XILINX®

XAPP466 (v1.0) 2003.4.10

# Spartan-3 デバイスで専用マルチプレクサを使用

## 概要

Spartan-3 アーキテクチャには、CLB に専用マルチプレクサが含まれています。これらを使用することによって、パフォーマンスが向上するだけでなく、多入力マルチプレクサおよび多入力ファンクションの集積度を上げることができます。32:1 マルチプレクサは、1 つのロジック レベルでの構成が可能になり、あるブール ロジック ファンクションでは 79 入力まで対応できます。

## はじめに

マルチプレクサは、複数の入力から 1 つを出力するブロックで多くのロジック デザインで使用されます。Spartan-3 FPGA では、LUT 内の小規模なマルチプレクサや専用マルチプレクサ リソースを使用し多入力なマルチプレクサを非常に効果的に実現できます。たとえば、Spartan-3 のスライスでは 4 : 1 マルチプレクサ、各 CLB では最大 16 : 1 までのマルチプレクサを構成でき、さらに 2 つの CLB では最大 32 : 1 までのマルチプレクサを構成できます。これらのロジック リソースは、汎用のロジック ファンクションにも幅広く使用でき、コンパレータ、エンコーダ、デコーダなどのアプリケーションでは、ケース文を使用することによって最適なソリューションとなります。このようなマルチプレクサ リソースは、ザイリンクス開発システムで自動的に使用されます。

このアプリケーション ノートでは、Spartan-3 アーキテクチャにおける専用マルチプレクサの使用について説明します。マルチプレクサに関連の信号およびパラメータを示し、デザインでマルチプレクサを用いたデザインの推奨する使用方法やガイドラインについても説明します。

### 専用マルチプレクサの利点

Spartan-3 FPGA は、4 入力の LUT を基本に構成されており、これらの 4 入力で可能なすべてのファンクションを実現できます。1 つの LUT で構成できる最も大規模なマルチプレクサは、4 番目の入力をイネーブル入力としても使用できる 2:1 マルチプレクサです。さらに大規模なマルチプレクサを実現するには、複数の LUT をカスケードします。たとえば、4 : 1 マルチプレクサを構成する場合は、2 つの LUT の出力を 3 番目の LUT に入力します。ただし、この場合 2 つのフル レベル ロジックの遅延と LUT 間での配線遅延が追加されます。特別なリソースを使用せずに 8 : 1 マルチプレクサを構成するには、7 つの LUT を使用しますが、この場合も 3 レベルのロジック遅延と 2 レベルの配線遅延が追加されます。図 1 に 8 : 1 マルチプレクサを示します。

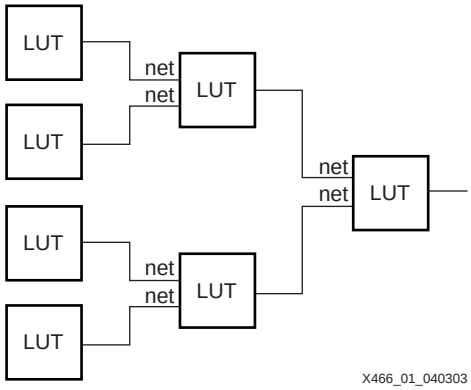


図 1: 3 レベルのロジックで 7 つの LUT を使用して構成された 8 : 1 マルチプレクサ

Spartan-3 FPGA では、マルチプレクサの高速化および高集積度化を実現するため、LUT ベースのロジックと置き換えられた 2 : 1 専用マルチプレクサが、すべての LUT についています。このようなロジックの 1 つが F5MUX と示されています。この F5MUX は、隣接する LUT の出力を結合し 4 : 1 マルチプレクサを構成できます。以下の図の各 LUT ペアで実現されたマルチプレクサは、さらに多入力なファンクションに出力を入力します。このファンクションは、CLB でのマルチプレクサの位置により機能が異なり、FiMUX と示され、FiMUX の i の値には 6、7、または 8 が入ります。たとえば、8 : 1 マルチプレクサを構成するには 2 つの F5MUX エレメントの出力を F6MUX に入力します。これについては、図 2 を参照してください。LUT からマルチプレクサへの接続およびマルチプレクサ間の接続は、専用となっているため配線遅延はありません。LUT や専用マルチプレクサの出力を結合することにより多入力なマルチプレクサを非常に効果的に実現できます。

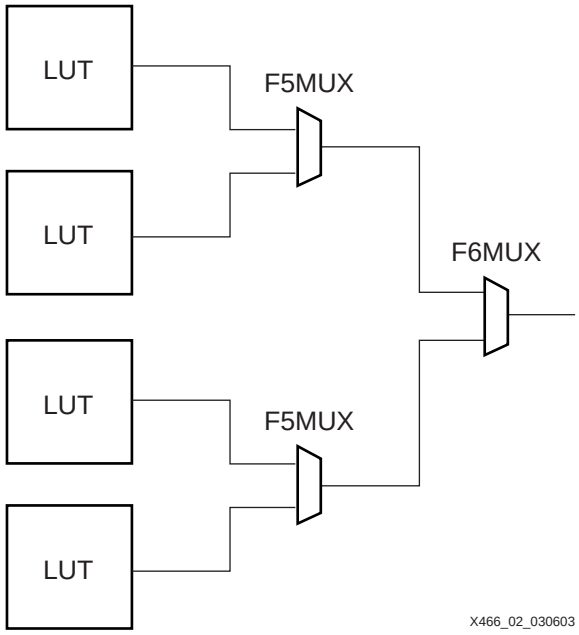


図 2: 1 レベルのロジックで 4 つの LUT を使用して構成された 8:1 マルチプレクサ

## Spartan-3 CLB マルチプレクサ リソース

Spartan-3 アーキテクチャは、同一 CLB のアレイで構成されており、各 CLB にはメモリ機能を持つ 2 つの SLICEM およびロジック機能のみを持つ 2 つの SLICEL の 4 つがあります。各スライスのロジックおよびマルチプレクサ リソースはすべて同一で、2 つの LUT、1 つの F5MUX とさらに別のマルチプレクサが 1 つあります（図 3 を参照）。

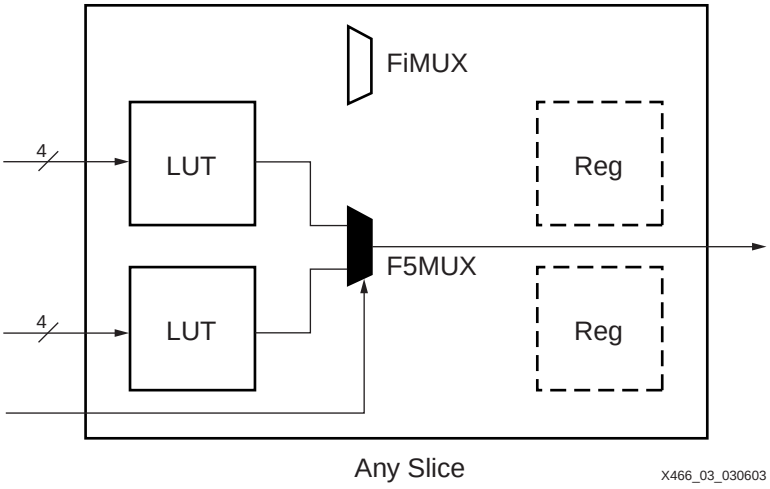


図 3: スライス内の LUT と F5MUX

## F5MUX

F5MUX は、スライスで 2 つの LUT からの出力を 1 つにします。これらの 2 つの LUT に同じ制御入力をもつ 2 : 1 マルチプレクサが含まれている場合、4 : 1 マルチプレクサを構成できます (図 4 参照)。

