# インテル® Quartus® Prime タイミング・アナライザー・クックブック

2017.11.21

MNL-01035 更新情報 フィードバック

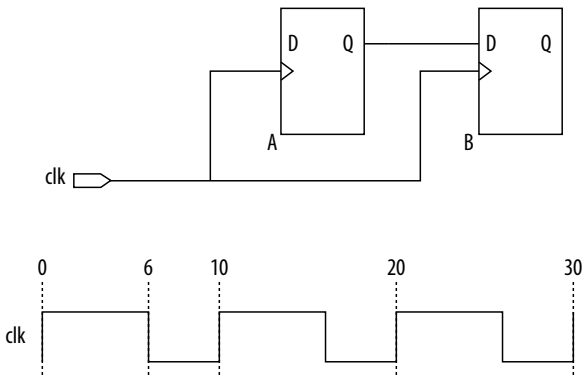
このマニュアルでは、デザイン・シナリオ、制約のガイドライン、および推奨事項より構成されています。このガイドラインを適用するには、タイミング・アナライザーの使用経験および Synopsys* (SDC) の基本的な知識が必要となります。

## クロックと生成クロック

### 50/50 ではない基本的なデューティー・サイクル・クロック

クロックのデューティー・サイクルは、デザインごとに異なる場合があります。デフォルトでは、タイミング・アナライザーで作成されるクロックのデューティ・サイクル は、50/50 です。ただし、`-waveform` オプションを使用すれば、クロックのデューティー・サイクルを変更することができます。

**図 1: 60/40 のデューティー・サイクルを持つ単純なレジスター間のパス**



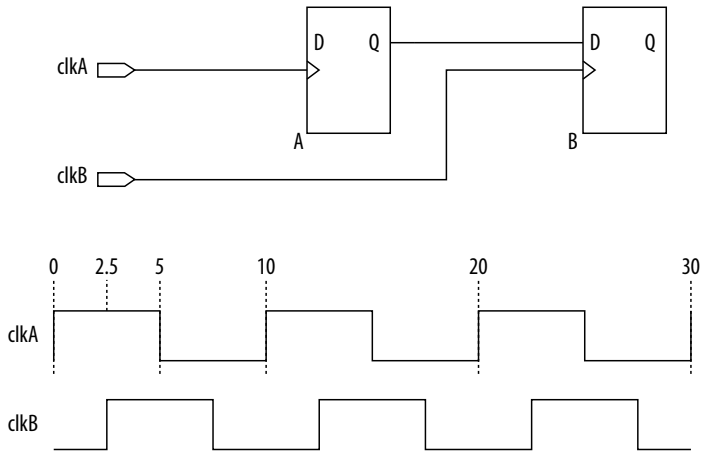

**例 1: 60/40 デューティー・サイクル・クロックの制約**

```
#60/40 duty cycle clock
create_clock \
    -period 10.000 \
    -waveform {0.000 6.000} \
    -name clk6040 [get_ports {clk}]
```

# オフセットクロック

タイミング・アナライザーにクロックを含める場合、クロックの最初の立ち上がりエッジまたは立下りエッジは、デフォルトで絶対値 0 で発生します。 `-waveform` オプションを使用すれば、クロックに対してオフセットを作成することができます。

**図 2: clkB でクロックされる単純なレジスター間のパス**



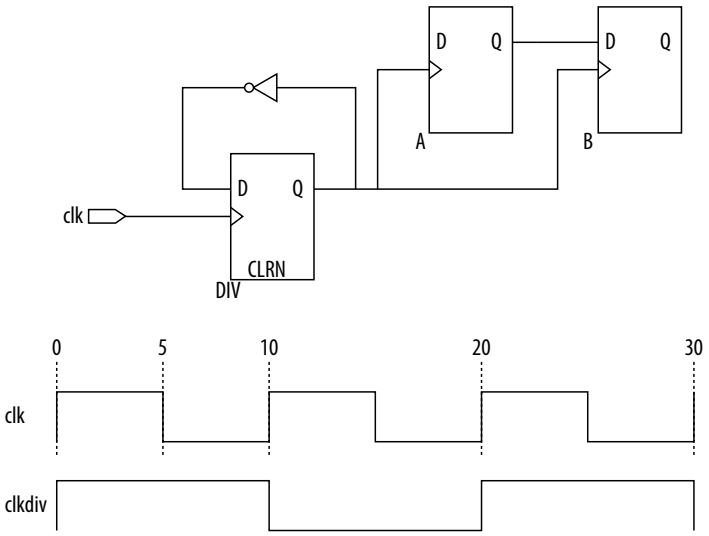

**例 2: オフセットクロックの制約**

```
# -waveform defaults to 50/50 duty cycle
create_clock -period 10.000 \
    -name clkA \
    [get_ports {clkA}]
#create a clock with a 2.5 ns offset
create_clock -period 10.000 \
    -waveform {2.500 7.500} \
    -name clkB [get_ports {clkB}]
```

# -divide_by を使用した基本的なクロック・ディバイダー

派生クロックがソース・クロックより遅い場合、クロック・ソースからデザイン内にク ロックを派生させることができます。クロック・ソースから派生した低速クロックを制約する場 合、`-divide_by` オプションを使用します。

**図 3: 2 分周した派生クロック**

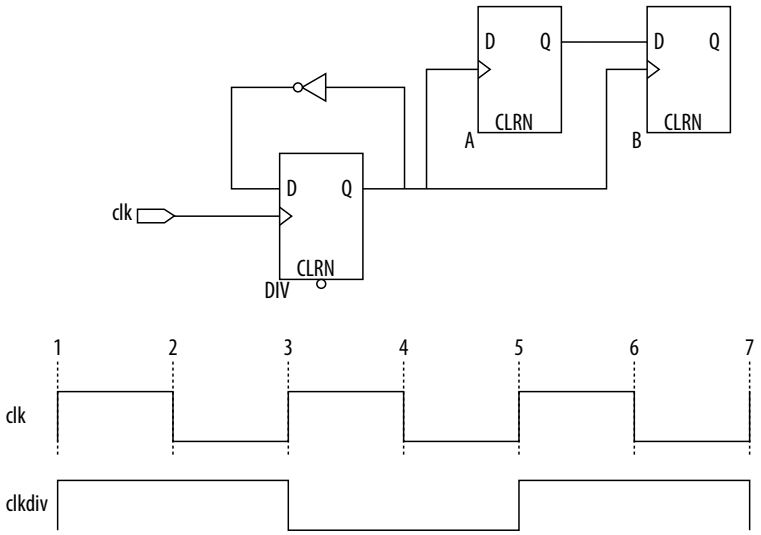



**例 3: -waveform クロック制約の分周**

```
create_clock -period 10.000 -name clk [get_ports {clk}]
# Using -divide_by option
create_generated_clock \
    -divide_by 2 \
    -source [get_ports {clk}] \
    -name clkdiv \
    [get_pins {DIV|q}]
# Alternatively use pins to constrain the divider without
# knowing about the original source
create_generated_clock \
    -divide_by 2 \
    -source [get_pins {DIV|clk}] \
    -name clkdiv \
    [get_pins {DIV|q}]
# the second option works since the
# clock pin of the register DIV is
# connected to the same net fed by the
# clock port clk
```

