF_MULTIPLIERS: The number of multiplier which are to be added together. This value must be greater than or equal to 1.

WIDTH_A: The width of the dataa input busses.

WIDTH_B: The width of the datab input busses.

WIDTH_RESULT: Width of the result output bus.

INPUT_REGISTER_A0: Clock source for the A inputs of the first multiplier. Legal values are “UNREGISTERED” and “CLOCK0”.

INPUT_ACLR_A0: Asynchronous clear source for the A inputs of the first multiplier. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

INPUT_SOURCE_A0: Specifies the source of the A inputs to the first multiplier. Legal values are “DATAA”, “SCANA”, and “VARIABLE”. The value of “VARIABLE” is supported in Stratix II devices only.

INPUT_REGISTER_A1: Clock source for the A inputs to the second multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_A1: Asynchronous clear source for the A inputs to the second multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_SOURCE_A1: Specifies the source for the A inputs to the second multiplier. Legal values are "DATAA", "SCANA", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

INPUT_REGISTER_A2: Clock source for the A inputs to the third multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_A2: Asynchronous clear source the the A inputs to the third multiplier. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_A2: Specifies the source for the A inputs to the third multiplier. Legal values are "DATAA", "SCANA", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

INPUT_REGISTER_A3: Clock source for the fourth and all subsequent multipliers. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_A3: Asynchronous clear source for the fourth and all subsequest multipliers. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_A3: Specifies the source of the A inputs to the fourth multiplier. Legal values are "DATAA", "SCANA", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

REPRESENTATION_A: For specifying the number representation of the A multiplier inputs. Legal values are "UNSIGNED", "SIGNED", and "UNUSED". A value of "UNSIGNED" causes the A inputs to be interpreted as unsigned numbers. A value of "SIGNED" causes the A inputs to be interpreted as signed two's complement numbers. A value of "UNUSED" allows for dynamic control of the representation through the signa port.

SIGNED_REGISTER_A: The clock source for the first signa register. Legal values are "UNREGISTERED" and "CLOCK0".

SIGNED_ACLR_A: Asynchronous clear source for the first signa register. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

SIGNED_PIPELINE_REGISTER_A: The clock source for the second signa register. Legal values are "UNREGISTERED" and "CLOCK0".

SIGNED_PIPELINE_ACLR_A: Asynchronous clear source for the second signa register. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value. The width of the datab input buses.

INPUT_REGISTER_B0: Clock source for the B inputs of the first multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_B0: Asynchronous clear source for the B inputs of the first multiplier. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_B0: Specifies the source of the B inputs to the first multiplier. Legal values are "DATAB", "SCANB", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

INPUT_REGISTER_B1: Clock source for the B inputs to the second multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_B1: Asynchronous clear source for the B inputs to the second multiplier. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_B1: Specifies the source for the B inputs to the second multiplier. Legal values are "DATAB", "SCANB", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

INPUT_REGISTER_B2: Clock source for the B inputs to the third multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_B2: Asynchronous clear source for the B inputs to the third multiplier. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_B2: Specifies the source for the B inputs to the third multiplier. Legal values are "DATAB", "SCANB", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

INPUT_REGISTER_B3: Clock source for the fourth and all subsequent multipliers. Legal values are "UNREGISTERED" and "CLOCK0".

INPUT_ACLR_B3: Asynchronous clear source for the fourth and all subsequest multipliers. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

INPUT_SOURCE_B3: Specifies the source of the B inputs to the fourth multiplier. Legal values are "DATAB", "SCANB", and "VARIABLE". The value of "VARIABLE" is supported in Stratix II devices only.

REPRESENTATION_B: For specifying the number representation of the B multiplier inputs. Legal values are "USNIGNED", "SIGNED", and "UNUSED". A value of "UNSIGNED" causes the B inputs to be interpreted as unsigned numbers. A value of "SIGNED" causes the B inputs to be interpreted as signed two's complement numbers. A value of "UNUSED" allows for dynamic control of the representation through the signb port.

SIGNED_REGISTER_B: The clock source for the first signb register. Legal values are "UNREGISTERED" and "CLOCK0".

SIGNED_ACLR_B: Asynchronous clear source for the first signb register. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

SIGNED_PIPELINE_REGISTER_B: The clock source for the second signb register. Legal values are "UNREGISTERED" and "CLOCK0".