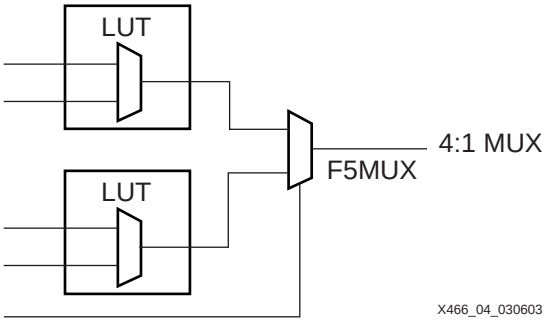


図 4: F5MUX を使用して構成された 4 : 1 マルチプレクサ

F5MUX は、あらゆる 5 入力のブール ロジック ファンクション (図 5 参照) を生成するため、F5MUX と示されます。2 つの LUT に同じ 4 入力のファンクションがそれぞれ含まれている場合、マルチプレクサのセレクタが 5 番目の入力となります。F5MUX は、5 入力のファンクションをインプリメントする場合、3 入力の LUT と同様の効果があり、Spartan-3 FPGA アーキテクチャは、ほかの FPGA アーキテクチャよりも優れています。

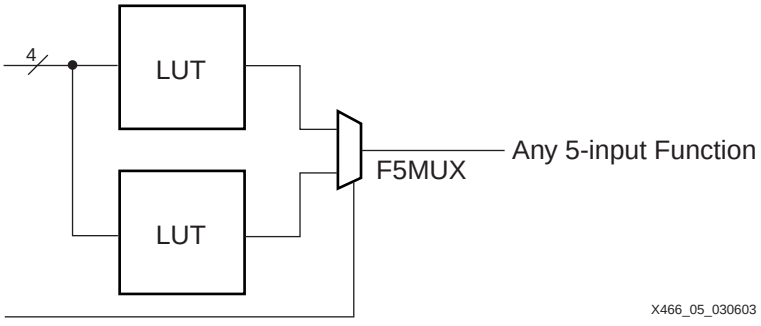


図 5: F5MUX を使用して構成された 5 入力ファンクション

F5MUX では、9 入力のファンクションを 2 つの 4 入力 LUT と 1 つのマルチプレクサに分割することで最大 9 入力までのファンクションも生成できます。(図 6 参照)。

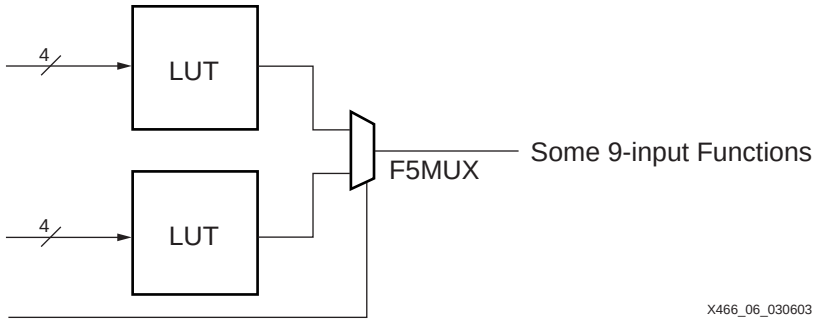


図 6: F5MUX を使用して構成された 9 入力ファンクション

このように F5MUX では、あらゆる 5 入力ファンクション、4 : 1 マルチプレクサの 6 入力ファンクションまたは9 入力ファンクションを生成できます。

## FiMUX

FiMUX は、ロケーションやその他のマルチプレクサとの接続により F6MUX、F7MUX または F8MUX と示されます。

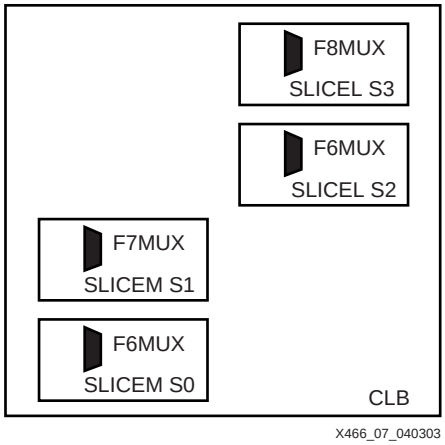


図 7: CLB のマルチプレクサ

これらの各 FiMUX には、1 つ手前のマルチプレクサからの出力が入力されます。たとえば、F7MUX には、2 つの F6MUX の出力が入力されます。FiMUX は、F5MUX と同様にマルチプレクサ以外のファンクションを容易に実現できます。F6MUX は、あらゆる 6 入力のファンクションを実現するため F6MUX と示され、F7MUX は、あらゆる 7 入力のファンクションを実現するため F7MUX と示されます。同様に F8MUX は、8 入力のファンクションを実現するため F8MUX と示されます。

表 1: マルチプレクサの機能

| マルチプレクサ | 用法 | 入力ソース | 各ファンクションの総入力数 | | |
|---|---|---|---|---|---|
| | | | 可能なファンクション | マルチプレクサ | 制限のあるファンクション |
| F5MUX | F5MUX | LUT | 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 |

### ネーミング規則

このアプリケーション ノートおよび Spartan-3 のデータ シートでは、F6MUX、F7MUX、または F8MUX として機能するマルチプレクサは、すべて FiMUX (i=6、7、8) と示されています。これらのマルチプレクサは、X 出力を生成する F5MUX との混乱を避けるため FiMUX と示され、FPGA Editor では FXMUX と示されます。また、FPGA Editor では、FiMUX は常に F6MUX と示され、Timing Analyzer では、FiMUX から CLB ピンまでのパスを示す場合に、このマルチプレクサが、F7MUX や F8MUX として機能していても TIF6Y と示されます。

これらのライブラリ コンポーネントは、それぞれ MUXF5、MUXF6、MUXF7、MUXF8 と示され、MUXF6、MUXF7 および MUXF8 は、FiMUX のライブラリを使用して CLB の指定した相対ロケーションに配置されます。

## 専用のローカル配線

専用マルチプレクサを使用する上での 1 番の利点は、異なったレベル間を専用の配線を使用して接続できることにあります。各マルチプレクサは、CLB でそれぞれ構成されますが、その出力は、配線遅延のないローカル インターコネクトを介し CLB の入力に接続されます。この結果は、マルチプレクサが CLB で直列に接続された場合と同一になります。