`-edges` オプションを使用すれば、分周クロックを作成することができます。このオプションを使用することで、クロックの立ち上がり、立下り、および次の立ち上がりエッジを指定することが可能となります。

**図 4: -edges オプションを使用した 2 分周クロック**




**例 4: -waveform クロック制約の分周**

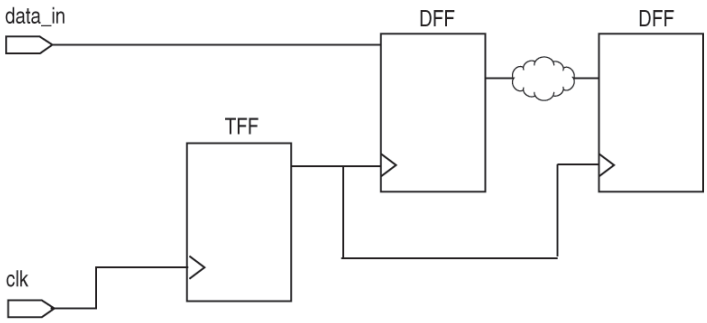
```
# Edge numbers are based on the master clock
create_generated_clock \
    -edges {1 3 5} \
    -source [get_pins {DIV|clk}] \
    -name clkdiv \
    [get_pins {DIV|q}]
```

## トグルレジスターによるクロックの生成

2 分周クロックを作成するには、トグルレジスターを使用します。トグルレジスターを供給するデータが論理「1」の値であり、10 ns 周期のクロックによって供給されている場合、このレジスターの出力は 20 ns の周期を持つクロックとなります。

トグルレジスター・クロックに対する制約は、前の例と非常に似ています。

**図 5: トグルレジスターによる 2 分周クロックの生成**




**例 5: トグルレジスターの制約**

```
# Create a base clock
create_clock \
    -period 10.000 \
    -name clk \
    [get_ports {clk}]
# Create the generated clock on the output
# of the toggle register.
create_generated_clock \
    -name tff_clk \
    -source [get_ports {clk}] \
    -divide_by 2 \
    [get_pins {tff|q}]
```

## PLL クロック

このセクションでは、`derive_pll_clocks`、`create_clock`、および `create_generated_clock` 制約を例として説明します。

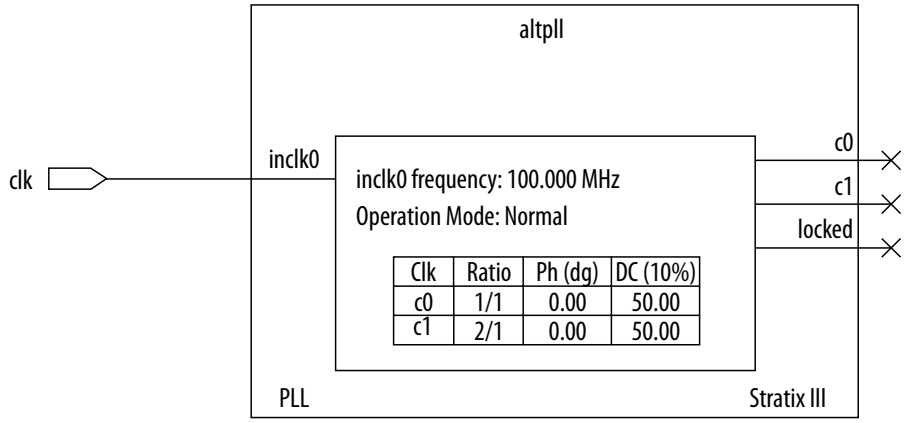
フェーズ・ロック・ループ (PLL) は、 Intel® FPGA でクロックを合成する目的で使用します。適切な解析を実行するためには、すべての出力クロックを制約する必要があります。ALTPLL IP コアは、お客様のデザインの Intel FPGA に PLL を統合します。

PLL を制約するには、以下の方法を使用します。

- ベースクロックと PLL 出力クロックを自動で作成する。
- ベースクロックを手動で生成し、PLL 出力クロックを自動で作成する。
- ベースクロックと PLL 出力クロックを手動で作成する。

このセクションは、それぞれの方法の利点を記載しています。

**図 6: ALTPLL IP コア**




### 方法 1 – ベースクロックと PLL 出力クロックを自動で作成する

この方法では、PLL の入力クロックと出力クロックを自動で作成することが可能です。ALTPLL IP コアで指定されたすべての PLL パラメーターは、PLL の入力クロックと出力クロックを制約します。PLL の入力クロックと出力クロックを作成する際、PLL パラメーターの変更を追跡したり、正しい値を指定したりする必要はありません。

すべての入力と出力を自動的に制限するには、`-create_base_clocks` オプションの `derive_pll_clocks` コマンドを使用します。タイミング・アナライザーは、PLL の IP カタログのインスタンシエーションに基づいて正しい設定を決定します。

### 例 6: PLL ベースクロックを自動で制約する

```
derive_pll_clocks -create_base_clocks
```

### 方法 2 – ベースクロックを手動で生成し、PLL 出力クロックを自動で作成する

この方法では、PLL の入力クロックを手動で制限し、タイミング・アナライザーが自動的に PLL の出力クロックを制限することが可能です。さらに、ALTPLL IP コアで指定された入力クロック周波数の代わりに、別の入力クロック周波数を指定することも可能です。ALTPLL IP コアは、指定されたパラメーターで PLL 出力クロックを自動的に作成します。同じ PLL 出力クロック・パラメーターを維持しながら、異なる入力クロック周波数を試すことも可能です。

**注意:** 指定した入力クロック周波数が、現在設定されている PLL と互換性があることを確認してください。

この方法は、`derive_pll_clocks` コマンドを使用して PLL に対して入力クロックを手動で作成することが可能です。

### 例 7: PLL ベースクロックを手動で制約する

```
create_clock -period 10.000 -name clk [get_ports {clk}]
derive_pll_clocks
```

### 方法 3 – ベースクロックと PLL 出力クロックを手動で作成する

この方法では、PLL の入力クロックと出力クロックを手動で制限することが可能です。すべての PLL パラメーターは指定されています。パラメーターの値は ALTPLL IP コアで指定された値とは別の値を設定することが可能です。加えて、さまざまな PLL 入出力周波数とパラメーターを試すことができます。

この方法は、`create_clock` コマンドと `create_generate_clock` コマンドを組み合わせて使用することができます。

### 例 8: PLL 出力クロックとベースクロックを手動で制約する

```
create_clock -period 10.000 -name clk [get_ports {clk}]
create_generated_clock \
    -name PLL_C0 \
    -source [get_pins {PLL|altpll_component|pll|inclk[0]}] \
    [get_pins {PLL|altpll_component|pll|clk[0]}]
create_generated_clock \
    -name PLL_C1 \
    -multiply_by 2 \
    -source [get_pins {PLL|altpll_component|pll|inclk[0]}] \
    [get_pins {PLL|altpll_component|pll|clk[1]}]
```

