XAPP466 (v1.1) May 20, 2005

Using Dedicated Multiplexers in Spartan-3 Generation FPGAs

Summary

The Spartan™-3 Generation architecture includes dedicated multiplexers within the Configurable Logic Blocks (CLBs). These specialized multiplexers improve the performance and density of not just wide multiplexers but almost any wide-input function. Using these resources, a 32:1 multiplexer fits in just one level of logic, as do some Boolean logic functions of up to 79 inputs.

This application note applies to all Spartan-3 Generation FPGA families, which include the Spartan-3 family, the Spartan-3L family, and the Spartan-3E family.

Introduction

A multiplexer, or mux, is a common building block of almost every logic design, selecting one of several possible input signals. Spartan-3 Generation FPGAs are very efficient at implementing multiplexers: small ones in the look-up tables and larger ones using dedicated multiplexer resources. Any Spartan-3 Generation device easily implements:

- a 4:1 mux in one slice
- a 16:1 mux in one CLB
- a 32:1 mux in two CLBs

The same logic resources also can be used for wide, general-purpose logic functions. For applications like comparators, encoder-decoders, or case statements, these resources provide an optimal solution. These resources are used automatically by the Xilinx development system, especially when a CASE statement is used, and then optimized for the timing requirements of a given design. This application note explains how to further optimize the use of dedicated multiplexers and how to analyze their use in a design.

This document describes the dedicated multiplexer resources in the Spartan-3 Generation architecture. The signals and parameters associated with the multiplexers are defined. The many methods to include multiplexers in a design are described along with recommendations and guidelines for their use.

Advantages of Dedicated Multiplexers

Spartan-3 Generation FPGAs are based on four-input look-up tables (LUTs) that can provide any possible function of the four inputs. The largest mux that a single LUT supports is a 2:1 mux, with the fourth input available as a possible enable. One method to construct larger muxes would be to cascade multiple LUTs. For example, a 4:1 mux could be built by combining the outputs of two LUTs into a third LUT. However, this method adds two full levels of logic delays plus an additional routing delay between the LUTs. Without special resources, an 8:1 mux would consume seven LUTs as well as add three levels of logic delays plus two levels of routing delays, as shown in Figure 1.

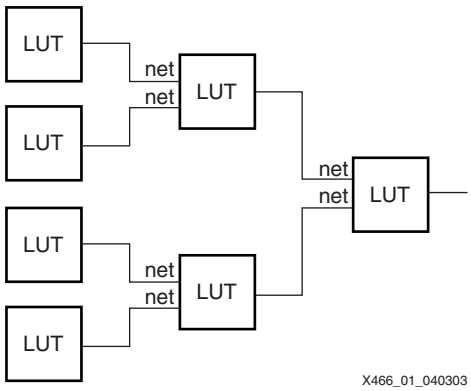

Figure 1: **8:1 Mux, 7 LUTs, 3 Levels of Logic**

To increase multiplexer speed and density, Spartan-3 Generation FPGAs provide a dedicated 2:1 mux following every LUT, which replaces additional levels of LUT-based logic. One of these, called the F5MUX, combines adjacent LUTs to create a 4:1 mux. The other mux, following every pair of LUTs, combines muxes into even wider functions, with different capabilities depending on its location in the CLB. This mux is called the FiMUX, where the index "i" equals 6, 7, or 8. For example, the F6MUX combines the results of two F5MUX elements to create an 8:1 mux as shown in Figure 2. The connections from the LUTs to the muxes and between the muxes are dedicated and have zero connection delay. The combination of LUTs and dedicated multiplexers allows very efficient implementation of even large multiplexers.

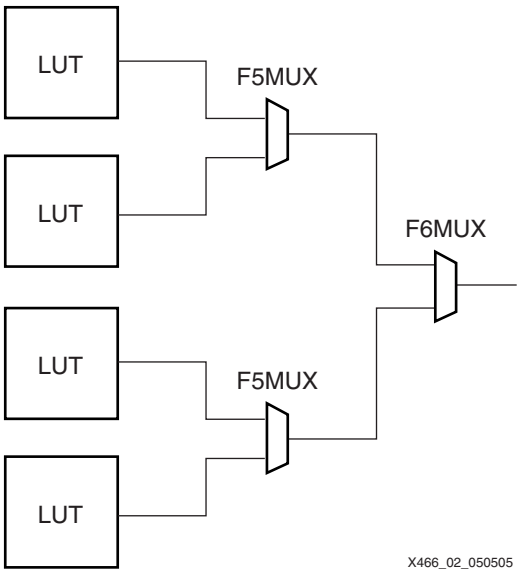

Figure 2: **8:1 Mux, 4 LUTs, 1 Level of Logic**

CLB Multiplexer Resources

The Spartan-3 Generation architecture consists of an array of identical Configurable Logic Blocks, or CLBs. Each CLB is made up of four slices: two SLICEMs with memory capability and two SLICELs with logic-only capability. Each slice is identical with respect to logic and mux resources. Each slice has two LUTs, an F5MUX, and a second expansion mux (see Figure 3).

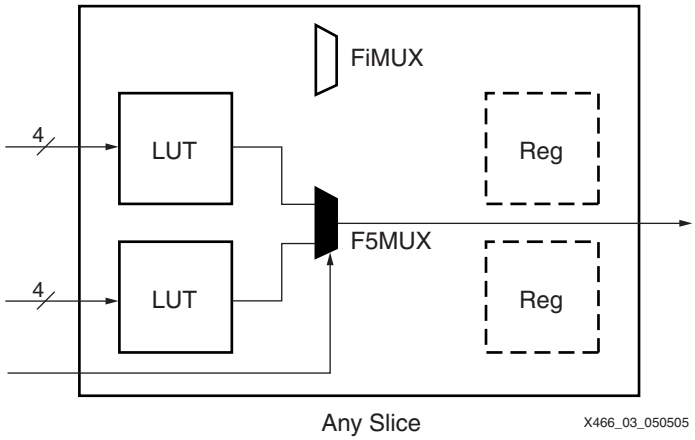

Figure 3: **LUTs and F5MUX in a Slice**

F5MUX

The F5MUX always combines the two LUTs in a slice. If those two LUTs contain 2:1 muxes with the same control input, then the overall result is a 4:1 mux (see Figure 4).

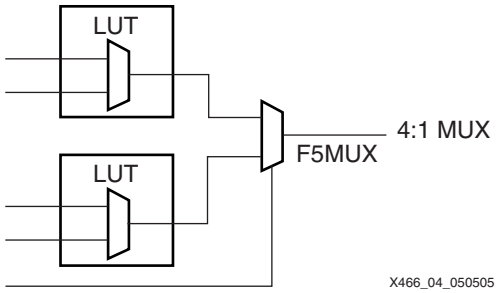

Figure 4: **4:1 Mux Implemented Using F5MUX**

The F5MUX is so named because it generates any possible Boolean logic function of five inputs (see Figure 5). If the two LUTs contain independent functions of the same four inputs, the mux select line becomes the fifth input. The F5MUX becomes a function expander that is just as efficient as another 3-input LUT for implementing any 5-input function. This is a significant advantage over other FPGA architectures.

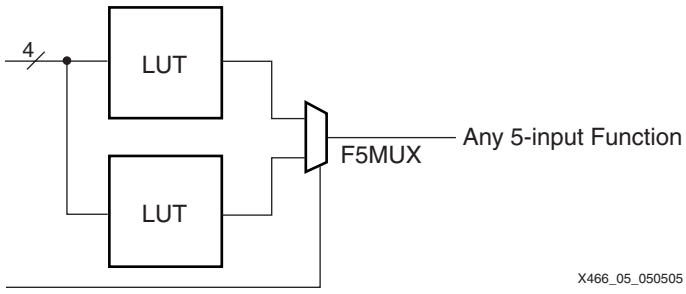

Figure 5: **Any 5-input Function Can Be Implemented Using F5MUX**

As shown in Figure 6, the F5MUX also produces some functions of up to nine inputs, if they can be partitioned into two 4-input LUTs and a mux.