SIGNED_PIPELINE_ACLR_B: Asynchronous clear source for the second signb register. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

MULTIPLIER_REGISTER0: Clock source for the register immediately after the first multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

MULTIPLIER_ACLR0: Asynchronous clear source for the register immediately after the first multiplier. Legal values are "NONE", "ACLR0", "ACLR1", "ACLR2", and "ACLR3", but all registered ports must use the same value.

MULTIPLIER_REGISTER1: INPUT_REGISTER_A0: Clock source for the A inputs of the first multiplier. Legal values are "UNREGISTERED" and "CLOCK0".

MULTIPLIER_ACLR1: Asynchronous clear source for the register immediately after the second multiplier. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULTIPLIER_REGISTER2: Clock source for the register immediately after the third multiplier. Legal values are “UNREGISTERED” and “CLOCK0”.

MULTIPLIER_ACLR2: Asynchronous clear source for the register immediately after the third multiplier. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULTIPLIER_REGISTER3: Clock source for the register immediatley after the fourth and all subsequent multipliers. Legal values are “UNREGISTERED” and “CLOCK0”.

MULTIPLIER_ACLR3: Asynchronous clear source for the register immediately after the fourth and all subsequent multipliers. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULTIPLIER1_DIRECTION: Specifies whether the second multiplier will add or subtract its value from the sum. Legal values are “ADD” and “SUB”.

ADDNSUB_MULTIPLIER_REGISTER1: Clock source for the first register on the addnsub1 input. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_MULTIPLIER_ACLR1: Asynchronous clear source for the first register on the addnsub1 input. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ADDNSUB_MULTIPLIER_PIPELINE_REGISTER1: Clock source for the second register on the addnsub1 input. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_MULTIPLIER_PIPELINE_ACLR1: Asynchronous clear source for the second register on the addnsub1 input. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

MULTIPLIER3_DIRECTION: Specifies whether the fourth and all subsequent odd-numbered multipliers will add or subtract their results from the total. Legal values are “ADD” and “SUB”.

ADDNSUB_MULTIPLIER_REGISTER3: Clock source for the first register on the addnsub3 input. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_MULTIPLIER_ACLR3: Asynchronous clear source for the first register on the addnsub3 input. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

ADDNSUB_MULTIPLIER_PIPELINE_REGISTER3: Clock source for the second register on the addnsub3 input. Legal values are “UNREGISTERED” and “CLOCK0”.

ADDNSUB_MULTIPLIER_PIPELINE_ACLR3: Asynchronous clear source for the second register on the addnsub3 input. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

OUTPUT_REGISTER: Clock source for the output register. Legal values are “UNREGISTERED” and “CLOCK0”.

OUTPUT_ACLR: Asynchronous clear source for the output register. Legal values are “NONE”, “ACLR0”, “ACLR1”, “ACLR2”, and “ACLR3”, but all registered ports must use the same value.

EXTRA_LATENCY: Specifies latency to add to the circuit in addition to any registers which were already specified.

DEDICATED_MULTIPLIER_CIRCUITRY: Specifies whether or not to use the DSP block to implement the circuit. Legal values are “YES”, “NO”, and “AUTO”. A value of “YES” will cause an implementation using the DSP blocks.

MULTIPLIER01_ROUNDING: Enable rounding at the output of the multiplier stage, for multipliers 0 and 1. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is “NO” for Cyclone II devices.

MULTIPLIER01_SATURATION: Enable saturation handling at the output of the multiplier stage, for multipliers 0 and 1. Saturation at the output of the multiplier can only occur if dataa and datab are both the maximum negative number. In this case, the product is a positive that is too large to be represented in the given number of bits. Turning on saturation handling will give a maximum positive result for this case. Legal value is “NO” for Cyclone II devices.

MULTIPLIER23_ROUNDING: Enable rounding at the output of the multiplier stage, for multipliers 2 and 3. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is “NO” for Cyclone II devices.

MULTIPLIER23_SATURATION: Enable saturation handling at the output of the multiplier stage, for multipliers 2 and 3. Saturation at the output of the multiplier can only occur if dataa and datab are both the maximum negative number. In this case, the product is a positive that is too large to be represented in the given number of bits. Turning on saturation handling will give a maximum positive result for this case. Legal value is “NO” for Cyclone II devices.

ADDER1_ROUNDING: Enable rounding in the adder stage, for the adder used with multiplier 1. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is “NO” for Cyclone II devices.