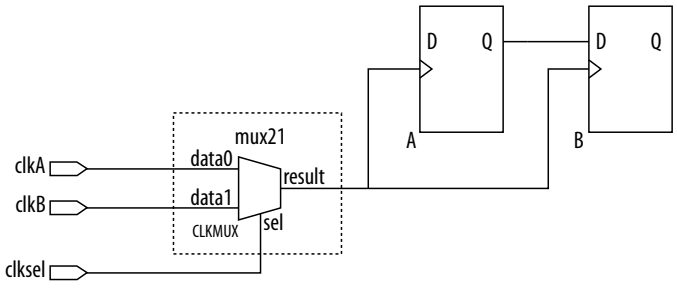
## マルチ周波数解析

一部のデザインには FPGA をドライブするクロックを複数必要とします。この場合、あるクロックが他のクロックよりも速くなったり、遅くなったりすることがあります。

### クロック・マルチプレクサ

クロック・マルチプレクサでは、`create_clock` 制約と `set_clock_groups` 制約を使用することで 2 つ以上のクロックを選択することが可能です。

**図 7: 一般的な 2:1 クロック・マルチプレクサの制約**



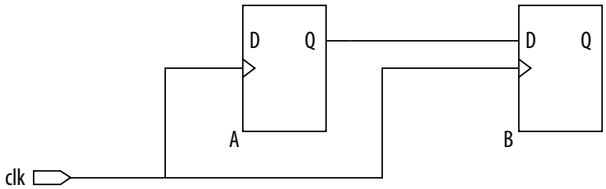
### 例 9: クロック・マルチプレクサの制約

```
#Create the first input clock clkA to the mux
create_clock -period 10.000 -name clkA [get_ports {clkA}]
#Create the second input clock clkB to the mux
create_clock -period 20.000 -name clkB [get_ports {clkB}]
#Cut paths between clkA and clkB
set_clock_groups     -exclusive     -group {clkA}     -group {clkB}
```

## 外部からクロックを切り替える

外部マルチプレクサやジャンパーの設定により、デジタルシステムは同一のクロックポートに対して異なるクロック周波数を提供することが可能です。タイミング・アナライザーは `create_clock` 制約と `-add` オプションを使用して、このビヘイビアをモデル化することが可能です。次の図は、クロックポートのクロックを 100-MHz クロックあるいは 50-MHz クロックでドライブ可能な単純なレジスター間パスを示しています。

**図 8: 単純なレジスター間パス**



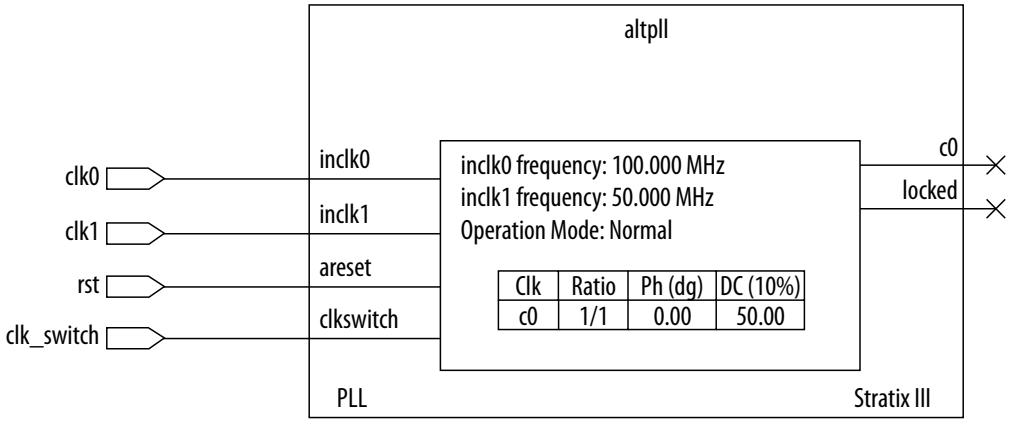

### 例 10: 外部からクロックを切り替える制約

```
# The clk port can be driven at 100MHz (10ns) or
# 50MHz (20ns)
# clkA is 10ns
create_clock \
    -period 10.000 \
    -name clkA \
    [get_ports {clk}]
# clkB is 20ns assigned to the same port
# Requires -add option
create_clock \
    -period 20.000 \
    -name clkB \
    [get_ports {clk}] \
    -add
set_clock_groups \
    -exclusive \
    -group {clkA} \
    -group {clkB}
```

## PLL クロック・スイッチオーバー

PLL は、Intel FPGA の PLL クロック・スイッチオーバー機能を使用することで 2 つの入力クロックからクロックを選択することが可能です。

**図 9: PLL クロック・スイッチオーバー**




**例 11: PLL クロック・スイッチオーバーの制約**

```
#create a 10ns clock for clock port clk0
create_clock \
    -period 10.000 \
    -name clk0 \
    [get_ports {clk0}]
#create a 20ns clock for clock port clk1
create_clock \
    -period 20.000 \
    -name clk1 \
    [get_ports {clk1}]
#automatically create clocks for the PLL output clocks
#derive_pll_clocks automatically makes the proper
#clock assignments for clock-switchover
derive_pll_clocks
set_clock_groups \
    -exclusive \
    -group {clk0} \
    -group {clk1}
```

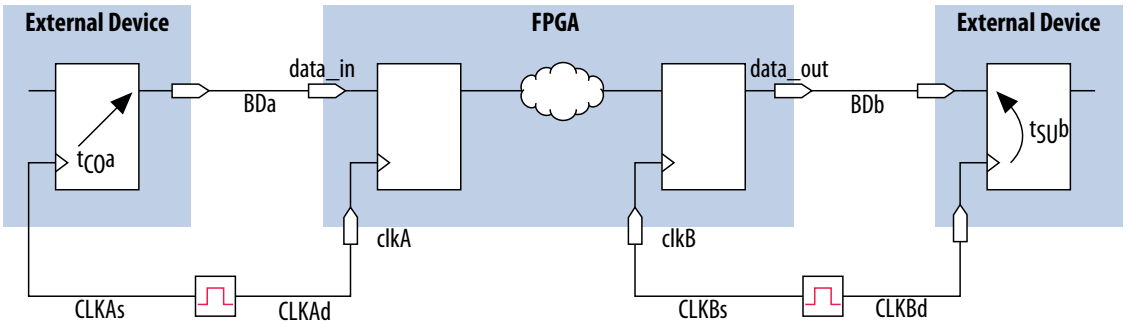
# I/O 制約

## 仮想クロックを使用した入力遅延と出力遅延

すべての入力遅延と出力遅延は、仮想クロックを参照する必要があります。 仮想クロックを使用することで、タイミング・アナライザーは `derive_clock_uncertainty` コマンドを使用して、適切なクロック不確実性の値を導出し、適用することができます。入力と出力の遅延が仮想クロックではなく、ベースクロックまたは PLL クロックを参照する場合、`derive_clock_uncertainty` によって決定されるクロック内およびクロック間転送クロックの不確実性が I/O ポートに誤って適用されます。また、仮想クロックを使用すると、`derive_clock_uncertainty` によって決定されるクロック不確実性の独立性とは無関係に追加の外部のクロック不確実性を適用することが可能となります。