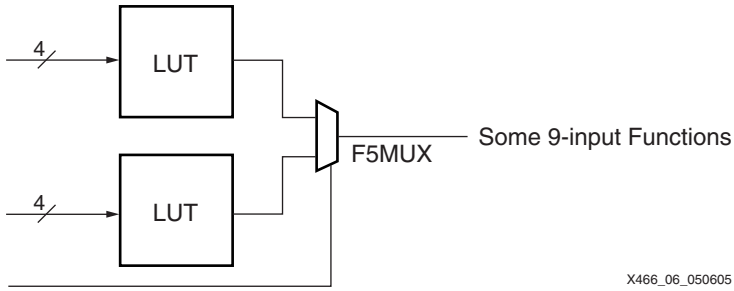

Figure 6: **Some 9-Input Functions Can Be Implemented Using a F5MUX**

Consequently, the F5MUX generates any 5-input function, the 4:1 mux 6-input function, or some 9-input functions.

FiMUX

The second mux, called the FiMUX, functions as either an F6MUX, F7MUX, or F8MUX, depending on its location and connections to the other muxes.

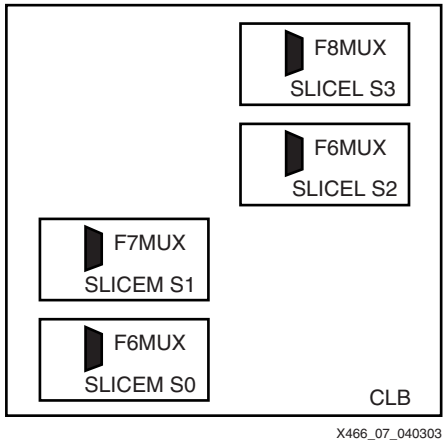

Figure 7: **FiMUX Positions in a CLB**

Each FiMUX receives inputs from muxes of the next lower number; for example, the two F6MUX results drive the F7MUX. Like the F5MUX, the FiMUX has the flexibility to implement other types of functions besides just multiplexers. The F6MUX is so named because it creates any function of six inputs. Similarly, the F7MUX generates any function of seven inputs, and the F8MUX generates any function of eight inputs.

Table 1: **Mux Capabilities**

Mux	Usage	Input Source	Total Number of Inputs per Function		
			For Any Function	For Mux	For Limited Functions
F5MUX	F5MUX	LUTs	5	6 (4:1 mux)	9
FiMUX	F6MUX	F5MUX	6	11 (8:1 mux)	19
	F7MUX	F6MUX	7	20 (16:1 mux)	39
	F8MUX	F7MUX	8	37 (32:1 mux)	79

Naming Conventions

In this document and in the Spartan-3 data sheets, the mux that serves as either F6MUX, F7MUX, or F8MUX generically is called an FiMUX (i = 6, 7, or 8). This name avoids confusion with the static CLB mux that generates the X output, which the FPGA Editor refers to as the "FXMUX". The FiMUX is always referred to as the "F6MUX" in the FPGA Editor. The timing analyzer also refers to the path through the FiMUX to the CLB pin as "TIF6Y", although it may be used as an F7MUX or F8MUX.

The library components are called MUXF5, MUXF6, MUXF7, and MUXF8. MUXF6, MUXF7, and MUXF8 use the FiMUX and restrict the placement to a specific relative location in the CLB.

Dedicated Local Routing

A significant benefit of the dedicated multiplexers is the dedicated routing that connects between levels. Although each mux is implemented as one pass through the CLB, the outputs connect back to the CLB inputs through local interconnect with zero routing delay. The result is the same as if the muxes were in series within the CLB.

The F5MUX feeds the F5 CLB output pin, which only connects back to an FiMUX input on the same CLB (called FXINA and FXINB). The FiMUX feeds the FX CLB output pin, which also feeds back to an FiMUX input on the same CLB, or in the case of the F7MUX, also to the CLB below. If the mux result is needed elsewhere, it connects to a general-purpose CLB output (X for the F5MUX, Y for the FiMUX).

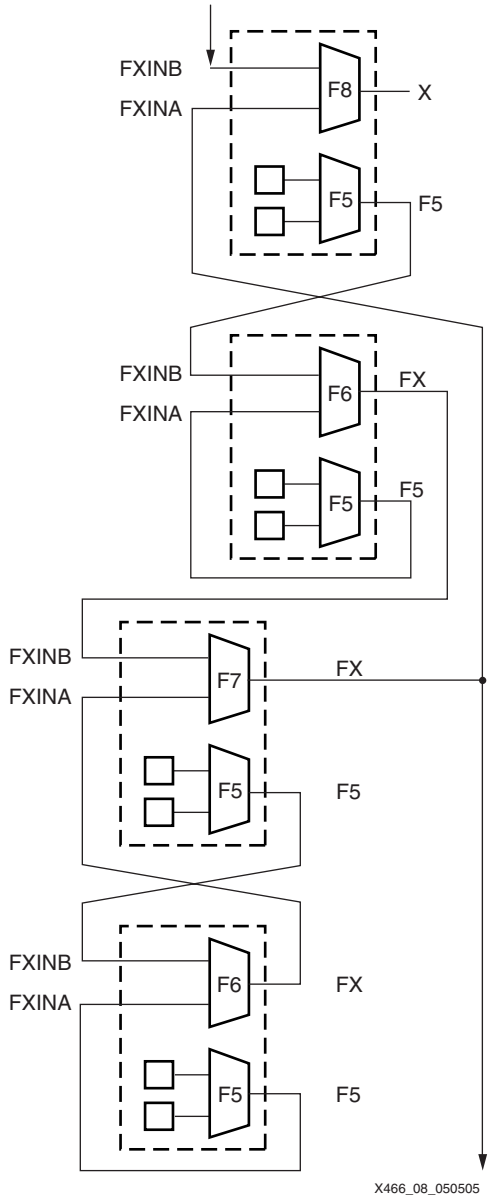

Figure 8: **Muxes and Dedicated Feedback in a Spartan-3 Generation CLB**

Mux Select Inputs

The select inputs for the multiplexers come from general-purpose routing. The select input for the F5MUX is the BX input on the CLB, and the select input for the FiMUX is the BY input on the CLB.

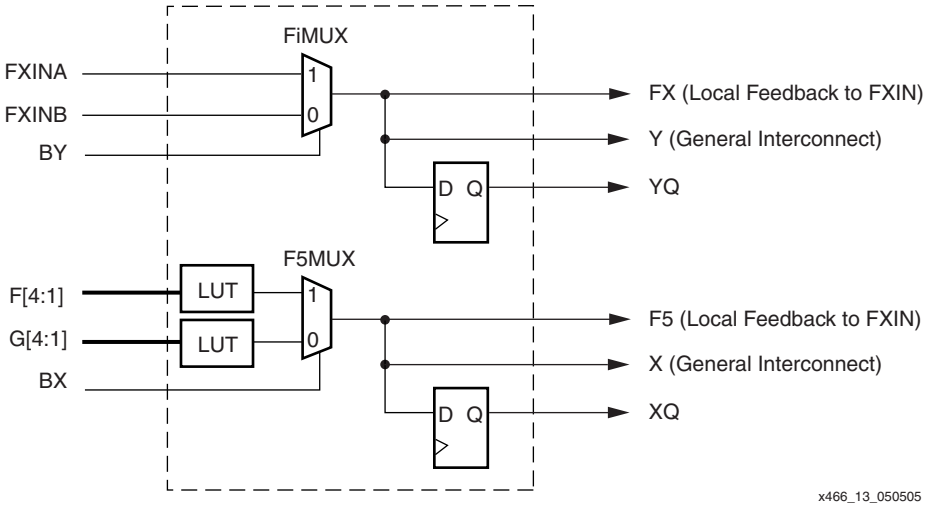

Figure 9: **Dedicated Multiplexers in a Spartan-3 Generation CLB**

Implementation Examples

Wide-Input Multiplexers

Each LUT optionally implements a 2:1 multiplexer. In each slice, the F5MUX and two LUTs can implement a 4:1 multiplexer. As shown in Figure 10, the F6MUX and two slices implement an 8:1 multiplexer. The F7MUX and the four slices of any CLB implement a 16:1 multiplexer, and the F8MUX and two CLBs implement a 32:1 multiplexer.