ADDER1 _SATURATION: Enable saturation handling in the adder stage, for the adder used with multiplier 1. Saturation should only be used with input that conforms to signed 1.15 fractional number representation. Turning on saturation handling will give a maximum positive or negative result in overflow situations. Legal value is “NO” for Cyclone II devices.

ADDER3_ROUNDING: Enable rounding in the adder stage, for the adder used with multiplier 3. Rounding should only be used with input that conforms to signed 1.15 fractional number representation. Legal value is “NO” for Cyclone II devices.

ADDER3 _SATURATION: Enable saturation handling in the adder stage, for the adder used with multiplier 3. Saturation should only be used with input that conforms to signed 1.15 fractional number representation. Turning on saturation handling will give a maximum positive or negative result in overflow situations. Legal value is “NO” for Cyclone II devices.

PORT_MULT0_IS_SATURATED: Indicates that the *mult0_is_saturated* output should be used or unused. Legal value is “UNUSED” for Cyclone II devices. If the value is “UNUSED”, the *mult0_is_saturated* output will be stuck at GND.

PORT_MULT1_IS_SATURATED: Indicates that the *mult1_is_saturated* output should be used or unused. Legal value is “UNUSED” for Cyclone II devices. If the value is “UNUSED”, the *mult1_is_saturated* output will be stuck at GND.

PORT_MULT2_IS_SATURATED: Indicates that the *mult2_is_saturated* output should be used or unused. Legal Legal value is “UNUSED” for Cyclone II devices. If the value is “UNUSED”, the *mult2_is_saturated* output will be stuck at GND.

PORT_MULT3_IS_SATURATED: Indicates that the *mult3_is_saturated* output should be used or unused. Legal value is “UNUSED” for Cyclone II devices. If the value is “UNUSED”, the *mult3_is_saturated* output will be stuck at GND.

INTENDED_DEVICE_FAMILY: Defaults to “Stratix”. Normally, this parameter would only be used for behavioral models, since megafunctions look at the Quartus II-defined device family. In this case, it is used to select Stratix-compatible assumptions about port and parameter priority resolution. In the original Stratix version of altmult_add, dynamic control inputs such as signa, when connected, took precedence over the value of conflicting parameters, in this case REPRESENTATION_A. Going forward, we prefer that all behavioral information be taken from parameter and port values, and not from connectivity information, since this requires non-standard Verilog and VHDL instantiations. The MegaWizard plug-in for altmult_add will always

write a value for this parameter starting in the Quartus II software version 4.0. The megafunction will check to see if a value is assigned, and if none is assigned, it will revert to pre-Quartus II 4.0 port/parameter priority rules.

2.3.4 Port definitions

dataa: These are the A inputs to the multipliers.

datab: These are the B inputs to the multipliers.

clock3, clock2, clock1, clock0: These are the four clock inputs.

aclr3, aclr2, aclr1, aclr0: These are the four asynchronous clear inputs.

ena3, ena2, ena1, ena0: These are the four clock enables. Ena3 corresponds to clock3; ena2 corresponds to clock2, etc.

signa, signb: These are for dynamically controlling the representation of the a and b inputs. A high value on signa/b causes the A/B inputs to be interpreted as signed two's complement numbers. A low value on signa/b causes the A/B inputs to be interpreted as unsigned numbers.

addnsub1, addnsub3: These are for dynamically controlling whether to do an add or subtract of the corresponding multiplier. A high value on addnsub1/3 causes an addition to be performed of the second/(fourth and subsequent odd) multipliers. A low value causes a subtraction.

scanina: The optional input to the A scan chain when INPUT_SOURCE_A is "SCANA" or "VARIABLE".

scaninb: The optional input to the B scan chain when INPUT_SOURCE_B is "SCANB" or "VARIABLE".

mult01_round: Enables multiplier stage rounding for multiplier 0 and 1 when MULTIPLIER01_ROUNDING = "VARIABLE".

mult23_round: Enables multiplier stage rounding for multiplier 2 and 3 when MULTIPLIER23_ROUNDING = "VARIABLE".

mult01_saturate: Enables multiplier stage saturation handling for multiplier 0 and 1 when MULTIPLIER01_SATURATION = "VARIABLE".

mult23_saturate: Enables multiplier stage saturation handling for multiplier 2 and 3 when MULTIPLIER23_SATURATION = "VARIABLE".

adder1_round: Enables adder stage rounding for the adder used with multiplier 1 when ADDER1_ROUNDING = "VARIABLE".