F5MUX は、F5 CLB 出力ピンを供給し、同じ CLB 内でその出力を FiMUX 入力 (FXINA および FXINB) のみに接続します。FiMUX の場合は、FX の CLB 出力ピンを供給し、同一 CLB 内で FiMUX (例 F7MUX) または 次の CLB にも出力をフィード バックします。マルチプレクサの結果が必要な場合は、マルチプレクサを汎用の CLB 出力 (F5MUX の場合 X、FiMUX の場合 Y) に接続します。

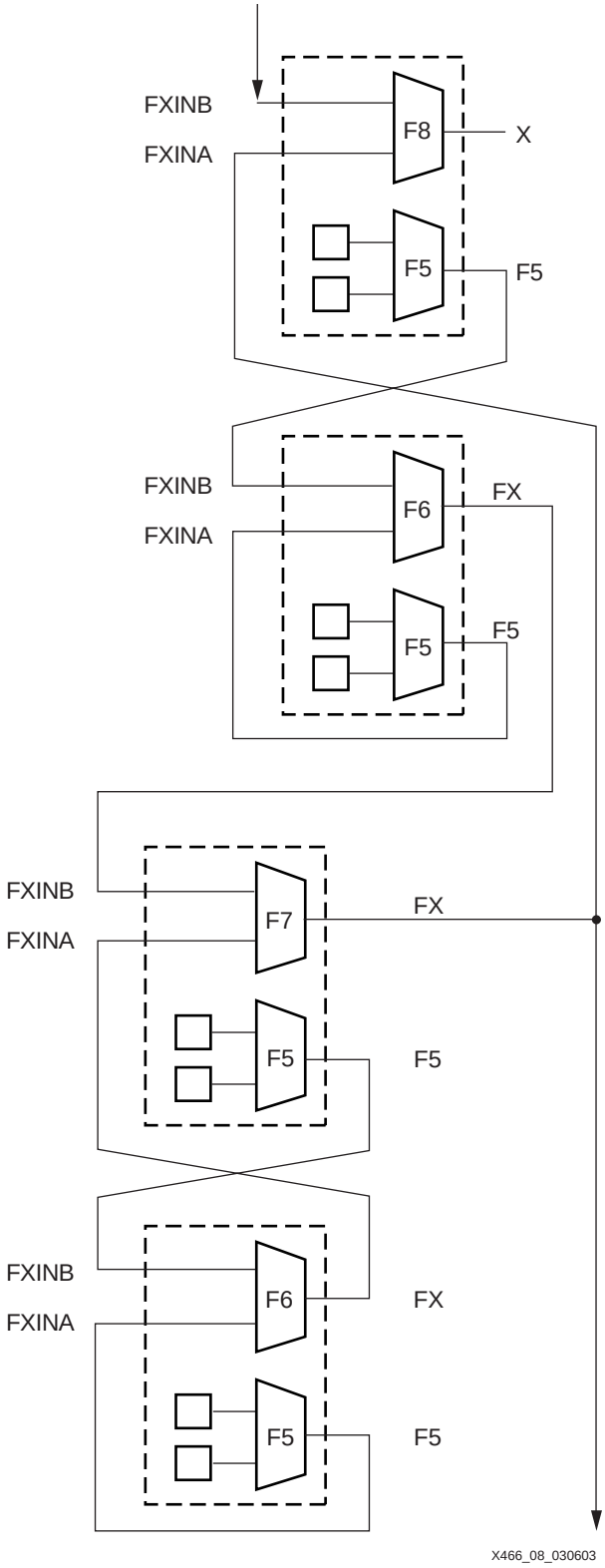


図 8: Spartana-3 の CLB 内でのマルチプレクサと専用のフィードバック

### マルチプレクサのセレクト入力

マルチプレクサのセレクト入力では、汎用の配線を使用します。CLB での F5MUX および FiMUX のセレクト入力は、それぞれ BX 入力と BY 入力です。

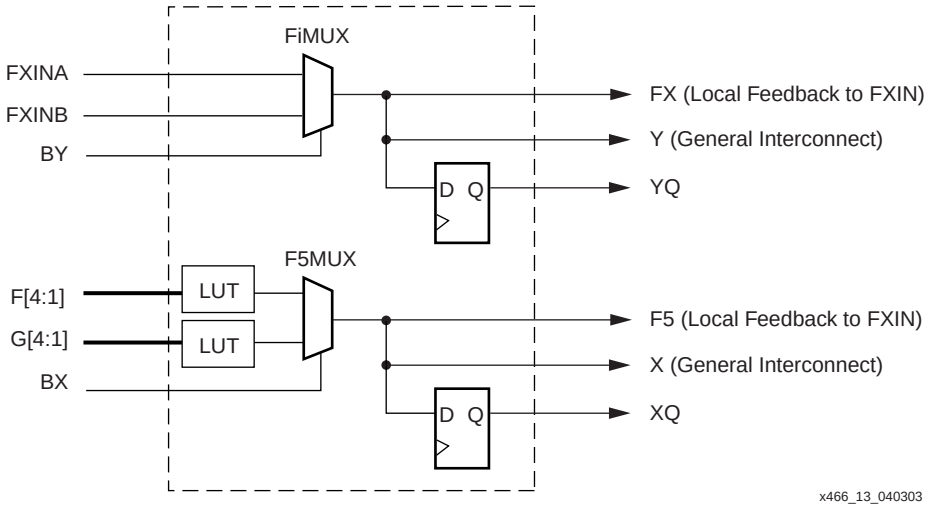


図 9:　Spartan-3 の CLB 内での専用マルチプレクサ

## インプリメンテーションの例

### 多入力マルチプレクサ

各スライスの LUT では、2 : 1 マルチプレクサを構成できます。また、各スライスの F5MUX と 2 つの LUT を使用すると 4 : 1 マルチプレクサを構成できます。同様に、F6MUX と 2 つのスライスでは、8 : 1 マルチプレクサ（図 10 参照）、F7MUX と あらゆる CLB の 4 つのスライスでは、16 : 1 マルチプレクサを構成できます。さらに、F8MUX と 2 つの CLB では、32 : 1 マルチプレクサを構成できます。