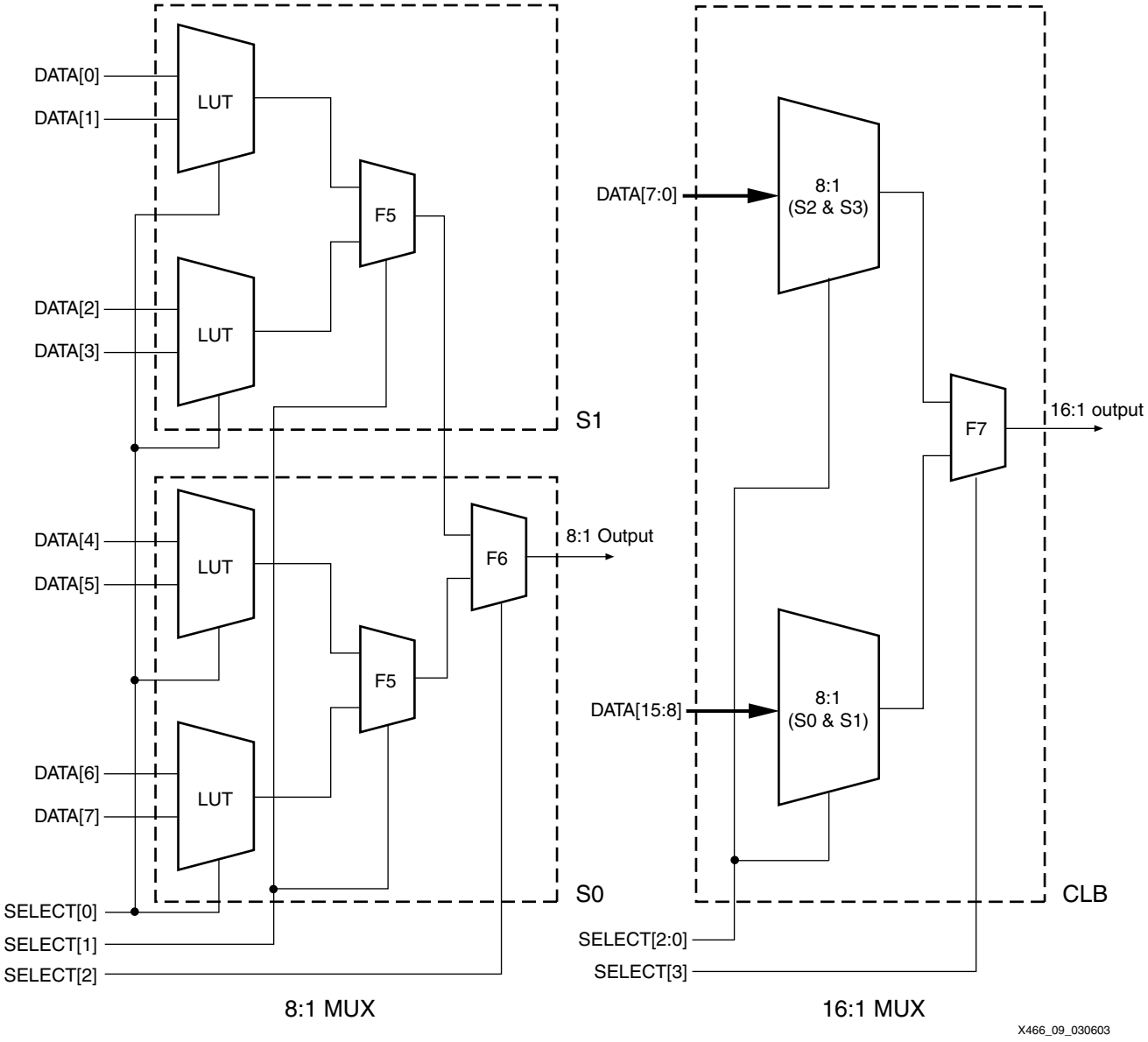

Figure 10: **8:1 and 16:1 Multiplexers**

Wide-Input Functions

Slices S0 and S2 have an F6MUX, designed to combine the outputs of two F5MUX resources. Figure 11 illustrates a combinatorial function up to 19 inputs in the slices S0 and S1, or in the slices S2 and S3.

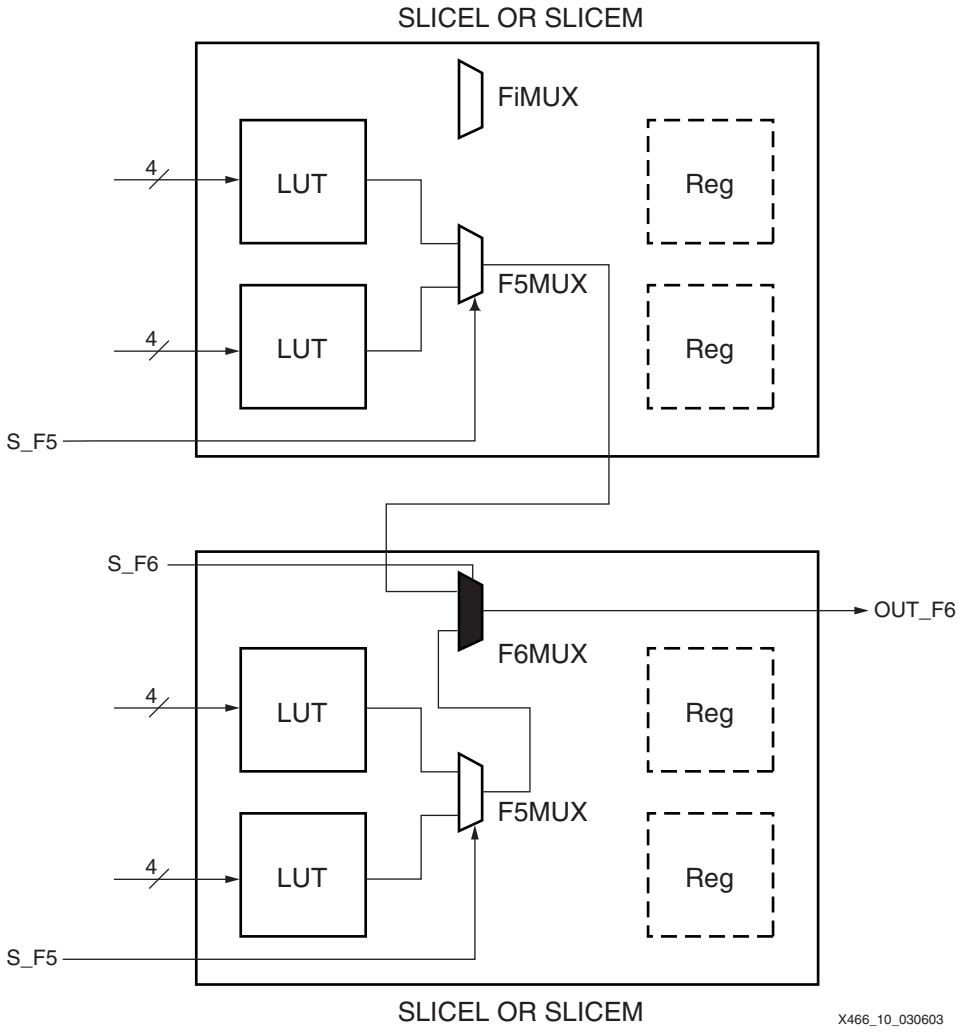

Figure 11: **19-input Function Using F6MUX in Two Slices**

The slice S1 has an F7MUX, designed to combine the outputs of two F6MUXs. Figure 12 illustrates a combinatorial function up to 39 inputs in a Spartan-3 CLB.