adder3_round: Enables adder stage rounding for the adder used with multiplier 3 when ADDER3_ROUNDING = "VARIABLE".

result: The result of the multiply and addition.

scanouta, scanoutb: These are the outputs of the A and B scan chains.

mult0_is_saturated: Indicates that saturation handling has occured for multiplier 0. Must be explicitly enabled with the parameter PORT_MULT0_IS_SATURATED = "USED".

mult1_is_saturated: Indicates that saturation handling has occured for multiplier 1. Must be explicitly enabled with the parameter PORT_MULT1_IS_SATURATED = "USED".

mult2_is_saturated: Indicates that saturation handling has occured for multiplier 2. Must be explicitly enabled with the parameter PORT_MULT2_IS_SATURATED = "USED".

mult3_is_saturated: Indicates that saturation handling has occured for multiplier 3. Must be explicitly enabled with the parameter PORT_MULT3_IS_SATURATED = “USED”.

2.4 altsyncram

The RAM blocks in the Cyclone II architecture are compatible with the M4K RAM blocks in the Stratix II architecture. They are backwards-compatible with M4K RAM blocks in the original Cyclone device family, with the trivial exception that input registers will no longer support asynchronous clear. This is a minor issue because the asynchronous clear of input registers in the Cyclone architecture did not have a visible effect on the RAM outputs.

Compared to the original Cyclone device family, there is an enhancement in clock enable choices for RAM input and output registers on both the A and B ports. Four new parameters, *clock_enable_input_a, clock_enable_output_a, clock_enable_input_b, clock_enable_output_b,* give the user the choice of disabling the effect of the corresponding clock enable input for these groups of registers. A related change is the addition of address stall inputs for both A and B address ports. Address stall acts like an extra clock enable input for the address registers only. It only affects the loading of address registers, and not the related read or write operation. Address stall is added to improve the efficiency in cache-miss applications.

2.4.1 MegaWizard plug-in

The MegaWizard plug-in will enforce Cyclone II feature subset restrictions.

2.4.2 Parameter and port list

```
component altsyncram
     generic (
        address_aclr_a                  : string := "UNUSED";
        address_aclr_b                  : string := "NONE";
        address_reg_b                   : string := "CLOCK1";
        byte_size                       : natural := 8;
        byteena_aclr_a                  : string := "UNUSED";
        byteena_aclr_b                  : string := "NONE";
        byteena_reg_b                   : string := "CLOCK1";
        clock_enable_input_a            : string := "NORMAL";
        clock_enable_input_b            : string := "NORMAL";
        clock_enable_output_a           : string := "NORMAL";
        clock_enable_output_b           : string := "NORMAL";
        indata_aclr_a                   : string := "UNUSED";
        indata_aclr_b                   : string := "NONE";
        indata_reg_b                    : string := "CLOCK1";
        init_file                       : string := "UNUSED";
        init_file_layout                : string := "PORT_A";
        maximum_depth                   : natural := 0;
        numwords_a                      : natural := 0;
        numwords_b                      : natural := 0;
        operation_mode                  : string := "BIDIR_DUAL_PORT";
        outdata_aclr_a                  : string := "NONE";
        outdata_aclr_b                  : string := "NONE";
        outdata_reg_a                   : string := "UNREGISTERED";
        outdata_reg_b                   : string := "UNREGISTERED";
        ram_block_type                  : string := "AUTO";
        rdcontrol_aclr_b                : string := "NONE";
```

```
        rdcontrol_reg_b                 : string := "CLOCK1";
        read_during_write_mode_mixed_ports    : string := "DONT_CARE";
        width_a                         : natural;
        width_b                         : natural := 1;
        width_byteena_a                 : natural := 1;
        width_byteena_b                 : natural := 1;
        widthad_a                       : natural;
        widthad_b                       : natural := 1;
        wrcontrol_aclr_a                : string := "UNUSED";
        wrcontrol_aclr_b                : string := "NONE";
        wrcontrol_wraddress_reg_b       : string := "CLOCK1";
        lpm_type                        : string := "altsyncram"
    );
    port(
        aclr0           : in std_logic := '0';
        aclr1           : in std_logic := '0';
        address_a       : in std_logic_vector(widthad_a-1 downto 0);
        address_b       : in std_logic_vector(widthad_b-1 downto 0)
                                := (others => '1');
        addressstall_a  : in std_logic := '0';
        addressstall_b  : in std_logic := '0';
        byteena_a       : in std_logic_vector(width_byteena_a-1 downto 0)
                                := (others => '0');
        byteena_b       : in std_logic_vector(width_byteena_b-1 downto 0)
                                := (others => '0');
        clock0          : in std_logic := '1';
        clock1          : in std_logic := '1';
        clocken0        : in std_logic := '1';
        clocken1        : in std_logic := '1';
        data_a          : in std_logic_vector(width_a-1 downto 0)
                                := (others => '1');
        data_b          : in std_logic_vector(width_b-1 downto 0)
                                := (others => '1');
        q_a             : out std_logic_vector(width_a-1 downto 0);
        q_b             : out std_logic_vector(width_b-1 downto 0);
        rden_b          : in std_logic := '1';
        wren_a          : in std_logic := '0';
        wren_b          : in std_logic := '0'
    );
end component;
```