仮想クロックのプロパティーは、入力 (入力遅延) または出力 (出力遅延) ポートのいずれかに使用した元のクロックと同じものでなければいけません。

**図 10: 仮想クロックを入力/出力ポートとして使用したチップ間のデザイン**




### 例 12: 仮想クロックを参照する入力遅延と出力遅延

```
#create the input clock
create_clock -name clkA -period 10 [get_ports clkA]
#create the associated virtual input clock
create_clock -name clkA_virt -period 10
#create the output clock
create_clock -name clkB -period 5 [get_ports clkB]
#create the associated virtual input clock
create_clock -name clkB_virt -period 5
#determine internal clock uncertainties
derive_clock_uncertainty
#create the input delay referencing the virtual clock
#specify the maximum external clock delay from the external
#device
set CLKAs_max 0.200
#specify the minimum external clock delay from the external
#device
set CLKAs_min 0.100
#specify the maximum external clock delay to the FPGA
set CLKAd_max 0.200
#specify the minimum external clock delay to the FPGA
set CLKAd_min 0.100
#specify the maximum clock-to-out of the external device
set tCOa_max 0.525
#specify the minimum clock-to-out of the external device
set tCOa_min 0.415
#specify the maximum board delay
set BDa_max 0.180
#specify the minimum board delay
set BDa_min 0.120
#create the input maximum delay for the data input to the
#FPGA that accounts for all delays specified
set_input_delay -clock clkA_virt \
-max [expr $CLKAs_max + $tCOa_max + $BDa_max - $CLKAd_min] \
[get_ports {data_in[*]}]
#create the input minimum delay for the data input to the
#FPGA that accounts for all delays specified
set_input_delay -clock clkA_virt \
-min [expr $CLKAs_min + $tCOa_min + $BDa_min - $CLKAd_max] \
[get_ports {data_in[*]}]
#creating the output delay referencing the virtual clock
#specify the maximum external clock delay to the FPGA
```

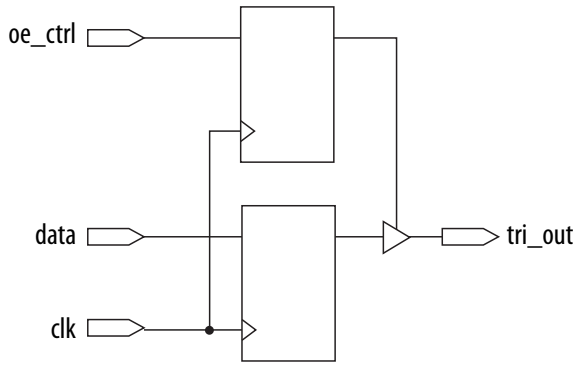
```
set CLKBs_max 0.100
#specify the minimum external clock delay to the FPGA
set CLKBs_min 0.050
#specify the maximum external clock delay to the external device
set CLKBd_max 0.100
#specify the minimum external clock delay to the external device
set CLKBd_min 0.050
#specify the maximum setup time of the external device
set tSUb 0.500
#specify the hold time of the external device
set tHb 0.400
#specify the maximum board delay
set BDb_max 0.100
#specify the minimum board delay
set BDb_min 0.080
#create the output maximum delay for the data output from the
#FPGA that accounts for all delays specified
set_output_delay -clock clkB_virt \
-max [expr $CLKBs_max + $tSUb + $BDb_max - $CLKBd_min] \
[get_ports {data_out}]
#create the output minimum delay for the data output from the
#FPGA that accounts for all delays specified
set_output_delay -clock clkB_virt \
-min [expr $CLKBs_min - $tHb + $BDb_min - $CLKBd_max] \
[get_ports {data_out}]
```

## トライステート出力

トライステート出力は、有効なデータ信号またはハイインピーダンス信号のいずれかを出力ポートまたは I/O ポートからドライブすることを可能とします。どちらの信号のタイミングも、デザインの全体的なシステムのタイミングにおいて 重要となります。

トライステート出力のタイミング制約は、通常の出力ポートと同じです。

**図 11: トライステート・バッファーによって供給される一般的な出力**




### 例 13: トライステート出力ポートの制約

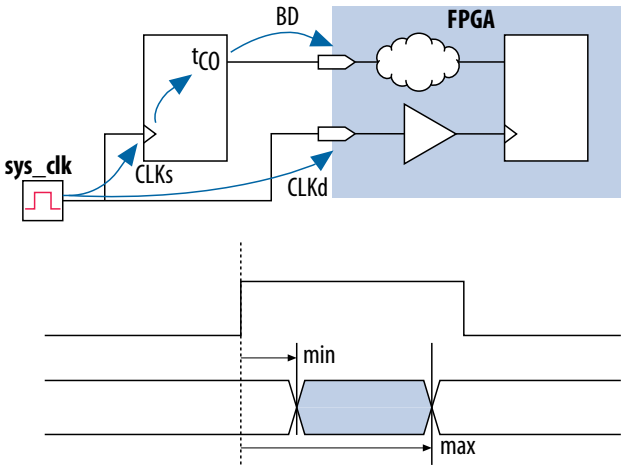
```
# Base clock
create_clock [get_ports {clk}] \
        -name {clk} \
        -period 10.0 \
        -waveform {0.0 5.0}
# Virtual clock for the output port
```

```
create_clock  \
        -name {clk_virt} \
        -period 10.0 \
        -waveform {0.0 5.0}
# Output constraints
set_output_delay 2.0 \
        -max \
        -clock [get_clocks {clk_virt}] \
        [get_ports {tri_out}]
set_output_delay 1.0 \
        -min \
        -clock [get_clocks {clk_virt}] \
        [get_ports {tri_out}]
```

## システム同期入力

次の図は、一般的なチップ間入力インターフェイスと、このインターフェイスに向けた入力遅延の指定に必要なさまざまなパラメーターを示しています。

**図 12: 単純なチップ間入力インターフェイス**




### 例 14: システム同期入力の制約

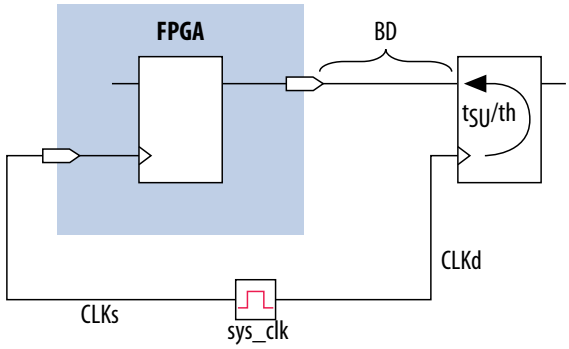
```
#specify the maximum external clock delay from the external device
set CLKs_max 0.200
#specify the minimum external clock delay from the external device
set CLKs_min 0.100
#specify the maximum external clock delay to the FPGA
set CLKd_max 0.200
#specify the minimum external clock delay to the FPGA
set CLKd_min 0.100
#specify the maximum clock-to-out of the external device
set tCO_max 0.525
#specify the minimum clock-to-out of the external device
set tCO_min 0.415
#specify the maximum board delay
set BD_max 0.180
#specify the minimum board delay
set BD_min 0.120
#create a clock 10ns
```

```
create_clock -period 10 -name sys_clk [get_ports sys_clk]
#create the associated virtual input clock
create_clock -period 10 -name virt_sys_clk
#create the input maximum delay for the data input to the FPGA that
#accounts for all delays specified
set_input_delay -clock virt_sys_clk \
    -max [expr $CLKs_max + $tCO_max + $BD_max - $CLKd_min] \
    [get_ports {data_in[*]}]
#create the input minimum delay for the data input to the FPGA that
#accounts for all delays specified
set_input_delay -clock virt_sys_clk \
    -min [expr $CLKs_min + $tCO_min + $BD_min - $CLKd_max] \
    [get_ports {data_in[*]}]
```