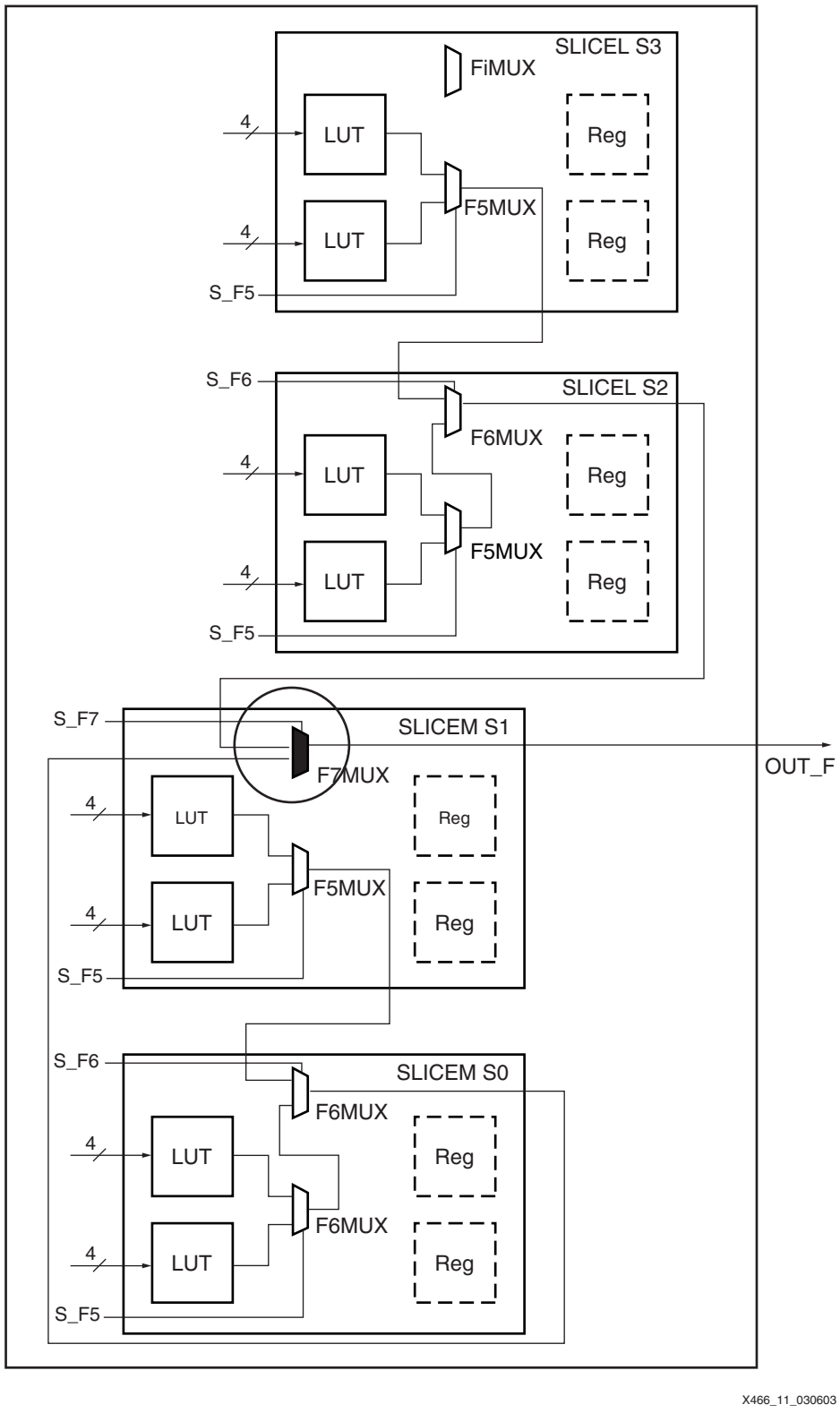

Figure 12: **39-input Function Using F7MUX in One CLB**

The slice S3 of each CLB has an F8MUX. Combinatorial functions of up to 79 inputs fit in two CLBs as shown in Figure 13. The outputs of two F7MUXs are combined through dedicated routing resources between two adjacent CLBs in a column.