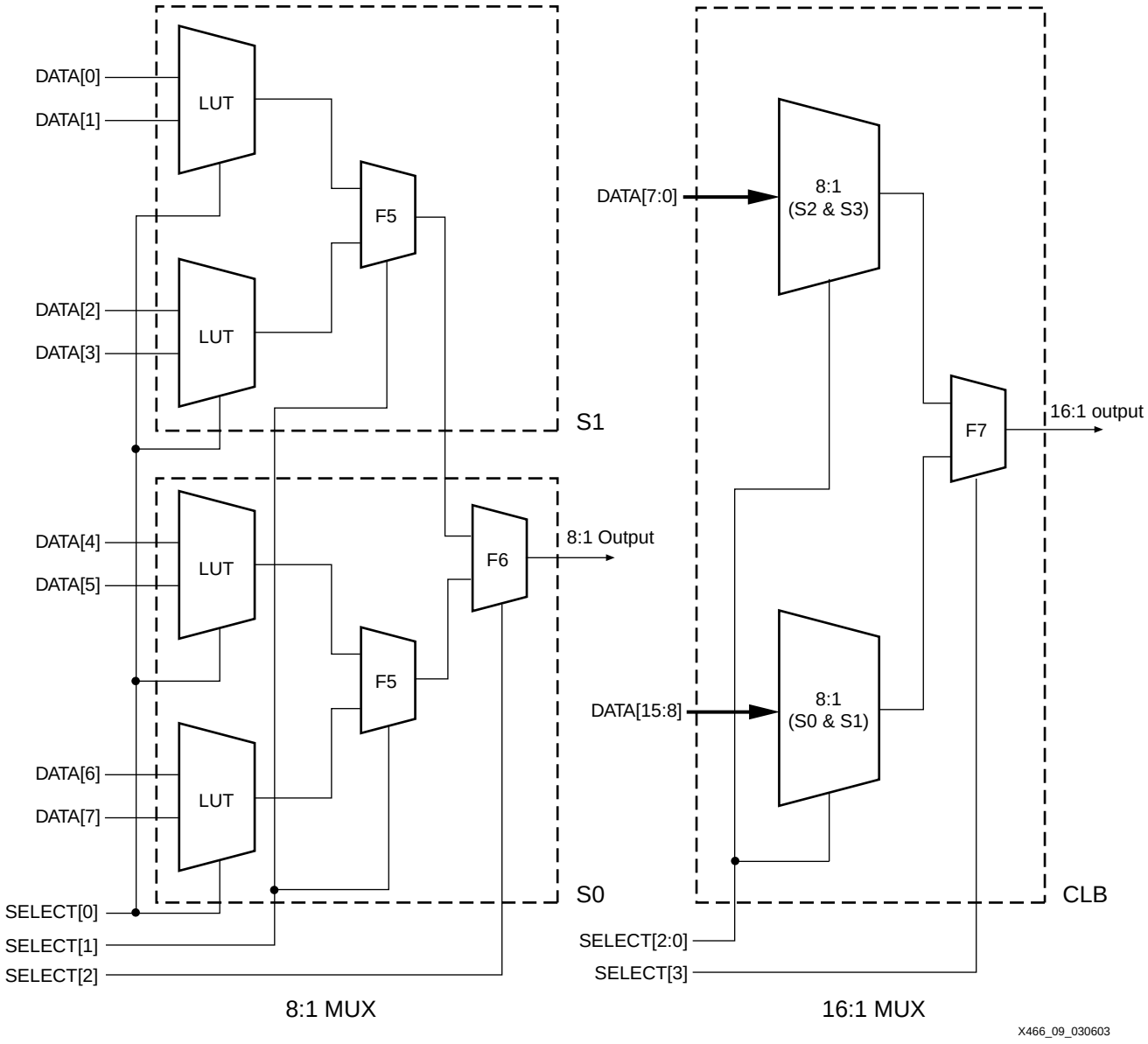


図 10: 8:1 マルチプレクサおよび 16:1 マルチプレクサ

## 多入力ファンクション

S0 と S2 スライスには、2 つの F5MUX からの出力を 1 つにまとめる F6MUX があります。図 11 に、S0 と S1 または S2 と S3 の 2 つのスライス間における最大 19 入力の組み合わせファンクションを示します。

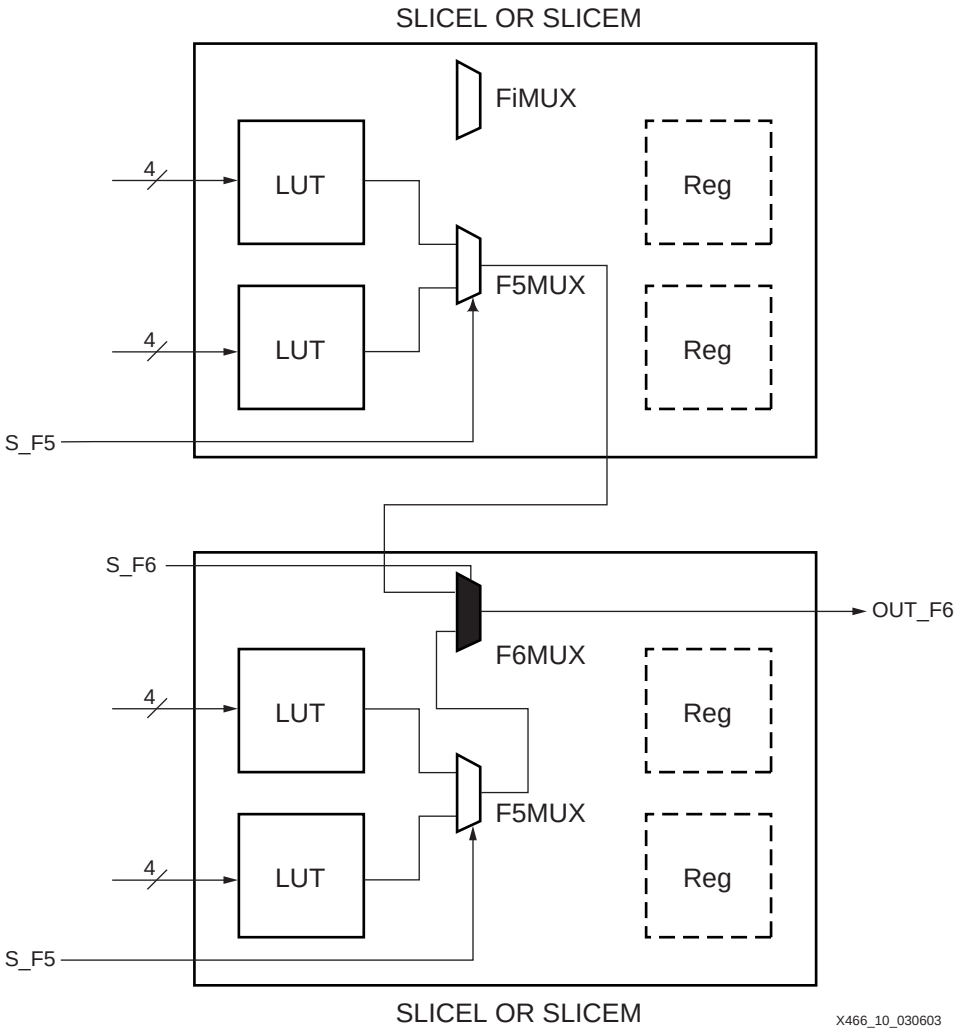


図 11: 2 つのスライスで F6MUX を使用して構成された 19 入力ファンクション

また、S1 スライスには 2 つの F6MUX からの出力を 1 つにする F7MUX があります。図 12 に Spartan-3 の CLB における最大 39 入力の組み合わせファンクションを示します。