2.4.3 Parameter Definitions

OPERATION_MODE: Specifies the operation mode of the ram, It is one of the four choices “ROM”, “SINGLE_PORT”, “DUAL_PORT” or “BIDIR_DUAL_PORT”

WIDTH _A: Width of the input data and output data bus for port A

WIDTHAD_ A: Width of input address bus for port A

NUMWORDS_ A: Number of words in the view of port A

OUTDATA_REG_A: Registering option of output data register for port A, can be one of the four choices “CLOCK0”, “CLOCK1”, “UNREGISTERED” or “UNUSED”. Default is “UNREGISTERED”

ADDRESS_ACLR_A: Clear for address of port A. Can be one of three options: “CLEAR0”, “NONE” or “UNUSED”. Default is “NONE”

OUTDATA_ACLR_A: Clear for output register of port A. Can be one of four options: “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

CLOCK_ENABLE_INPUT_A: Clock enable bypass control for input registers of port A. All input registers are affected, including address, data, wren, rden, and byteena. Choices are “NORMAL” or “BYPASS”. Default is “NORMAL” to maintain Stratix behavior.

CLOCK_ENABLE_OUTPUT_A: Clock enable bypass control for output register of port A. Choices are “NORMAL” or “BYPASS”. Default is “NORMAL” to maintain Stratix behavior.

WRCONTROL_ACLR_A: Clear for input write enable register of port A. One of three options: “CLEAR0”, “NONE” or “UNUSED”. Default is “NONE”

INDATA_ACLR_A: Clear for input data register of port A. One of three options: “CLEAR0”, “NONE” or “UNUSED”. Default is “NONE”

BYTEENA_ACLR_A: Clear for input byte enable register for port A. One of three options: “CLEAR0”, “NONE” or “UNUSED”. Default is “NONE”

WIDTH_ B: Width of the input data and output data bus for port B

WIDTHAD_ B: Width of input address bus for port B

NUMWORDS_ B: Number of words in the view of port B

OUTDATA_REG_B: Registering option of output data register for port B, can be one of the four choices “CLOCK0”, “CLOCK1”, “UNREGISTERED” or “UNUSED”. Default is “UNREGISTERED”

ADDRESS_ACLR_B: Clear for address of port B. Can be one of four options : “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

OUTDATA_ACLR_B: Clear for output register of port B. Can be one of four options: “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

WRCONTROL_ACLR_B: Clear for input write enable register of port B. One of four options: “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

INDATA_ACLR_B: Clear for input data register of port B. One of four options: “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

BYTEENA_ACLR_B: Clear for input byte enable register for port A. One of four options: “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

INDATA_REG_B: Clock source for input data register. One of three options “CLOCK0”, “CLOCK1” or “UNUSED”. Has to be used when data_b is used. Default is “CLOCK1”

WRCONTROL_WRADDRESS_REG_B: Clock source for write control and address register during write mode for port B. One of three options “CLOCK0”, “CLOCK1” or “UNUSED”. Has to be used when wren_b is used and in write mode. Default is “CLOCK1”.