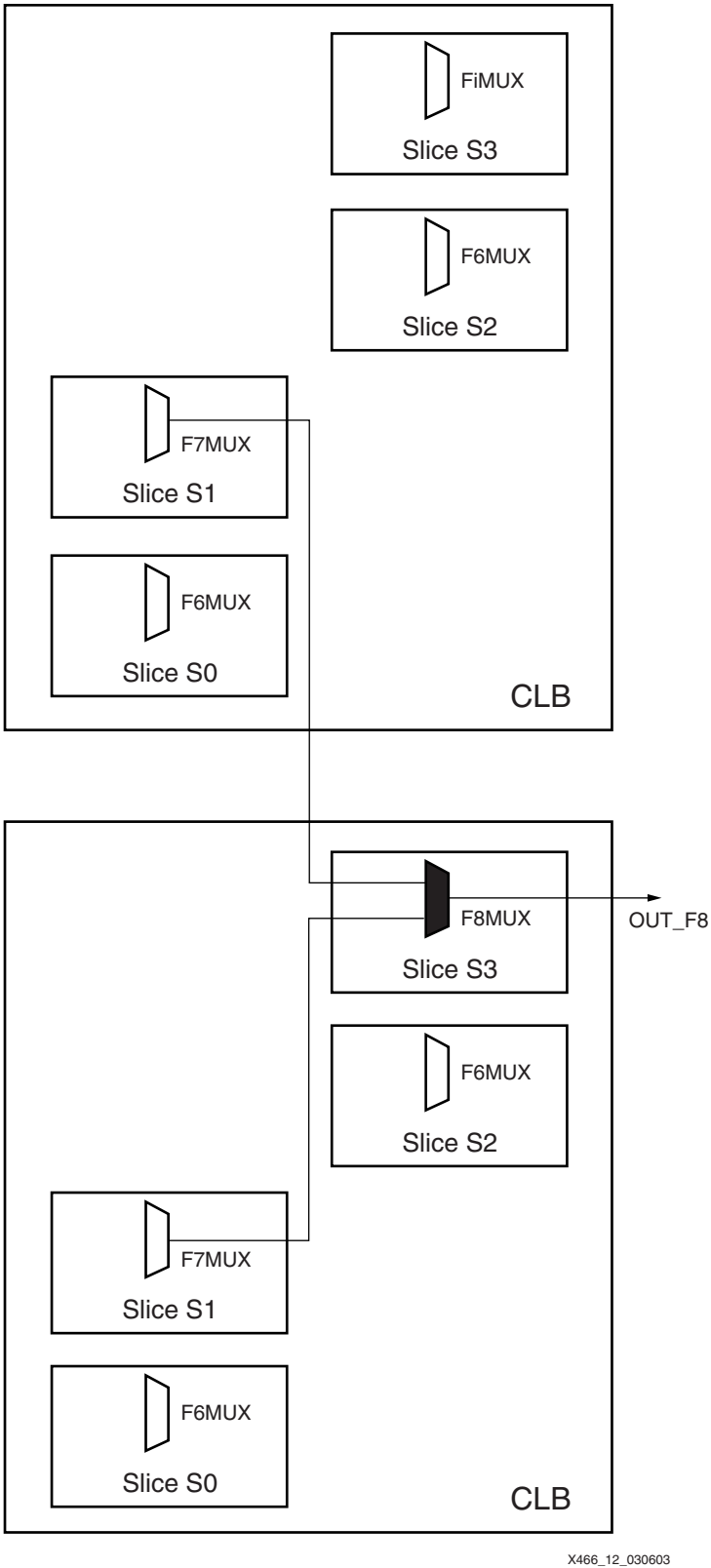

Figure 13: **79-input Function Using F8MUX in Two Adjacent CLBs**

Timing Parameters

There are several possible paths through the CLB multiplexers. The two types of multiplexers are considered separately (F5MUX and FiMUX). Each multiplexer type has two types of inputs: data inputs and select lines. The output of the mux drives the local interconnect through the F5 and FX CLB pins, the general interconnect through the X and Y CLB pins, or the D input on the flip-flop. See Figure 9, page 7 for a block diagram showing dedicated multiplexers in a Spartan-3 Generation CLB. Note that although the mux functionality is identical between the slices with memory and those without, the timing values are independent and may vary slightly.

Although the multiplexers are connected in series inside the CLB, each mux actually feeds a CLB output pin, which feeds back to an input pin through zero-delay local interconnect. Thus each reported block delay element will have only one mux from input to output. The Spartan-3 Generation architecture improves on the Virtex™-II architecture by providing a direct path from the F5MUX or FiMUX to the flip-flop in the CLB.

Table 2: **Multiplexer Timing Paths**

Symbol	CLB Input	Through	CLB Output
t_{IF5}	F/G LUT Inputs	LUT and F5MUX Inputs	F5
t_{IF5X}	F/G LUT Inputs	LUT and F5MUX Inputs	X
t_{IF5CK}	F/G LUT Inputs	LUT and F5MUX Inputs	D input on flip-flop
t_{BXF5}	BX	F5MUX Select	F5
t_{BXX}	BX	F5MUX Select	X
t_{INAFX}	FXINA	FiMUX Inputs	FX
t_{INBFX}	FXINB	FiMUX Inputs	FX
t_{IF6Y}	FXINA or FXINB	FiMUX Inputs	Y
t_{BYFX}	BY	FiMUX Select	FX
t_{BYY}	BY	FiMUX Select	Y

Programmable Polarity

As with most resources in the Spartan-3 Generation FPGA, inverters are free in large multiplexers. The functions in the LUT can have inverters added to inputs or outputs with no effect on performance or utilization. The control inputs to the F5MUX (BX) and FiMUX (BY) have programmable polarity inside the CLB.

Floorplanning Multiplexers

The wide multiplexers force a particular placement on the LUTs being combined. The LUTs must always be in the same slice for the F5MUX and in adjacent vertical slices for the wider muxes. This vertical orientation aligns nicely with the arithmetic logic.

The wide multiplexers cannot be used in conjunction with the arithmetic logic because the arithmetic XOR gate is multiplexed with the F5MUX result. Also, the 32x1 configuration of the distributed RAM uses the F5MUX for the fifth address input.

Related Uses of Multiplexers

Multiplexers and Three-State Buffers

The LUT and mux resources multiplex one of several input signals onto an internal routing resource, using the routing like an internal bus. This is equivalent to the BUFT-based multiplexers found in other FPGA architectures. In most modern FPGA families, these three-state buffers actually are implemented as dedicated logic gates to avoid possible contention when more than one is enabled at a time. The Spartan-3 family reduces die size and cost by

eliminating the overhead of these internal three-state buffer gates. Instead, internal functions defined as a three-state buffer in the Spartan-3 family must be implemented in the LUTs and dedicated muxes.

The CLB multiplexers provide binary encoding of the select lines, requiring fewer signals than the one-hot encoding of the BUFT-based multiplexers. CLB-based multiplexers have no limit on width as BUFT-based multiplexers did, nor any special placement considerations.

The BUFT component, representing a three-state buffer, is not available in the Spartan-3 library, except for the output function in the IOBs. The CORE Generator™ functions of the BUFT-based Multiplexer (and the equivalent BUFE-based Multiplexer) will be implemented as multiplexers in the CLBs.

Using Memory in Place of Multiplexers

To optimize designs, consider replacing multiplexers with memories. A 4:1 mux requires two LUTs and an F5MUX. If the inputs are static, the same function can be thought of as a 4-bit memory and can fit in less than one LUT. In fact, the LUT can be considered to be a 16:1 mux with the LUT inputs serving as the select lines. In any situation where the mux inputs are static, a memory-based implementation saves resources by using the built-in address decode as the mux logic. The 32x1 distributed RAM uses the F5MUX for the fifth address input. For more information, see Xilinx application note XAPP464: “Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs”.

A 4:1 mux with changeable inputs still can be built in one level of logic using the LUT RAM by reprogramming the RAM as the method of selecting one of the four inputs. An easy way of doing this is to use the SRL16 mode to write data into the RAM in 16 clock cycles. For more information, see Xilinx application note XAPP465: “Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Generation FPGAs.”

Creative design concepts such as these can save significant resources. More information is found in the TechXclusive “Performance + Time = Memory (Cost-saving with 3-D Design)”.

Other Multiplexers

The CLB also contains other multiplexers for routing signals through the logic resources. The CYMUX for propagating carry signals is the only other dynamic mux. Several other muxes are used for selecting one of multiple paths. One is called the FXMUX in the FPGA Editor, since it routes the F LUT signal to the X CLB output. Do not confuse this static mux with the FXMUX name that is sometimes used for the FiMUX described here.

When multiplexing clock signals, remember to use the BUFGMUX, which helps eliminate glitches on the resulting clock. Another special multiplexer is found in the I/O to support DDR interfaces. The DDR mux combines two signals onto one output by automatically muxing back and forth between them as they are clocked into the IOB. See the Spartan-3 Generation data sheets for more information on these other multiplexing features.

Designing with Multiplexers

There are several ways multiplexers can be used in a design. The most common is to simply have them inferred by synthesis tools when appropriate for a design. Library primitives can be used to instantiate specific multiplexers. This document provides HDL submodules that combine the library primitives into larger muxes. The CORE Generator system includes the Bus Multiplexer and Bit Multiplexer functions, and many other CORE solutions take advantage of the dedicated multiplexers.

Inference

Multiplexers are typically inferred by a conditional statement, most commonly the CASE or IF-THEN-ELSE statement. The IF statement generally produces priority-encoded logic. The CASE statement is more likely to generate an optimized multiplexer.

Synthesis options may determine whether multiplexers are inferred and how they are implemented. For XST, the MUX_EXTRACT constraint specifies whether multiplexers are inferred, and the MUX_STYLE constraint specifies whether they are implemented in the dedicated logic multiplexers or the carry multiplexers (CY_MUX). The default is to infer automatically the best resource.

CASE statements should be full (all branches defined) to avoid creating a latch. Undefined branches assume the current value needs to be maintained, implying memory. They also should be parallel (branch conditions all mutually exclusive) to avoid a priority encoder. Some synthesis tools, such as XST, have options to assume full and parallel CASE statements even if not written that way. It is good practice to include a “When Others” (VHDL) or “Default” (Verilog) branch to make sure even undefined inputs do not generate a latch.