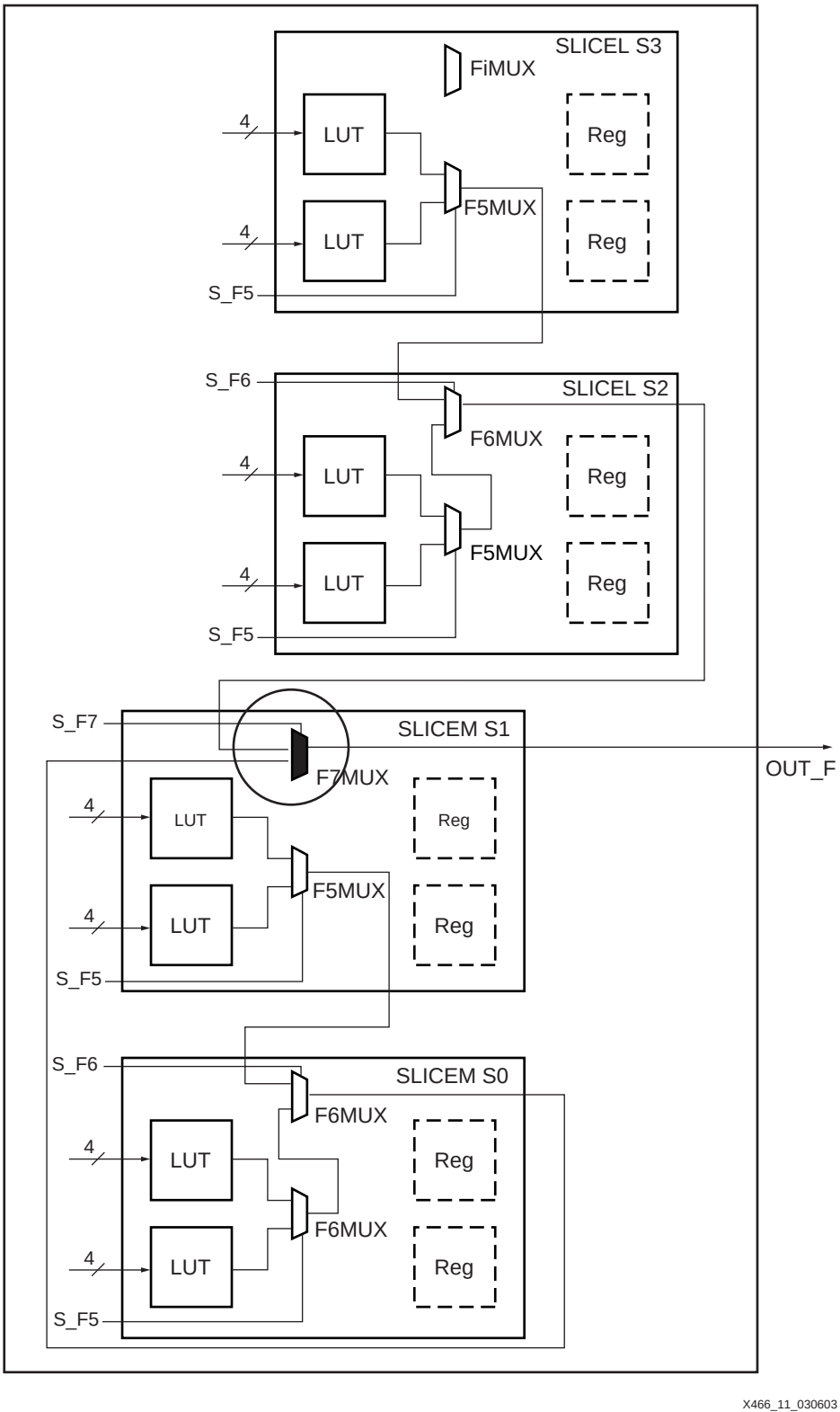


図 12: 1 つの CLBで F7MUX を使用して構成された 39 入力ファンクション

各 CLB の S3 スライスには F8MUX があります。図 13 に 2 つの CLB で実現できる 最大 79 入力の組み合わせファンクションを示します。この 2 つの F7MUX の出力は、同じ列で隣接した 2 つの CLB 間の専用の配線リソースを介して F8MUX に入力されます。