## システム同期出力

次の図は、一般的なチップ間出力インターフェイスと、このインターフェイスに向けた出力遅延の指定に必要なさまざまなパラメーターを示しています。

**図 13: 単純なチップ間出力インタフェース**




### 例 15: システム同期出力の制約

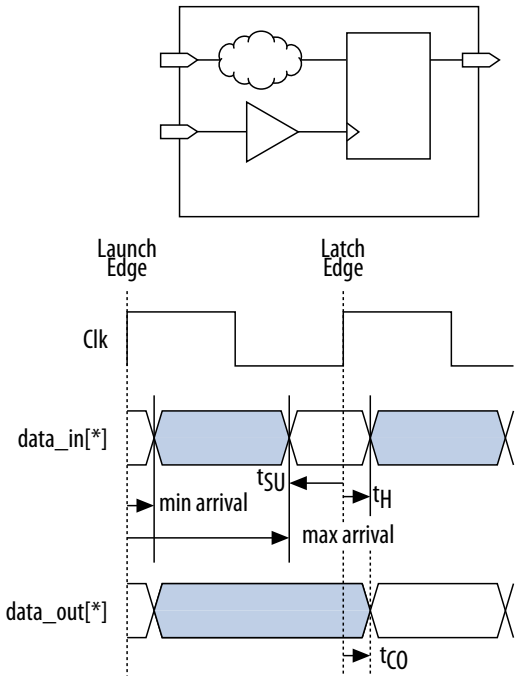
```
#specify the maximum external clock delay to the FPGA
set CLKs_max 0.200
#specify the minimum external clock delay to the FPGA
set CLKs_min 0.100
#specify the maximum external clock delay to the external device
set CLKd_max 0.200
#specify the minimum external clock delay to the external device
set CLKd_min 0.100
#specify the maximum setup time of the external device
set tSU 0.125
#specify the minimum setup time of the external device
set tH 0.100
#specify the maximum board delay
set BD_max 0.180
#specify the minimum board delay
set BD_min 0.120
#create a clock 10ns
create_clock -period 10 -name sys_clk [get_ports sys_clk]
#create the associated virtual input clock
create_clock -period 10 -name virt_sys_clk
#create the output maximum delay for the data output from the FPGA that
#accounts for all delays specified
```

```
set_output_delay -clock virt_sys_clk \
    -max [expr $CLKs_max + $BD_max + $tSU - $CLKd_min] \
    [get_ports {data_out[*]}]
#create the output minimum delay for the data output from the FPGA that
#accounts for all delays specified
set_output_delay -clock virt_sys_clk \
    -min [expr $CLKs_min + $BD_min - $tH - $CLKd_max] \
    [get_ports {data_out[*]}]
```

## I/O タイミング要件 ($t_{SU}$、$t_H$、および $t_{CO}$)

次の例は、`set_input_delay` を使用した $t_{SU}$ と $t_H$ の指定方法と `set_output_delay` を使用した $t_{CO}$ の指定方法を示しています。

**図 14: I/O タイミングの仕様**




### 例 16: $t_{SU}$、$t_H$、および $t_{CO}$ の制約

```
#Specify the clock period
set period 10.000
#Specify the required tSU
set tSU 1.250
#Specify the required tH
set tH 0.750
#Specify the required tCO
set tCO 0.4
#create a clock 10ns
create_clock -period $period -name clk [get_ports sys_clk]
#create the associated virtual input clock
create_clock -period $period -name virt_clk
set_input_delay -clock virt_clk \
    -max [expr $period - $tSU] \
```

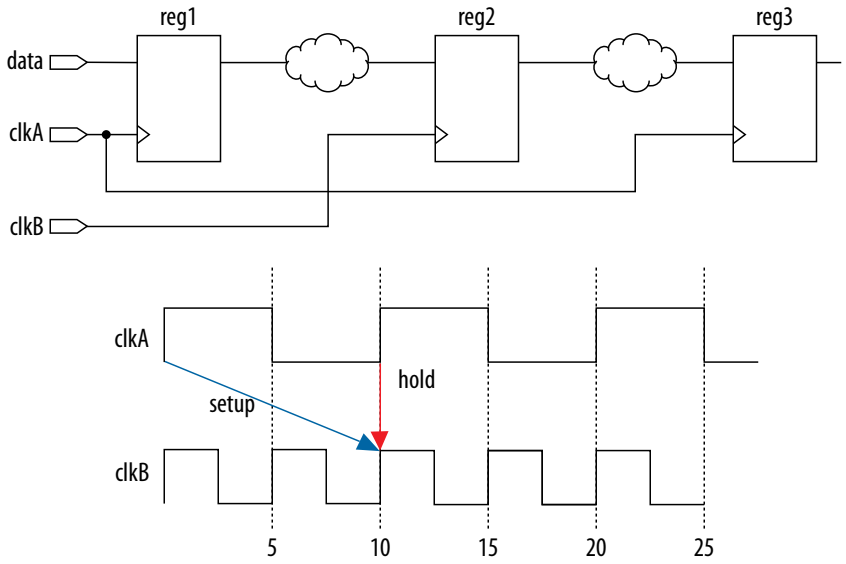
```
    [get_ports {data_in[*]}]
set_input_delay -clock virt_clk \
    -min $tH \
    [get_ports {data_in[*]}]
set_output_delay -clock virt_clk \
    -max [expr $period - $tCO] \
    [get_ports {data_out[*]}]
set_output_delay -clock virt_clk \
    -min [expr -1*$tco_min] \
    [get_ports {data_out [*]}]
```

# 例外

## マルチサイクル例外

デフォルトでは、タイミング・アナライザーはシングルサイクル解析を使用して、レジスター間パスのセットアップとホールドの両方の関係を決定します。これにより、設定および保留の要件は最も制限されたものとなりますが、マルチサイクル例外を使用することで、レジスター間パスのセットアップまたはホールド関係を緩和することができます。

**図 15: レジスター間パス**




マルチサイクルは、クロック間転送あるいは個別のレジスターに適用可能です。クロック間転送にマルチサイクルを適用すると、ソースクロックとデスティネーション・クロックによって供給されるレジスター間パスのターゲットクロックで指定するすべての設定またはホールド関係が影響を受けます。

**例 17: クロック間のマルチサイクル化**

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]
```

```
set_multicycle_path -from [get_clocks {clkA}] -to [get_clocks {clkB}] -setup
-end 2
```

クロック間のマルチサイクル化の例では、ソースクロックが clkA、そしてデスティネーション・クロックが clkB である任意のレジスター間パスに対して、デスティネーション・クロックが追加されることでセットアップ関係が緩和されています。これにより、レジスター reg1 とレジスター reg2 のセットアップ関係がデフォルトの 5 ns ではなく 12.5 ns となります。レジスター reg2 とレジスター reg3 間のセットアップ関係は、このマルチサイクルによる影響は受けません。

個々のレジスターにマルチサイクルを適用する場合、指定されたレジスターのセットアップまたはホールド関係にのみ影響します。

上記のクロック間のマルチサイクル化の例では、レジスター reg1 からレジスター reg2 までのレジスター間パスに対しデスティネーション・クロック周期を追加することで、セットアップ関係が緩和されています。これにより、レジスター reg1 とレジスター reg2 間のセットアップ関係は、デフォルトの 5 ns ではなく 10 ns となります。レジスター reg2 とレジスター reg3 間のセットアップ関係は、このマルチサイクルによる影響は受けません。

### 例 18: レジスター間のマルチサイクル化

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]
set_multicycle_path -from [get_pins {reg1|q}] -to [get_pins {reg2|d}] -setup
-end 2
```