RDCONTROL_REG_B: Clock source for rdenable register during read mode for port B. One of three options “CLOCK0”, “CLOCK1” or “UNUSED”. Has to be used when rden_b is used. Default is “CLOCK1”

ADDRESS_REG_B: Clock source for address register for port B. One of three options “CLOCK0”, “CLOCK1” or “UNUSED”. Has to be used when port B is used. Default is “CLOCK1”

OUTDATA_REG_B: Clock source for output register for port B. One of four options “CLOCK0”, “CLOCK1”, “UNREGISTERED” or “UNUSED”. Default is “UNREGISTERED”

BYTEENA_REG_B: Clock source for the byte enable register for port B. One of three options “CLOCK0”, “CLOCK1” or “UNUSED”. Has to be used when port b Byte enable is used. Default is “CLOCK1”

RDCONTROL_ACLR_B: Clear for rdenable control register of port B. One of four options “CLEAR0”, “CLEAR1”, “NONE” or “UNUSED”. Default is “NONE”

CLOCK_ENABLE_INPUT_B: Clock enable bypass control for input registers of port B. All input registers are affected, including address, data, wren, rden, and byteena. Choices are “NORMAL” or “BYPASS”. Default is “NORMAL” to maintain Stratix behavior.

CLOCK_ENABLE_OUTPUT_B: Clock enable bypass control for output register of port B. Choices are “NORMAL” or “BYPASS”. Default is “NORMAL” to maintain Stratix behavior.

WIDTH_BYTEENA_A: Width of byte enable port for port A. Has to be equal to width of port A / Byte size. Default is 1.

WIDTH_BYTEENA_B: Width of byte enable port for port B. Has to be equal to width of port B / Byte size. Default is 1.

BYTE_SIZE: Indicates whether the byte size used by user is 1, 2, 4, 8, or 9. Default is 8.

READ_DURING_WRITE_MODE_MIXED_PORTS: Indicates the type of behavior when read and write happen at different ports on the same RAM address. One of two options “OLD_DATA” or “DONT_CARE” (default)

RAM_BLOCK_TYPE: Indicates the type of RAM_BLOCK preferred by the user. Can be one of the two options, “M4K” or “AUTO” (default).

INIT_FILE: Specifies the name of the Initialization file used.

INIT_FILE_LAYOUT: Specifies whether the init file has to be used with respect to port a depth X width or with respect to port b depth X width. The default is port A. For simple dual, the megafunction assumes a default of port B and for other modes, it assumes a default of port A unless otherwise specified

MAXIMUM_DEPTH: Specifies the maximum depth to which the RAM should be segmented. Default is Zero, when the megafunction chooses the depth based on ram_block_type or on the other requirements (in case of auto).

2.4.4 Port definitions

wren_a: Write Enable port for port A.

wren_b: Write Enable port for port B. Only available in bidir-dual-port mode.

rden_b: Read Enable port for port B. Only available in simple-dual-port mode.

data_a: Data input for port A

data_b: Data input for port B.

address_a: Address port for port A.

address_b: Address port for port B.

addressstall_a: Address stall input for port A. A '1' input stalls the current address. For Stratix II and Cyclone II devices only.

addressstall_b: Address stall input for port B. A '1' input stalls the current address. For Stratix II and Cyclone II devices only.

clock0, clock1: Clock input ports for the RAM. Clock0 should always be connected.

clocken0, clocken1: Clock enable ports for the clocks.

aclr0, aclr1: Clear ports for the RAM input and output registers.

byteena_a: Byte enable ports for port A. Should be used only when width of the port A input data bus is at least two bytes.

byteena_b: Byte enable ports for port B. Should be used only when width of the port B input data bus is at least two bytes.

q_a: Output data bus for port A

q_b: Output data bus for port B

2.5 parallel_add

Parallel adder megafunction, adds a variable number of inputs to produce a single sum result. For 4-LUT-based devices, it uses a 2-to-1 adder tree implementation, and for p-term device families, it uses 3-to-2 carry-save adder trees and 2-to-1 adders as appropriate.

2.5.1 MegaWizard plug-in

No changes needed.

2.5.2 Megafunction Ports and Parameters

```
component parallel_add
    generic (
        msw_subtract          : string := "NO";
        pipeline              : natural := 0;
        representation        : string := "UNSIGNED";
        result_alignment      : string := "LSB";
        shift                 : natural := 0;
        size                  : natural;
```

```
        width                : natural;
        widthr               : natural;
        lpm_type             : string := "parallel_add"
    );
    port(
        aclr          : in std_logic := '0';
        clken         : in std_logic := '1';
        clock         : in std_logic := '0';
        data          : in std_logic_vector(size*width-1 downto 0)
                              := (others => '0');
        result        : out std_logic_vector(widthr-1 downto 0)
    );
end component;
```

2.5.3 Parameter definitions

width: Width of input data (in bits).