An IF statement can contain a set of different expressions while a CASE statement is evaluated against a common controlling expression. In general, use the CASE statement for complex decoding and use the IF statement for speed critical paths.

Most current synthesis tools can determine if the IF-ELSIF conditions are mutually exclusive, and will not create extra logic to build the priority tree. The following are points to consider when writing IF statements:

- Make sure that all outputs are defined in all branches of an IF statement. If not, they can create latches or long equations on the CE signal. A good way to prevent this is to have default values for all outputs before the IF statements.
- Limit the number of input signals into an IF statement to reduce the number of logic levels. If there are a large number of input signals, see if some of them can be predecoded and registered before the IF statement.
- Avoid bringing the dataflow into a complex IF statement. Only control signals should be generated in complex IF-ELSE statements.

Make sure you do not write the code such that your synthesis tool will infer BUFT-based multiplexers. A BUFT-based multiplexer usually requires a statement with a "Z" value. Some synthesis tools might automatically or optionally convert BUFT logic to multiplexers.

A decoder is a special case of a multiplexer where the inputs are fixed as one-hot values. Decoders of up to 4:16 in size are easily implemented in individual LUTs for each output and do not need to use the dedicated multiplexers, or they can even use the Carry muxes for high performance.

The following subsections provide examples of 2:1 muxes described using the CASE statement in Verilog and VHDL code.

Verilog Inference

```
module MUX_2_1 (DATA_I, SELECT_I, DATA_O);

input [1:0]DATA_I;
input SELECT_I;

output DATA_O;
reg DATA_O;

always @ (DATA_I or SELECT_I)

  case (SELECT_I)
     1'b0 : DATA_O <= DATA_I[0];
     1'b1 : DATA_O <= DATA_I[1];
     default : DATA_O <= 1'bx;
  endcase

endmodule
```

VHDL Inference

```
entity MUX_2_1 is
    port (
        DATA_I: in std_logic_vector (1 downto 0);
        SELECT_I: in std_logic;
        DATA_O: out std_logic
  );
end MUX_2_1;

architecture MUX_2_1_arch of MUX_2_1 is
--
begin
--
SELECT_PROCESS: process (SELECT_I, DATA_I)
begin
  case SELECT_I is
    when '0' => DATA_O <= DATA_I (0);
    when '1' => DATA_O <= DATA_I (1);
    when others => DATA_O <= 'X';
  end case;
end process SELECT_PROCESS;
--
end MUX_2_1_arch;
```

Library Primitives

Four library primitives are available that offer access to the dedicated multiplexers in each slice: MUXF5, MUXF6, MUXF7, and MUXF8. These use the F5MUX and FiMUX CLB resources (see "Naming Conventions," page 5). Each of the multiplexer primitives looks identical (see Figure 14). The actual selection simply determines where in the CLB the multiplexer can be located, as shown in Table 5.

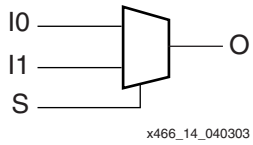

Figure 14: **MUXF5 Primitive**

Table 3: **MUX Inputs and Outputs**

Signal	Function
I0	Input selected when S is Low
I1	Input selected when S is High
S	Select input
LO	Local Output that connects to the F5 or FX CLB pins, which use local feedback to the FXIN inputs to the FiMUX for cascading (see "Modeling Local Output Timing," page 17)
O	General Output that connects to the general-purpose combinatorial or registered outputs of the CLB

Table 4: **MUX Function**

Inputs			Outputs	
S	**I0**	**I1**	**O**	**LO**
0	1	X	1	1
0	0	X	0	0
1	X	1	1	1
1	X	0	0	0

Table 5: **Multiplexer Resources**

Primitive	Slice	Physical Location	Control	Input	Output
MUXF5	S0, S1, S2, S3	F5MUX	S	I0, I1	O
MUXF6	S0, S2	FiMUX	S	I0, I1	O
MUXF7	S1	FiMUX	S	I0, I1	O
MUXF8	S3	FiMUX	S	I0, I1	O

The generic multiplexer components also can take advantage of the dedicated multiplexers. The M2_1 schematic library component is implemented in a LUT, while the larger multiplexers in the library use the F5MUX and FiMUX components.

Enable Signals in Multiplexers

An enable signal on a multiplexer can be used to keep the multiplexer output Low when disabled. Although the dedicated multiplexers do not have enable signals, the enable can be implemented on the preceding 2:1 mux that will be implemented in a LUT. The M4_1E and M8_1E schematic library components are built this way, using the F5MUX and F6MUX for the final result, respectively, while the M16_1E schematic library component keeps the enable on the final mux, forcing it into a LUT instead of the F7MUX. Figure 15 shows the M4_1E schematic library component logic.

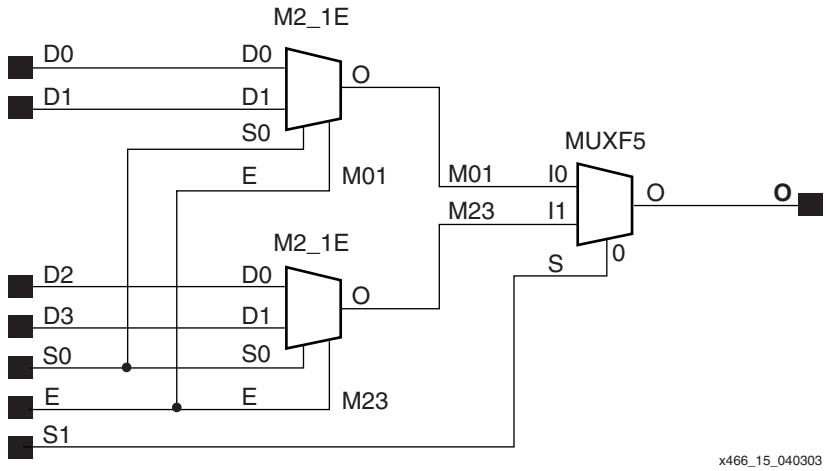

Figure 15: **M4_1E Library Component Logic**

Modeling Local Output Timing

There are also two alternative versions of each library component that are functionally identical but can be used for more accurate timing estimation before implementation. As mentioned previously, the multiplexers can drive one or both CLB outputs. The first output is the special CLB output that feeds directly back through local interconnect to the next multiplexer in series, known as the local output. The second output is the general-purpose CLB output, which can be routed to any other logic. For better pre-implementation timing estimation, the user can substitute special primitives that specify whether to use the local output timing or the general-purpose output timing. The MUXF5_L primitive models the local output, while the MUXF5_D primitive models both output paths (see Figure 16). The functionality is identical to that for the MUXF5 primitive.

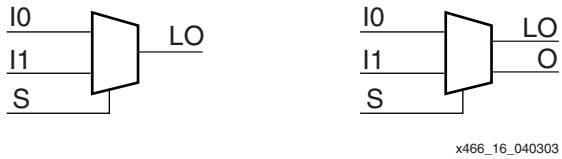

Figure 16: **MUXF5_L and MUXF5_D Primitives to Model Local Output Timing**

Submodules

In addition to the primitives, five submodules that implement multiplexers from 2:1 to 32:1 are provided in VHDL and Verilog code. Synthesis tools can automatically infer the above primitives (MUXF5, MUXF6, MUXF7, and MUXF8); however, the submodules described in this section use instantiation of the multiplexers to guarantee an optimized result. Table 6 lists available submodules.