**関連情報**

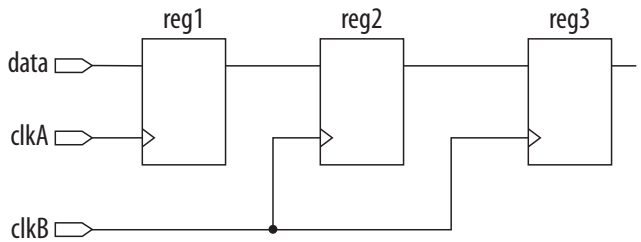
**インテル® Quartus® Prime タイミング・アナライザー**

タイミング・アナライザーで使用可能なマルチサイクル例外の種類の詳細については、こちらを参照してください。

## フォルスパス

すべてのパスでタイミングを分析する必要はありません。クリティカルではないパスの同期は、タイミング解析から削除または切断できます。クリティカルではないパスを宣言すると、インテル® Quartus® Prime フィッターはクリティカルパスの最適化に集中し、全体的なコンパイル時間を短縮することが可能となります。

### 図 16: レジスター間の切断パス



フォルスパスは、クロック間転送あるいは個別のレジスターに適用可能です。クロック間転送にフォルスパスを適用すると、ターゲットとするクロック間のすべてのパスが切断されます。

フォルスパスのクロック間の例では、パスは切断されており、ソースクロックが clkA でデスティネーション・クロックが clkB であるレジスター間パスに対しては、タイミング・アナライザーは解析を行いません。これは、ソースレジスターが clkB でクロックされ、デスティネーション・レジスターが clkA でクロックされるレジスター間のパスには影響しません。

**注意:** set_false_path コマンドは、clkA から clkB へのパスを切断します。このコマンドは、clkB から clkA までのパスは切断しません。clkB から clkA までのパスを切断するには、set_false_path -from clkB -to clkA といった別の set_false_path コマンドを適用する必要があります。別の方法としては、set_clock_groups を使用して clkA から clkB までのパスと clkB から clkA までのパスを一つのコマンドで切断することができます。

フォルスパスを個別のレジスターに適用すると、指定したパスだけが切断されます。

**例 19: フォルスパスクロック間**

```
create_clock -period 12 [get_ports clkA]
create_clock -period 5 [get_ports clkB]
set_false_path -from [get_clocks {clkA}] -to [get_clocks {clkB}]
```

フォールスパス・レジスター間の例では、レジスター reg1 からレジスター reg2 までのレジスター間パスが切断されています。これ以外のすべてのパスには影響はありません。

**例 20: フォルスパスレジスター間**

```
create_clock -period 10 [get_ports clkA]
create_clock -period 5 [get_ports clkB]
set_false_path -from [get_pins {reg1|q}] -to [get_pins {reg2|d}]
```

**関連情報**

**Set Clock Groups Dialog Box (set_clock_groups)**

インテル Quartus Prime Help の set_clock_groups コマンドについての詳細を提供しています。

# その他

## JTAG 信号

多くのインシステム・デバッグツールは、Intel FPGA の JTAG インターフェイスを使用します。

JTAG インターフェイスを使用してデザインをデバッグする場合、JTAG 信号である TCK、TMS、TDI、および TDO がデザインの一部として実装されます。このため、タイミング・アナライザーは制約されていないパスのレポートが生成される際、これらの信号を制約されていないものとしてフラグします。

**表 1: 制約されていないと表示される可能性のある JTAG 信号**

| 信号名 | 説明 |
|---|---|
| altera_reserved_tck [1] [2] | JTAG テストクロック入力ポート |
| altera_reserved_tms [2] | JTAG テストモード選択入力ポート |
| altera_reserved_tdi [2] | JTAG テストデータ入力ライン入力ポート |
| altera_reserved_tdo [2] | JTAG テストデータ出力ライン出力ポート |

次の SDC コマンドを適用することで、JTAG 信号を抑制することができます。

**例 21: JTAG 信号の制約**

```
# Search "---customize here---" for the few decisions you need to make
#
# By default, the most challenging timing spec is applied to work in
# many JTAG chain setup situations

set_time_format -unit ns -decimal_places 3

# This is the main entry point called at the end of this SDC file.
proc set_jtag_timing_constraints { } {
    # If the timing characteristic outside of FPGA is well understood, and
    # there is a need to provide more slack to allow flexible placement of
    # JTAG logic in the FPGA core, use the timing constraints for both
    # timing analysis and fitter; otherwise, use the default fitter timing
    # constraints.

    # ---customize here---
    set use_fitter_specific_constraint 1

    if { $use_fitter_specific_constraint && [string equal quartus_fit
$::TimeQuestInfo(nameofexecutable)] } {
        # Define a different set of timing spec to influence place-and-route
        # result in the jtag clock domain. The slacks outside of FPGA are
        # maximized.

        set_default_quartus_fit_timing_directive
    }  else {
        # Define a set of timing constraints that describe the JTAG paths
        # for the Timing Analyzer to analyze. The Timing Analyzer timing
reports show whether
        # the JTAG logic in the FPGA core will operates in this setup.

        set_jtag_timing_spec_for_timing_analysis
    }
}

proc set_default_quartus_fit_timing_directive { } {
    # A10 supports max 33.3Mhz clock
    set jtag_33Mhz_t_period 30

    create_clock -name {altera_reserved_tck} -period $jtag_33Mhz_t_period
[get_ports {altera_reserved_tck}]
```