size: Number of input numbers

widthr: Desired width of result

shift: relative shift of data vectors

representation: SIGNED/UNSIGNED addition

pipeline: Latency of pipelined adder

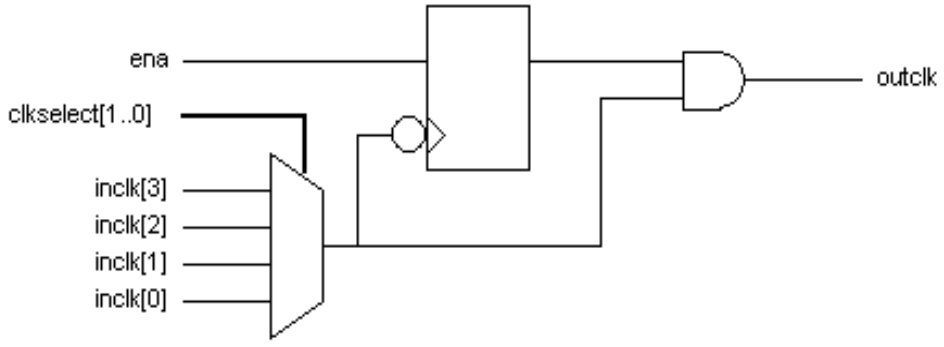
msw_subtract: "NO", or “YES”. Most Significant Word is Added or Subtracted

result_alignment: “MSB” or “LSB”. If widthr is less than optimal result width picks up the result bits from MSB or LSB

3. Non-Inferable Megafunctions

3.1 altclkbuf

3.1.1 Behavior Diagram



3.1.2 MegaWizard plug-in

Changes needed?

3.1.3 Megafunction Ports and Parameters

```
component altclkbuf
        generic (
                clock_type      :       string := "AUTO"
        );
        port(
                clkselect       :       in std_logic_vector(1 downto 0) := (others => '0');
                ena             :       in std_logic := '1';
                inclk           :       in std_logic_vector(3 downto 0) := (others => '0');
                outclk          :       out std_logic
        );
end component;
```

3.1.4 Parameter and Port Definitions

CLOCK_TYPE: Buffer clock type. Legal values are “AUTO,” “GCLK,” “LCLK,” “EXTCLK,” and “SIDE_CLK.”

inclk: clock inputs that drive the clock network

clkselect: input allows dynamically selecting which clock source drives the clock network driven by this clock buffer. The signal selects one of the four clock inputs where a value of ‘00’ selects inclk[0], ‘01’ selects inclk[1], ‘10’ selects inclk[2], and ‘11’ selects inclk[3]. Defaults to GND if left unconnected. If this signal is connected, then only the Global Clock network can be driven by this clock buffer.

ena: input that allows enabling or disabling the entire clock network driven by this clock buffer. If left unconnected, this signal defaults to VCC.

outclk: clock output that will drive the clock network associated with this clock buffer.

3.2 altddio_in, altddio_out, altddio_bidir

Double data rate I/O megafunctions.

3.3 altmemmult

This is the RAM-based coefficient (i.e. constant) multiplier megafunction. This megafunction implements a multi-cycle multiplier by using a ROM or RAM to do partial product look-up, and then a shifting accumulator (altaccumulate) to accumulate the final product. The behavior is very different from lpm_mult, and we believe that it is mainly useful for lower bandwidth DSP applications that need large numbers of moderate bandwidth multipliers.

This megafunction is unchanged for Cyclone II devices

See the document *cbx_altmemmult_ffd.doc* for more details.

3.4 altpll

This is the Cyclone II and Stratix PLL megafunction. The altclklock megafunction can still be used for very simple PLL needs, since it's generic enough that it supports all device families with PLLs.

3.5 dcfifo

The dual-clock FIFO megafunction takes advantage of the selectable clock enable on Cyclone II RAM blocks. The clock enable bypass feature is used on the read address register to allow the RAM to read internally on every clock cycle. In legacy mode (non-showahead), the clock enable can then be used to preserve valid data in the Q output register. This results in a dramatic LC savings compared to Cyclone dual-clock FIFOs, because the LE-based, 3-deep output FIFO can be omitted. It also results in much lower internal latency, and a simpler, cleaner design.

The reduced latency affects the behavior in legacy mode, and addresses a frequent customer complaint with Cyclone FIFOs. Show-ahead mode behavior is identical to Cyclone "area" show-shead mode. In Cyclone II devices, there is no such thing as "speed" mode, which exists only in Stratix and Cyclone devices because of the awkward RAM architecture of the Stratix family.