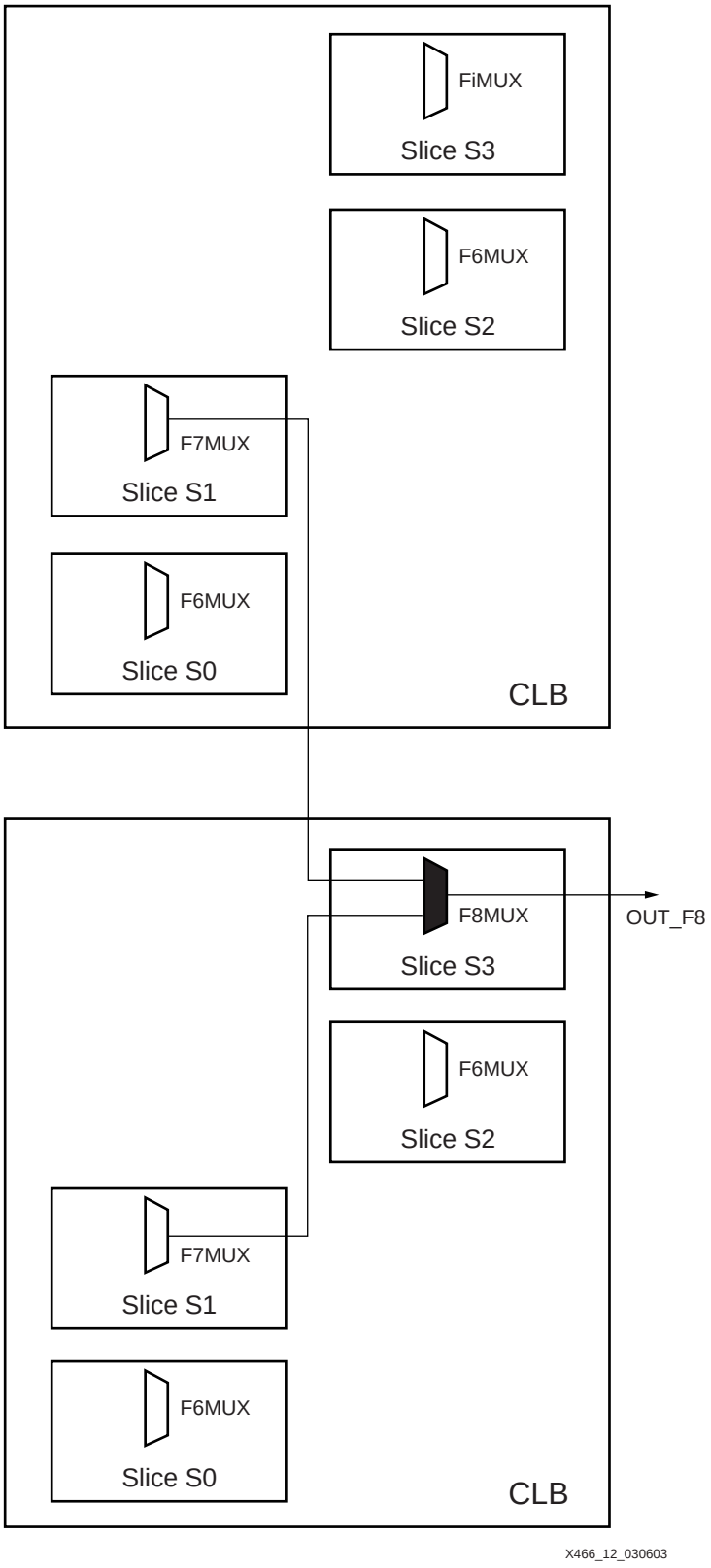


図 13: 2 つの隣接した CLB で F8MUX を使用して構成された 79 入力ファンクション

## タイミング パラメータ

CLB 内のマルチプレクサを配線するには、複数の手段があります。マルチプレクサには、F5MUX と FiMUX の 2 種類があり、各種のマルチプレクサには、それぞれ 2 つの入力 (データ入力およびセレクト ライン) があります。また、マルチプレクサの出力は、F5 および FX の CLB ピンを介してローカル インターコネクトを駆動、X および Y の CLB ピンを介して通常のインターコネクトを駆動、またはフリップフロップで D 入力を駆動します。Spartan-3 の CLB 内の専用マルチプレクサを示すブロック図については、7 ページの 図 9 を参照してください。マルチプレクサは、メモリのあるスライス間およびメモリのないスライス間でも同様に機能しますが、タイミング値はわずかに異なります。

また、マルチプレクサは、CLB 内で直列に接続されていますが、実際には各マルチプレクサでは CLB の出力ピンを使用します。この出力ピンは、遅延のないローカル インターコネクトを介し入力ピンに再接続されます。このため、表示されるブロックの各遅延エレメントには、入力から出力まで 1 つのマルチプレクサしかありません。Spartan-3 のアーキテクチャは、Virtex-II のアーキテクチャに CLB 内の F5MUX または FiMUX からフリップフロップへの直接パスが追加されたものです。

表 2: マルチプレクサのタイミング パス

| Symbol | CLB Input | Through | CLB Output |
|---|---|---|---|
| $t_{IF5}$ | F/G LUT 入力 | LUT および F5MUX 入力 | F5 |
| $t_{IF5X}$ | F/G LUT 入力 | LUT および F5MUX 入力 | X |
| $t_{IF5CK}$ | F/G LUT 入力 | LUT および F5MUX 入力 | フリップフロップでの D 入力 |
| $t_{BXF5}$ | BX | F5MUX セレクト | F5 |
| $t_{BXX}$ | BX | F5MUX セレクト | X |
| $t_{INAFX}$ | FXINA | FiMUX 入力 | FX |
| $t_{INBFX}$ | FXINB | FiMUX 入力 | FX |
| $t_{IF6Y}$ | FXINA または FXINB | FiMUX 入力 | Y |
| $t_{BYFX}$ | BY | FiMUX セレクト | FX |
| $t_{BYY}$ | BY | FiMUX セレクト | Y |

### プログラマブルな極性

多くのリソースと同様に、Spartan-3 FPGA では大規模なマルチプレクサの随所にインバータを挿入できます。LUT の入力または出力ファンクションには、パフォーマンスや稼動に影響を及ぼすことなくインバータを追加できます。これらを追加可能なプログラマブルな極性は、F5MUX (BX) および FiMUX (BY) への制御入力にあります。

## マルチプレクサの使用法

### マルチプレクサとトライステート バッファ

LUT およびマルチプレクサのリソースは、内部バスなどの配線を使用し内部配線リソースで複数の入力信号の 1 つを多重化します。これは、ほかの FPGA アーキテクチャで見られる BUFT ベースのマルチプレクサに相当します。今日の FPGA ファミリでは、このようなトライステート バッファが専用のロジック ゲートとして使用されている場合が多く、2 つ以上の信号が同時にイネーブルになった場合に起こりうる競合を回避します。Spartan-3 ファミリでは、このようなトライステート バッファ ゲートのオーバーヘッドを省くことによりダイ サイズの縮小とコスト削減を実現しています。このため、Spartan-3 の場合、トライステート バッファとして指定された内部ファンクションが、LUT および専用マルチプレクサに搭載されています。

CLB のマルチプレクサは、BUFT ベースのマルチプレクサのワンホット エンコーディングよりも少ないリソースでセレクト ラインをバイナリに エンコードします。CLB ベースのマルチプレクサには、

BUFT ベースのマルチプレクサで定義されているような幅に対する制限や配置に関する制限がありません。

Spartan-3 のライブラリでは、トライステート バッファを示す BUFT コンポーネントは、IOB の出力ファンクションでのみ使用できます。CORE Generator の BUFT ベースのマルチプレクサおよび BUFE ベースと同等のマルチプレクサのファンクションは、マルチプレクサとして CLB で構成されます。

### マルチプレクサとメモリ機能

F5MUX および FiMUX は、CLB の分散メモリとシフト レジスタ機能を拡張する場合にも使用します。この機能については、ここでは説明しません。

### その他の CLB マルチプレクサ

CLB には、ロジック リソースを介して信号を配線する複数のマルチプレクサもあります。これらのうち CYMUX は、唯一のダイナミック マルチプレクサであり、キャリー信号の伝搬に使用します。また、これ以外のマルチプレクサは、複数のパスから 1 つのパスを選択する場合に使用し、FPGA Editor では、LUT の F 信号を CLB の X 出力に配線するマルチプレクサは FXMUX と示されています。このマルチプレクサ名を前述の FiMUX で説明した FXMUX と混同しないようにしてください。

## マルチプレクサを用いたデザイン

マルチプレクサをデザインで使用するには、さまざまな手段があります。この場合、合成ツールを使用しマルチプレクサを推論するという手段が最も一般的で、特別なマルチプレクサのインスタンスを作成する場合は、ライブラリ プリミティブを使用します。このアプリケーション ノートでは、複数のライブラリ プリミティブを組み合わせてマルチプレクサを構成する HDL サブモジュールを示します。CORE Generator には、バス マルチプレクサおよびビット マルチプレクサのファンクションが含まれており、その他の CORE ソリューションでは、専用マルチプレクサを使用します。

### 推論

通常、マルチプレクサは、CASE 文や IF-THEN-ELSE 文といった条件文で推論され、IF 文でプライオリティ エンコーダを生成し、CASE 文で最適化されたマルチプレクサを生成します。

合成オプションでは、マルチプレクサを推論するか否かの設定および構成方法が選択できます。XST では、MUX_EXTRACT 制約でこのような設定を行い、MUX_STYLE 制約でマルチプレクサを専用のロジックマルチプレクサに構成するかキャリー マルチプレクサ (CY_MUX) に構成するかを設定します。デフォルトでは、最適なリソースが自動推論されるように設定されています。

また、不正なラッチの生成を避ける場合は、CASE 文ですべての条件を満たし (すべての分岐を定義)、プライオリティ エンコーディングを避ける場合は、同じ条件が重ならないよう (条件を相互に排他的) にします。ただし、コードがこのような条件のもとに記述されていない場合でも、自動で推測するオプションがついている XST などの合成ツールもあります。