[1] インテル Quartus Prime スタンダード・エディション によってデフォルトで制約されています。
[2] JTAG 信号制約のテンプレートに含まれる SDC 例のバリエーションを使用しない限り、インテル Quartus Prime 開発ソフトウェア・プロ・エディション で制約されていません。

```
    set_clock_groups -asynchronous -group {altera_reserved_tck}
    # Force fitter to place register driving TDO pin to be as close to
    # the JTAG controller as possible to maximize the slack outside of FPGA.
    set_max_delay -to [get_ports { altera_reserved_tdo } ] 0
}

proc set_jtag_timing_spec_for_timing_analysis { } {
derive_clock_uncertainty

    # There are few possible JTAG chain configurations:
    # a. This device is the only device in the JTAG chain
    # b. This device is the first one in the JTAG chain
    # c. This device is in the middle of the JTAG chain
    # d. This device is the last one in the JTAG chain

    # No matter where the device is in the chain. The tck and tms are driven
    # directly from JTAG hardware.
    set_tck_timing_spec
    set_tms_timing_spec

    # Depending on where the device is located along the chain, tdi can be
    # either driven by blaster hw (a. b.) or driven by another device in the
    # chain(c. d.)
    # ---customize here---
    set tdi_is_driven_by_blaster 1

    if { $tdi_is_driven_by_blaster } {
    set_tdi_timing_spec_when_driven_by_blaster
    } else {
    set_tdi_timing_spec_when_driven_by_device
    }

    # Depending on where the device is located along the chain, tdo can
    # drive either blaster hw (a. d.) or another device in the chain (b. c.)
    # ---customize here---
    set tdo_drive_blaster 1

    if { $tdo_drive_blaster } {
        set_tdo_timing_spec_when_drive_blaster
    } else {
        set_tdo_timing_spec_when_drive_device
    }

    set_optional_ntrst_timing_spec

    # Cut a few timing paths that are not related to JTAG logic in
    # the FPGA core, such as security mode.
    set_false_path -from [get_ports {altera_reserved_tdi}] -to [get_ports
{altera_reserved_tdo}]
    if { [get_collection_size [get_registers -nowarn *~jtag_reg]] > 0 } {
        set_false_path -from [get_registers *~jtag_reg] -to [get_ports
{altera_reserved_tdo}]
    }

}

proc set_tck_timing_spec { } {
    # USB Blaster 1 uses 6 MHz clock = 166.666 ns period
    set ub1_t_period 166.666
    # USB Blaster 2 uses 24 MHz clock = 41.666 ns period
    set ub2_default_t_period 41.666
    # USB Blaster 2 running at 16 MHz clock safe mode = 62.5 ns period
    set ub2_safe_t_period 62.5

    # ---customize here---
    set tck_t_period $ub2_default_t_period
```

```
    create_clock -name {altera_reserved_tck} -period $tck_t_period
[get_ports {altera_reserved_tck}]
    set_clock_groups -asynchronous -group {altera_reserved_tck}
}

proc get_tck_delay_max { } {
    set tck_blaster_tco_max 14.603
    set tck_cable_max 11.627

    # tck delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tck_header_trace_max 0.5

    return [expr $tck_blaster_tco_max + $tck_cable_max +
$tck_header_trace_max]
}

proc get_tck_delay_min { } {
    set tck_blaster_tco_min 14.603
    set tck_cable_min 10.00

    # tck delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tck_header_trace_min 0.1

    return [expr $tck_blaster_tco_min + $tck_cable_min +
$tck_header_trace_min]
}

proc set_tms_timing_spec { } {
    set tms_blaster_tco_max 9.468
    set tms_blaster_tco_min 9.468

    set tms_cable_max 11.627
    set tms_cable_min 10.0

    # tms delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tms_header_trace_max 0.5
    set tms_header_trace_min 0.1

    set tms_in_max [expr $tms_cable_max + $tms_header_trace_max +
$tms_blaster_tco_max - [get_tck_delay_min]]
    set tms_in_min [expr $tms_cable_min + $tms_header_trace_min +
$tms_blaster_tco_min - [get_tck_delay_max]]

    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -max
$tms_in_max [get_ports {altera_reserved_tms}]
    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -min
$tms_in_min [get_ports {altera_reserved_tms}]
}

proc set_tdi_timing_spec_when_driven_by_blaster { } {
    set tdi_blaster_tco_max 8.551
    set tdi_blaster_tco_min 8.551

    set tdi_cable_max 11.627
    set tdi_cable_min 10.0

    # tms delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
```

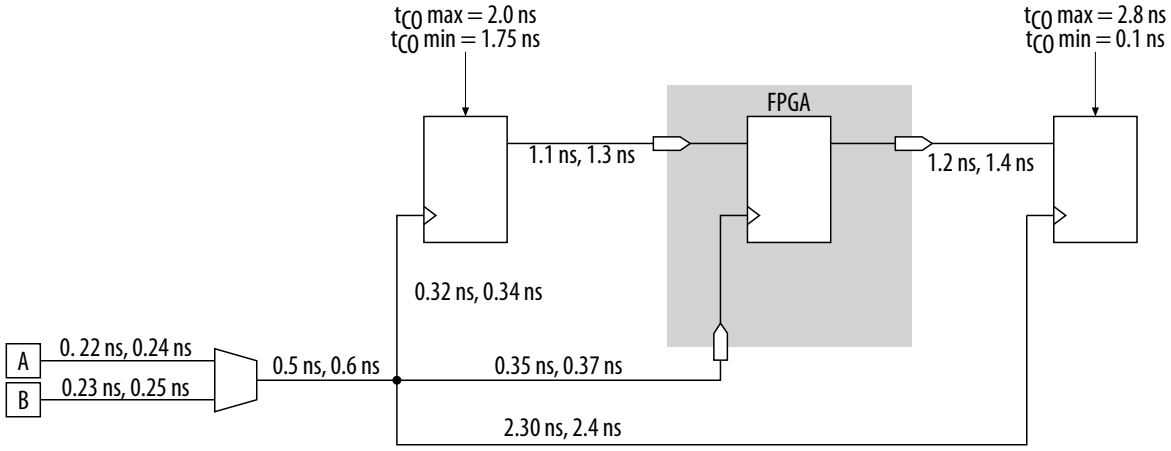
```
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tdi_header_trace_max 0.5
    set tdi_header_trace_min 0.1

    set tdi_in_max [expr $tdi_cable_max + $tdi_header_trace_max +
$tdi_blaster_tco_max - [get_tck_delay_min]]
    set tdi_in_min [expr $tdi_cable_min + $tdi_header_trace_min +
$tdi_blaster_tco_min - [get_tck_delay_max]]

    #TDI launches at the falling edge of TCK per standard
    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -max
$tdi_in_max [get_ports {altera_reserved_tdi}]
    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -min
$tdi_in_min [get_ports {altera_reserved_tdi}]
}

proc set_tdi_timing_spec_when_driven_by_device { } {
    # TCO timing spec of tdo on the device driving this tdi input
    # ---customize here---
    set previous_device_tdo_tco_max 10.0
    set previous_device_tdo_tco_min 10.0

    # tdi delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tdi_trace_max 0.5
    set tdi_trace_min 0.1

    set tdi_in_max [expr $previous_device_tdo_tco_max + $tdi_trace_max -
[get_tck_delay_min]]
    set tdi_in_min [expr $previous_device_tdo_tco_min + $tdi_trace_min -
[get_tck_delay_max]]

    #TDI launches at the falling edge of TCK per standard
    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -max
$tdi_in_max [get_ports {altera_reserved_tdi}]
    set_input_delay -add_delay -clock_fall -clock altera_reserved_tck -min
$tdi_in_min [get_ports {altera_reserved_tdi}]
}

proc set_tdo_timing_spec_when_drive_blaster { } {
    set tdo_blaster_tsu 5.831
    set tdo_blaster_th -1.651

    set tdo_cable_max 11.627
    set tdo_cable_min 10.0

    # tdi delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tdo_header_trace_max 0.5
    set tdo_header_trace_min 0.1

    set tdo_out_max [expr $tdo_cable_max + $tdo_header_trace_max +
$tdo_blaster_tsu + [get_tck_delay_max]]
    set tdo_out_min [expr $tdo_cable_min + $tdo_header_trace_min -
$tdo_blaster_th + [get_tck_delay_min]]

    #TDO does not latch inside the USB Blaster II at the rising edge of TCK,
    # it actually is latched one half cycle later in packed mode
    # (equivalent to 1 JTAG fall-to-fall cycles)
    set_output_delay -add_delay -clock_fall -clock altera_reserved_tck -max
$tdo_out_max [get_ports {altera_reserved_tdo}]
    set_output_delay -add_delay -clock_fall -clock altera_reserved_tck -min
```

```
$tdo_out_min [get_ports {altera_reserved_tdo}]
}

proc set_tdo_timing_spec_when_drive_device { } {
    # TCO timing spec of tdi on the device driven by this tdo output
    # ---customize here---
    set next_device_tdi_tco_max 10.0
    set next_device_tdi_tco_min 10.0

    # tdi delay on the PCB depends on the trace length from JTAG 10-pin
    # header to FPGA on board. In general on the PCB, the signal travels
    # at the speed of ~160 ps/inch (1000 mils = 1 inch).
    # ---customize here---
    set tdo_trace_max 0.5
    set tdo_trace_min 0.1

    set tdo_out_max [expr $next_device_tdi_tco_max + $tdo_trace_max +
[get_tck_delay_max]]
    set tdo_out_min [expr $next_device_tdi_tco_min + $tdo_trace_min +
[get_tck_delay_min]]

    #TDO latches at the rising edge of TCK per standard
    set_output_delay -add_delay -clock altera_reserved_tck -max $tdo_out_max
[get_ports {altera_reserved_tdo}]
    set_output_delay -add_delay -clock altera_reserved_tck -min $tdo_out_min
[get_ports {altera_reserved_tdo}]
}

proc set_optional_ntrst_timing_spec { } {
    # ntrst is an optional JTAG pin to asynchronously reset the device JTAG
controller.
    # There is no path from this pin to any FPGA core fabric.
    if { [get_collection_size [get_ports -nowarn {altera_reserved_ntrst}]] >
0 } {
        set_false_path -from [get_ports {altera_reserved_ntrst}]
    }
}

set_jtag_timing_constraints
```