Another difference compared to Cyclone dcfifo is in the read input path. The Cyclone II implementation takes advantage of the new address stall RAM feature. This simplifies the read input path, and helps boost performance, but it has no impact on behavior.

The dcfifo write path in Cyclone II devices is identical to Cyclone devices. See the schematic below for additional details.

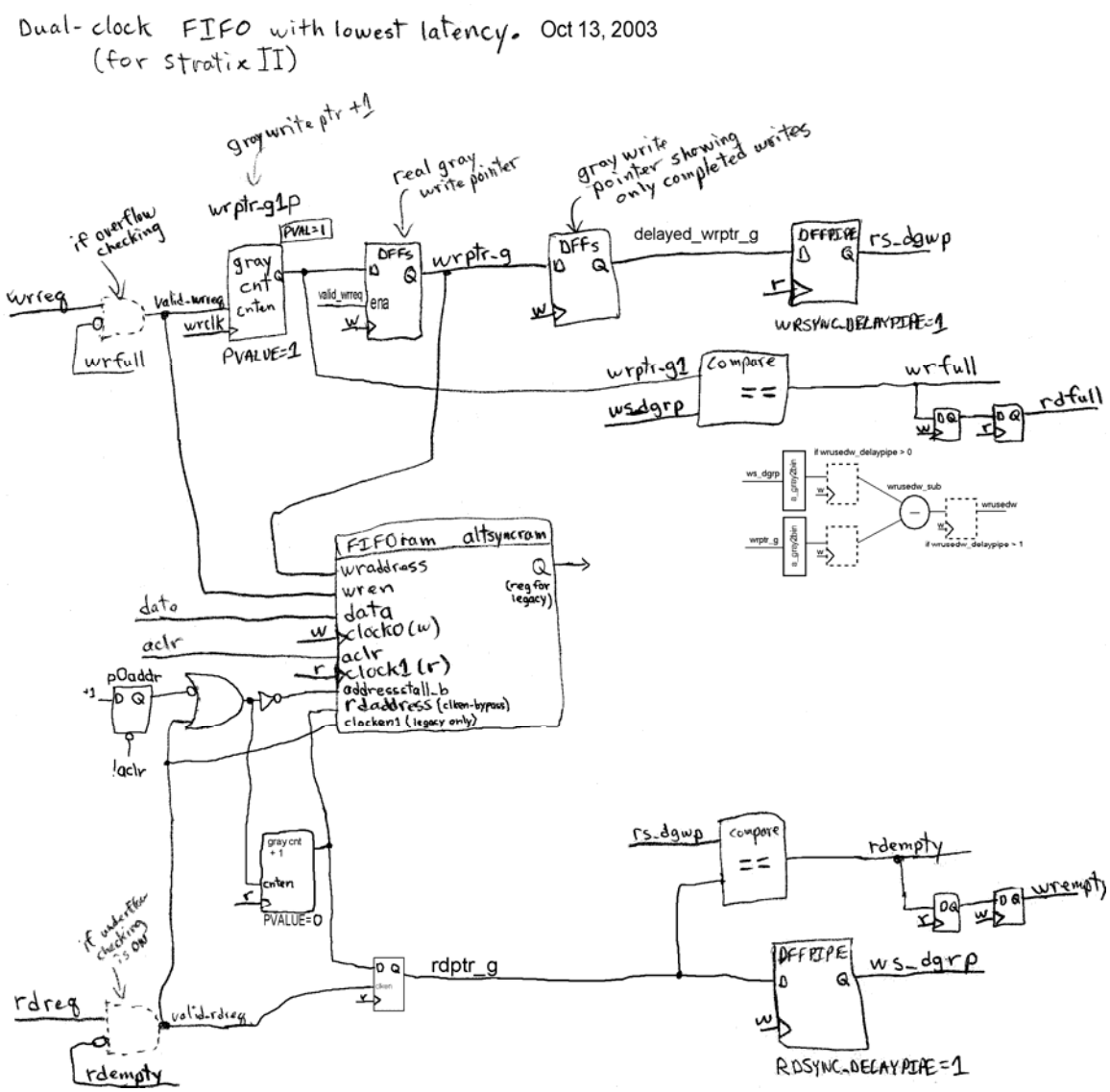


Figure 3-1 dcfifo schematic for Stratix II & Cyclone II Architecture

3.6 scfifo

This is the single-clock FIFO megafunction. It is unchanged vs. Cyclone devices.