また、合成ツールでは、BUFT ベースのマルチプレクサが推論されるようなコードを記述しないようにしてください。これらを推論するには、Z が含まれた文が必要となります。合成ツールには、自動またはオプションで BUFT ロジックをマルチプレクサに変換するものもあります。

デコーダは、入力がワンホット値に固定されている特殊なマルチプレクサです。4 : 16 までのデコーダは、出力用に各 LUT で容易に実現することが可能であり、専用マルチプレクサを使用する必要もありません。さらに、キャリー マルチプレクサを使用しパフォーマンスを向上させることもできます。

Verilog および VHDL の CASE 文を用いて記述した 2 : 1 マルチプレクサの例を以下に示します。

#### Verilog の場合

```
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 の場合

```
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;
```

## ライブラリ プリミティブ

各スライスの専用マルチプレクサ MUXF5、MUXF6、MUXF7 および MUXF8 への接続には、4 つのライブラリ プリミティブを使用できます。これらのマルチプレクサのプリミティブは、すべて図 14 とまったく同様に示されます。マルチプレクサが、CLB のどのスライスに配置されるかを表 3 に示します。

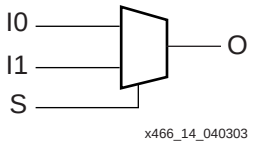


図 14: MUXF5 プリミティブ

表 3: マルチプレクサ リソース

| プリミティブ | スライス | 制御 | 入力 | 出力 |
|---|---|---|---|---|
| MUXF5 | S0, S1, S2, S3 | S | I0, I1 | O |
| MUXF6 | S0, S2 | S | I0, I1 | O |
| MUXF7 | S1 | S | I0, I1 | O |
| MUXF8 | S3 | S | I0, I1 | O |

汎用のマルチプレクサのコンポーネントでも専用マルチプレクサを使用できます。M2_1 コンポーネントは、LUT にインプリメントされており、2:1 より大規模なマルチプレクサは、F5MUX および FiMUX のコンポーネントを使用します。

### マルチプレクサのイネーブル信号

マルチプレクサのイネーブル信号は、ディスエーブルの場合にマルチプレクサの出力を Low に維持するために使用します。専用マルチプレクサにイネーブル信号はありませんが、LUT で構成される 2 : 1 マルチプレクサではこの信号を使用できます。M4_1E および M8_1E のライブラリ コンポーネントは、F5MUX と F6MUX の結果を使用してそれぞれ構成されます。また、M16_1E ライブラリ コンポーネントは、最終段のマルチプレクサでイネーブル信号を維持し、F7MUX ではなく LUT を使用します。

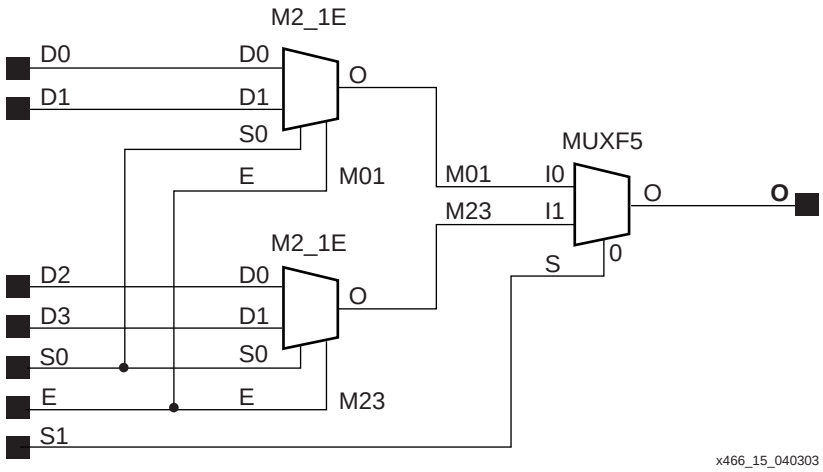


図 15: M4_1E ライブラリ コンポーネント ロジック

### ローカル出力のタイミングのモデル化

各ライブラリ コンポーネントには、機能的には同一の 2 つのバージョンがあり、インプリメンテーション前のタイミング測定をより正確に行うためにこれらの 2 つのバージョンを使用します。このアプリケーション ノートで示されているように、マルチプレクサは、CLB の 1 つまたは 2 つの出力を駆動します。このうちの一方の出力は、ローカル インターコネクトから次のマルチプレクサへ直接フィードバックされるローカル出力で、もう一方は、その他のあらゆるロジックに配線可能な汎用の CLB 出力です。インプリメンテーション前のタイミング測定をより正確に行うため、ローカル出力のタイミングの使用を指定するプリミティブや汎用出力のタイミングの使用を指定するプリミティブに置き換えることができます。MUXF5_L プリミティブは、ローカル出力パスをモデル化し、MUXF5_D プリミティブは 2 つの出力パスをモデル化します (図 16 参照) 。これらのプリミティブは、MUXF5 プリミティブとまったく同様に機能します。


図 16: ローカル出力のタイミングをモデル化する MUXF5_L および MUX5F_D プリミティブ

## サブモジュール

上記のプリミティブのほかに、VHDL および Verilog コードには 2 : 1 から 32 : 1 のマルチプレクサを構成する 5 つのサブモジュールが あります。これらのプリミティブは、合成ツールで自動推論されますが、この 5 つのサブモジュールでは、最適な結果を保証するためマルチプレクサのインスタンシエーションを使用します。表 4 に使用可能なサブモジュールを示します。

表 4: 使用可能なサブモジュール

| サブモジュール | マルチプレクサ | セレクト入力 | 入力 | 出力 |
|---|---|---|---|---|
| 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[8:0] | 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 |

## ポート信号

### データ入力 - DATA_I

データ入力とは、SELECT_I 信号が選択するデータです。

### セレクト入力 - SELECT_I

セレクト入力信号/バスは、DATA_I 信号を DATA_O 出力に接続することを決定します。たとえば、MUX_4_1_SUBM マルチプレクサには、2 ビットの SELECT_I バスと 4 ビットの DATA_I バスがあります。表 5 に、各 SELECT_I 値で選択された DATA_I を示します。

表 5: セレクト入力

| 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] |

### データ出力 - DATA_O

データ出力 O とは、制御入力で選択された 1 ビットのデータ値です。

## アプリケーション

マルチプレクサは、多様なアプリケーションに使用されます。通常は、case 文を用いて合成ツールで推論されます (以下の例を参照)。コンパレータ、エンコーダ - デコーダ、および多入力の組み合わせファンクションは、1 レベルの LUT と Spartan-3 の CLB の専用の MUXFX リソースがベースとなっている場合に最適化されます。

## VHDL および Verilog のインスタンシエーション

多入力ファンクションを実現する場合、MUXF5、MUXF6 などのプリミティブを VHDL または Verilog コードでインスタンシエートします。

また、マルチプレクサを構成する場合は、MUX_2_1_SUBM や MUX_4_1_SUBM などのサブモジュールを VHDL または Verilog コードでインスタンシエートします。ただし、この場合対応するモジュールを階層のサブモジュールとしてデザイン ディレクトリに追加する必要があります。
たとえば、MUX_16_1_SUBM を使用する場合は、MUX_16_1_SUBM.vhd または MUX_16_1_SUBM.v ファイルをデザイン ソース コードにコンパイルします。また、サブモジュール コードを切り取りデザイン ソース コードに貼り付けることもできます。

### VHDL および Verilog のサブモジュール

VHDL および Verilog のサブモジュールでは、最大 32:1 マルチプレクサまでを実現できます。これらのサブモジュールには、マルチプレクサのリソースを用いたデザイン方法が示されています。VHDL お

よび Verilog のビヘイビア コードで case 文を使用すると、合成でマルチプレクサ リソースが推論されます。case 文は、次のようなテンプレートのコメントに 示されており、適切なマルチプレクサもインスタンシエートされていますが、ほとんどの合成ツールでは、すべてのマルチプレクサを推論できます。また、この例を使用しその他の多入力ファンクションも実現できます。

使用可能な VHDL および Verilog のテンプレートは次の通りです。

- MUX_2_1_SUBM (ビヘイビア コード)
- MUX_4_1_SUBM
- MUX_8_1_SUBM
- MUX_16_1_SUBM
- MUX_32_1_SUBM

対応するサブモジュールをデザインで合成します。

VHDL および Verilog の MUX_16_1_SUBM サブ モジュールの例を次に示します。

VHDL のテンプレート