Table 6: **Available Submodules**

Submodule	Multiplexer	Control	Input	Output
MUX_2_1_SUBM	2:1	SELECT_I	DATA_I[1:0]	DATA_O
MUX_4_1_SUBM	4:1	SELECT_I[1:0]	DATA_I[3:0]	DATA_O
MUX_8_1_SUBM	8:1	SELECT_I[2:0]	DATA_I[7:0]	DATA_O
MUX_16_1_SUBM	16:1	SELECT_I[3:0]	DATA_I[15:0]	DATA_O
MUX_32_1_SUBM	32:1	SELECT_I[4:0]	DATA_I[31:0]	DATA_O

Port Signals

Data In — DATA_I

The data input provides the data to be selected by the SELECT_I signal(s).

Control In — SELECT_I

The select input signal or bus determines the DATA_I signal to be connected to the output DATA_O. For example, the MUX_4_1_SUBM multiplexer has a 2-bit SELECT_I bus and a 4-bit DATA_I bus. Table 7 shows the DATA_I selected for each SELECT_I value.

Table 7: **Selected Inputs**

SELECT_I[1:0]	DATA_O
0 0	DATA_I[0]
0 1	DATA_I[1]
1 0	DATA_I[2]
1 1	DATA_I[3]

Data Out — DATA_O

The data output O provides the data value (1 bit) selected by the control inputs.

Applications

Multiplexers are used in various applications. These are often inferred by synthesis tools when a “case” statement is used (see the example below). Comparators, encoder-decoders, and wide-input combinatorial functions are optimized when they are based on one level of LUTs and dedicated multiplexer resources of the Spartan-3 CLBs.

VHDL and Verilog Instantiation

The primitives (MUXF5, MUXF6, and so forth) can be instantiated in VHDL or Verilog code, to design wide-input functions.

The submodules (MUX_2_1_SUBM, MUX_4_1_SUBM, and so forth) can be instantiated in VHDL or Verilog code to implement multiplexers. However, the corresponding submodule must be added to the design directory as a hierarchical submodule. For example, if a module is using the MUX_16_1_SUBM, the `MUX_16_1_SUBM.vhd` file (VHDL code) or `MUX_16_1_SUBM.v` file (Verilog code) must be compiled with the design source code. The submodule code can also be “cut and pasted” into the designer source code.

VHDL and Verilog Submodules

VHDL and Verilog submodules are available to implement multiplexers up to 32:1. They illustrate how to design with the MUX resources. When synthesis infers the corresponding MUX resource(s), the VHDL or Verilog code is behavioral code (“case” statement). Otherwise, the equivalent “case” statement is provided in comments and the correct MUX resources are instantiated. However, most synthesis tools support the inference of all of the MUXs. The following examples can be used as guidelines for designing other wide-input functions.

The following submodules are available:

- MUX_2_1_SUBM (behavioral code)
- MUX_4_1_SUBM
- MUX_8_1_SUBM
- MUX_16_1_SUBM
- MUX_32_1_SUBM

The corresponding submodules have to be synthesized with the design.

The submodule MUX_16_1_SUBM is provided in VHDL and Verilog as an example:

VHDL Template

```
-- Module: MUX_16_1_SUBM
-- Description: Multiplexer 16:1
--
-- Device: Spartan-3 Family
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

-- Syntax for Synopsys FPGA Express
-- pragma translate_off
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
-- pragma translate_on

entity MUX_16_1_SUBM is
    port (
```

```
        DATA_I: in std_logic_vector (15 downto 0);
        SELECT_I: in std_logic_vector (3 downto 0);
        DATA_O: out std_logic
  );
end MUX_16_1_SUBM;

architecture MUX_16_1_SUBM_arch of MUX_16_1_SUBM is
-- Component Declarations:
component MUXF7
    port (
    I0: in std_logic;
    I1: in std_logic;
    S: in std_logic;
    O: out std_logic
    );
end component;
--
-- Signal Declarations:
signal DATA_MSB : std_logic;
signal DATA_LSB : std_logic;
--
begin
--
-- If synthesis tools support MUXF7 :
--SELECT_PROCESS: process (SELECT_I, DATA_I)
--begin
--case SELECT_I is
--  when "0000" => DATA_O <= DATA_I (0);
--  when "0001" => DATA_O <= DATA_I (1);
--  when "0010" => DATA_O <= DATA_I (2);
--  when "0011" => DATA_O <= DATA_I (3);
--  when "0100" => DATA_O <= DATA_I (4);
--  when "0101" => DATA_O <= DATA_I (5);
--  when "0110" => DATA_O <= DATA_I (6);
--  when "0111" => DATA_O <= DATA_I (7);
--  when "1000" => DATA_O <= DATA_I (8);
--  when "1001" => DATA_O <= DATA_I (9);
--  when "1010" => DATA_O <= DATA_I (10);
--  when "1011" => DATA_O <= DATA_I (11);
--  when "1100" => DATA_O <= DATA_I (12);
--  when "1101" => DATA_O <= DATA_I (13);
--  when "1110" => DATA_O <= DATA_I (14);
--  when "1111" => DATA_O <= DATA_I (15);
--  when others => DATA_O <= 'X';
--end case;
--end process SELECT_PROCESS;
--
-- If synthesis tools DO NOT support MUXF7 :
SELECT_PROCESS_LSB: process (SELECT_I, DATA_I)
begin
  case SELECT_I (2 downto 0) is
    when "000" => DATA_LSB <= DATA_I (0);
    when "001" => DATA_LSB <= DATA_I (1);
    when "010" => DATA_LSB <= DATA_I (2);
    when "011" => DATA_LSB <= DATA_I (3);
    when "100" => DATA_LSB <= DATA_I (4);
    when "101" => DATA_LSB <= DATA_I (5);
    when "110" => DATA_LSB <= DATA_I (6);
    when "111" => DATA_LSB <= DATA_I (7);
    when others => DATA_LSB <= 'X';
  end case;
end process SELECT_PROCESS_LSB;
```

```
--
SELECT_PROCESS_MSB: process (SELECT_I, DATA_I)
begin
  case SELECT_I (2 downto 0) is
    when "000" => DATA_MSB <= DATA_I (8);
    when "001" => DATA_MSB <= DATA_I (9);
    when "010" => DATA_MSB <= DATA_I (10);
    when "011" => DATA_MSB <= DATA_I (11);
    when "100" => DATA_MSB <= DATA_I (12);
    when "101" => DATA_MSB <= DATA_I (13);
    when "110" => DATA_MSB <= DATA_I (14);
    when "111" => DATA_MSB <= DATA_I (15);
    when others => DATA_MSB <= 'X';
  end case;
end process SELECT_PROCESS_MSB;
--
-- MUXF7 instantiation
U_MUXF7: MUXF7
    port map (
    I0 => DATA_LSB,
    I1 => DATA_MSB,
    S  => SELECT_I (3),
    O  => DATA_O
    );
--
end MUX_16_1_SUBM_arch;
--
```