```
-- 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 のテンプレート

```
// 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 システム

CORE Generator システムには、ビット マルチプレクサおよびバス マルチプレクサの BaseBLOX ファンクションがあります。図 17 に示されるビット マルチプレクサは、256 ビットまでの入力に対応しており、図 18 に示されるバス マルチプレクサは、各幅が 256 ビットまでのバスを最大 32 個までサポートします。これらのコア ソリューションには、BUFT または LUT ベースのマルチプレクサを選択するパラメータ マルチプレクサ タイプがあります。このタイプは、CORE Generator で選択できます。デフォルトでは、Spartan-3 のマルチプレクサで必要な LUT ベースが選択されています。また、CORE Generator には、マルチプレクサの出力をレジスタするオプションもついています。

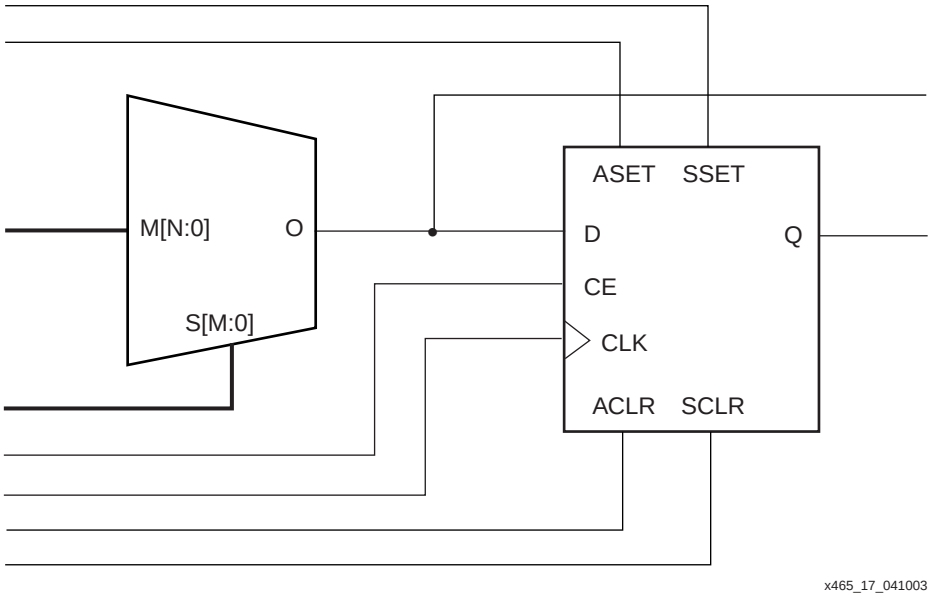


図 17: ビット マルチプレクサのコア シンボル

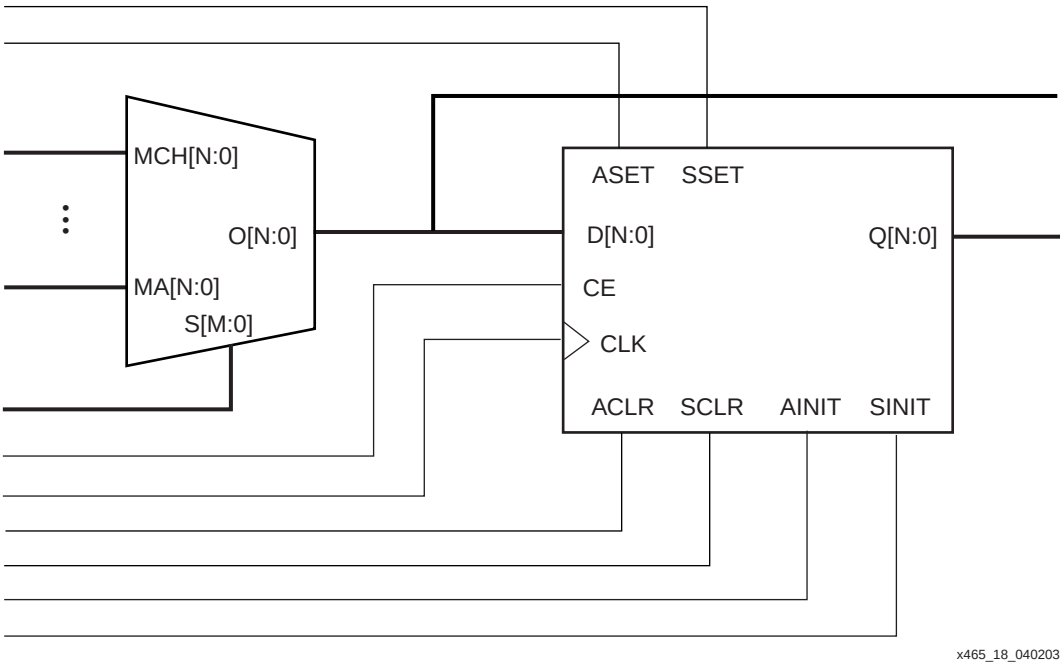


図 18: バス マルチプレクサのコア シンボル

さらに、CORE Generator システムでは、BUFT ベースおよび BUFE ベースのマルチプレクサのファンクションも使用できます。これらのファンクションは、汎用のビットマルチプレクサおよびバス マルチプレクサと同様に、LUT とマルチプレクサの双方またはいずれかで構成できます。

## 改訂履歴

次の表に、この文書の改訂履歴を示します。

| 日付 | バージョン | 改訂内容 |
|---|---|---|
| 04/10/03 | 1.0 | 初版リリース |