Verilog Template

```
// Module: MUX_16_1_SUBM
//
// Description: Multiplexer 16:1
// Device: Spartan-3 Family
//-----------------------------------------------------------------------
//
module MUX_16_1_SUBM (DATA_I, SELECT_I, DATA_O);

input [15:0]DATA_I;
input [3:0]SELECT_I;

output DATA_O;

wire [2:0]SELECT;

reg DATA_LSB;
reg DATA_MSB;

assign SELECT[2:0] = SELECT_I[2:0];

/*
//If synthesis tools support MUXF7 :
always @ (DATA_I or SELECT_I)

    case (SELECT_I)
     4'b0000 : DATA_O <= DATA_I[0];
     4'b0001 : DATA_O <= DATA_I[1];
     4'b0010 : DATA_O <= DATA_I[2];
     4'b0011 : DATA_O <= DATA_I[3];
     4'b0100 : DATA_O <= DATA_I[4];
     4'b0101 : DATA_O <= DATA_I[5];
     4'b0110 : DATA_O <= DATA_I[6];
```

```
     4'b0111 : DATA_O <= DATA_I[7];
     4'b1000 : DATA_O <= DATA_I[8];
     4'b1001 : DATA_O <= DATA_I[9];
     4'b1010 : DATA_O <= DATA_I[10];
     4'b1011 : DATA_O <= DATA_I[11];
     4'b1100 : DATA_O <= DATA_I[12];
     4'b1101 : DATA_O <= DATA_I[13];
     4'b1110 : DATA_O <= DATA_I[14];
     4'b1111 : DATA_O <= DATA_I[15];
     default : DATA_O <= 1'bx;
    endcase
*/
//If synthesis tools do not support MUXF7 :
always @ (SELECT or DATA_I)

    case (SELECT)
     3'b000 : DATA_LSB <= DATA_I[0];
     3'b001 : DATA_LSB <= DATA_I[1];
     3'b010 : DATA_LSB <= DATA_I[2];
     3'b011 : DATA_LSB <= DATA_I[3];
     3'b100 : DATA_LSB <= DATA_I[4];
     3'b101 : DATA_LSB <= DATA_I[5];
     3'b110 : DATA_LSB <= DATA_I[6];
     3'b111 : DATA_LSB <= DATA_I[7];
     default : DATA_LSB <= 1'bx;
    endcase

always @ (SELECT or DATA_I)

    case (SELECT)
     3'b000 : DATA_MSB <= DATA_I[8];
     3'b001 : DATA_MSB <= DATA_I[9];
     3'b010 : DATA_MSB <= DATA_I[10];
     3'b011 : DATA_MSB <= DATA_I[11];
     3'b100 : DATA_MSB <= DATA_I[12];
     3'b101 : DATA_MSB <= DATA_I[13];
     3'b110 : DATA_MSB <= DATA_I[14];
     3'b111 : DATA_MSB <= DATA_I[15];
     default : DATA_MSB <= 1'bx;
    endcase

// MUXF7 instantiation

MUXF7 U_MUXF7   (.I0(DATA_LSB),
     .I1(DATA_MSB),
     .S(SELECT_I[3]),
     .O(DATA_O)
     );
endmodule
```

CORE Generator System

The CORE Generator system offers the basic logic functions of the Bit Multiplexer and the Bus Multiplexer. The Bit Multiplexer, shown in Figure 17, supports sizes up to 256 inputs. The Bus Multiplexer, shown in Figure 18, supports muxes of up to 32 inputs for buses of up to 256 bits each. These core solutions have a parameter Mux Type to select a BUFT or LUT based multiplexer. Select the appropriate radio button in the CORE Generator system for the construction of the multiplexer. The default setting is LUT based, which is required for Spartan-3 Generation multiplexers. The CORE Generator system also offers options for registering the output of the multiplexer.

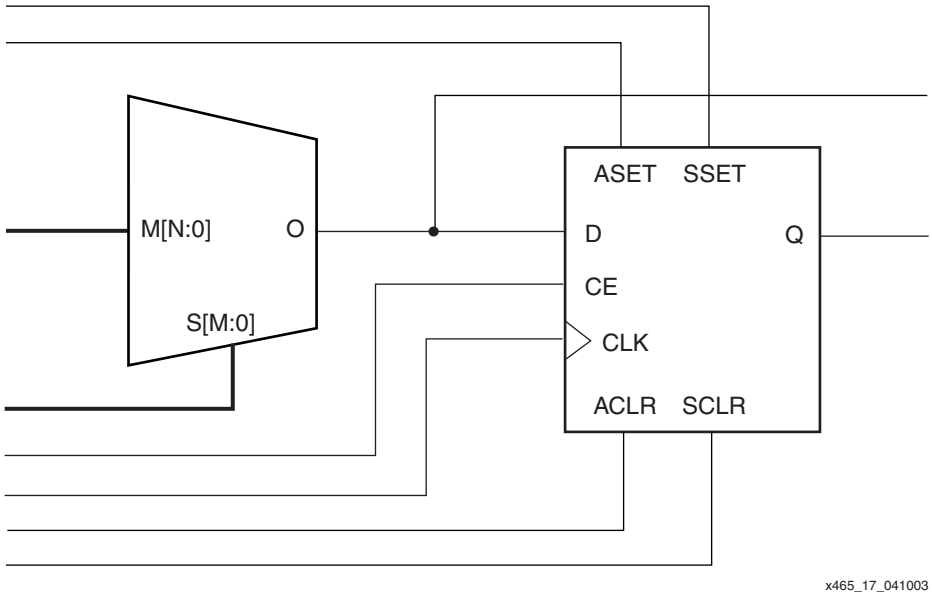

Figure 17: **Bit Multiplexer CORE Symbol**

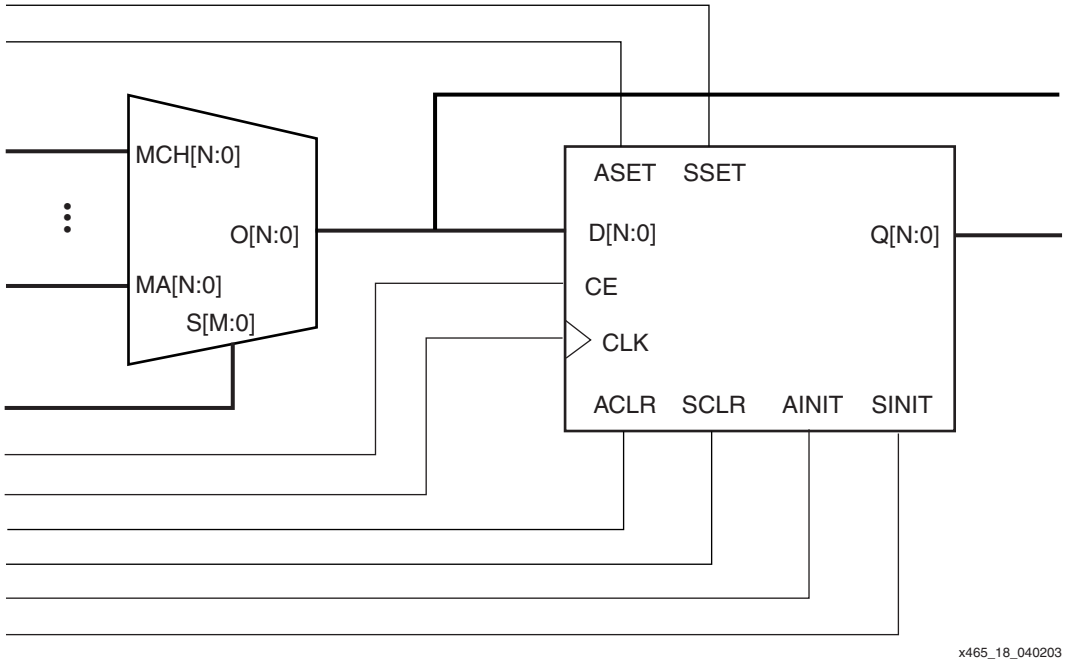

Figure 18: **Bus Multiplexer CORE Symbol**

The CORE Generator system also offers the specific functions of the BUFT-based Multiplexer (and the equivalent BUFE-based Multiplexer). As with the generic Bit and Bus Multiplexers, they are implemented in LUTs and/or muxes.

Summary

The dedicated multiplexers in the Spartan-3 Generation architecture enable wider functions than possible in the four-input LUTs. These multiplexers are automatically used by the software tools but careful coding can help optimize their use to minimize resource requirements and improve performance of designs.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/10/03	1.0	Initial Xilinx release.
07/05/03	1.0.1	Changed title.
05/20/05	1.1	Included all Spartan-3 Generation families, including Spartan-3E and Spartan-3L devices. Made additional clarifications.