## 複数のクロックを使用した入力遅延と出力遅延

これらの制約は、プライマリーとセカンダリーの両方のクロックを提供します。プライマリークロックはメインクロックとして動作し、セカンダリークロックはより低速な冗長クロックとして動作します。

**図 17: プライマリークロックとセカンダリークロックを使用した単純なレジスター間のデザイン例**




### 例 22: 複数のクロックを使用した入力遅延

```
#########################
# Create all the clocks #
#########################
# Create variables for the clock periods.
set PERIOD_CLK_A 10.000
set PERIOD_CLK_B 7.000
# Create the clk_a clock which will represent the clock
# that routes to the FPGA.
create_clock \
    -name {clk_a} \
    -period \
    $PERIOD_CLK_A \
    [get_ports {clk}]
# Create the clk_b clock which will represent the clock
# that routes to the FPGA.
# Note the -add is needed because this is the second clock
# that has the same 'clk' port as a target.
create_clock \
    -name {clk_b} \
    -period $PERIOD_CLK_B \
    [get_ports {clk}] \
    -add
# Create a virtual clock which will represent the clock
# that routes to the external source device when clk_a is
# selected a the external mux.
create_clock \
    -name virtual_source_clk_a \
    -period $PERIOD_CLK_A
# Create a virtual clock which will represent the clock
# that routes to the external source device when clk_b is
# selected a the external mux.
create_clock \
    -name virtual_source_clk_b \
    -period $PERIOD_CLK_B
# Create a virtual clock which will represent the clock
# that routes to the external destination device when clk_a
# is selected a the external mux.
create_clock \
    -name virtual_dest_clk_a \
    -period $PERIOD_CLK_A
```

```
# Create a virtual clock which will represent the clock
# that routes to the external destination device when clk_b
# is selected a the external mux.
create_clock \
    -name virtual_dest_clk_b \
    -period $PERIOD_CLK_B
##########################################
# Cut clock transfers that are not valid #
##########################################
# Cut this because virtual_source_clk_b can not be clocking
# the external source device at the same time that clk_a is
# clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_a} \
    -group {virtual_source_clk_b}
# Cut this because virtual_source_clk_a can not be clocking
# the external source device at the same time that clk_b is
# clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_b} \
    -group {virtual_source_clk_a}
# Cut this because virtual_dest_clk_b can not be clocking
# the external destination device at the same time that
# clk_a is clocking the FPGA.
set_clock_groups -exclusive \
    -group {clk_a} \
    -group {virtual_dest_clk_b}
# Cut this because virtual_dest_clk_a can not be clocking
# the external destination device at the same time that
# clk_b is clocking the FPGA
set_clock_groups -exclusive \
    -group {clk_b} \
    -group {virtual_dest_clk_a}
########################################
# Define the latency of all the clocks #
########################################
# Since the Timing Analyzer does not know what part of the clock
# latency is common we must simply remove the common part
# from the latency calculation. For example when
# calculating the latency for virtual_source_clk_a we must
# ignore the 220ps,240ps route and the 500ps/600ps mux
# delay if we want to remove the common clock path
# pessimism.
#
# Define fastest and slowest virtual_source_clk_a path to
# the external source device.
set_clock_latency -source \
    -early .320 \
[get_clocks virtual_source_clk_a]
set_clock_latency -source \
    -late .340 \
    [get_clocks virtual_source_clk_a]
# Define fastest and slowest virtual_source_clk_b path to
# the external source device.
set_clock_latency -source \
    -early .320 \
    [get_clocks virtual_source_clk_b]
set_clock_latency -source \
    -late .340 \
    [get_clocks virtual_source_clk_b]
# Define fastest and slowest clk_a path to the FPGA.
set_clock_latency -source \
    -early .350 \
    [get_clocks clk_a]
set_clock_latency -source \
    -late .370 \
    [get_clocks clk_a]
```

```
# Define fastest and slowest clk_b path to the FPGA.
set_clock_latency -source \
    -early .350 \
    [get_clocks clk_b]
set_clock_latency -source \
    -late .370 \
    [get_clocks clk_b]
# Define fastest and slowest virtual_dest_clk_a path to
# the external destination device.
set_clock_latency -source \
    -early 2.3 \
    [get_clocks virtual_dest_clk_a]
set_clock_latency -source \
    -late 2.4 \
    [get_clocks virtual_dest_clk_a]
# Define fastest and slowest virtual_dest_clk_b path to
# the external destination device.
set_clock_latency -source \
    -early 2.3 \
    [get_clocks virtual_dest_clk_b]
set_clock_latency -source \
    -late 2.4 \
    [get_clocks virtual_dest_clk_b]
####################################
# Constrain the input port 'datain' #
####################################
# This Tco is the min/max value of the Tco for the
# external module.
set Tco_max 2.0
set Tco_min 1.75
# Td is the min/max trace delay of datain from the
# external device
set Td_min 1.1
set Td_max 1.3
# Calculate the input delay numbers
set input_max [expr $Td_max + $Tco_max]
set input_min [expr $Td_min + $Tco_min]
# Create the input delay constraints when clk_a is selected
set_input_delay \
    -clock virtual_source_clk_a \
    -max $input_max \
    [get_ports datain]
set_input_delay \
    -clock virtual_source_clk_a \
    -min $input_min \
    [get_ports datain]
# Create the input delay constraints when clk_b is selected
set_input_delay \
    -clock virtual_source_clk_b \
    -max $input_max \
    [get_ports datain] \
    -add_delay
set_input_delay \
    -clock virtual_source_clk_b \
    -min $input_min \
    [get_ports datain] \
    -add_delay
#####################################
# Constrain the output port 'dataout' #
#####################################
# This Tsu/Th is the value of the Tsu/Th for the external
# device.
set Tsu 2.8
set Th 0.1
# This is the min/max trace delay of dataout to the
# external device.
set Td_min 1.2
```

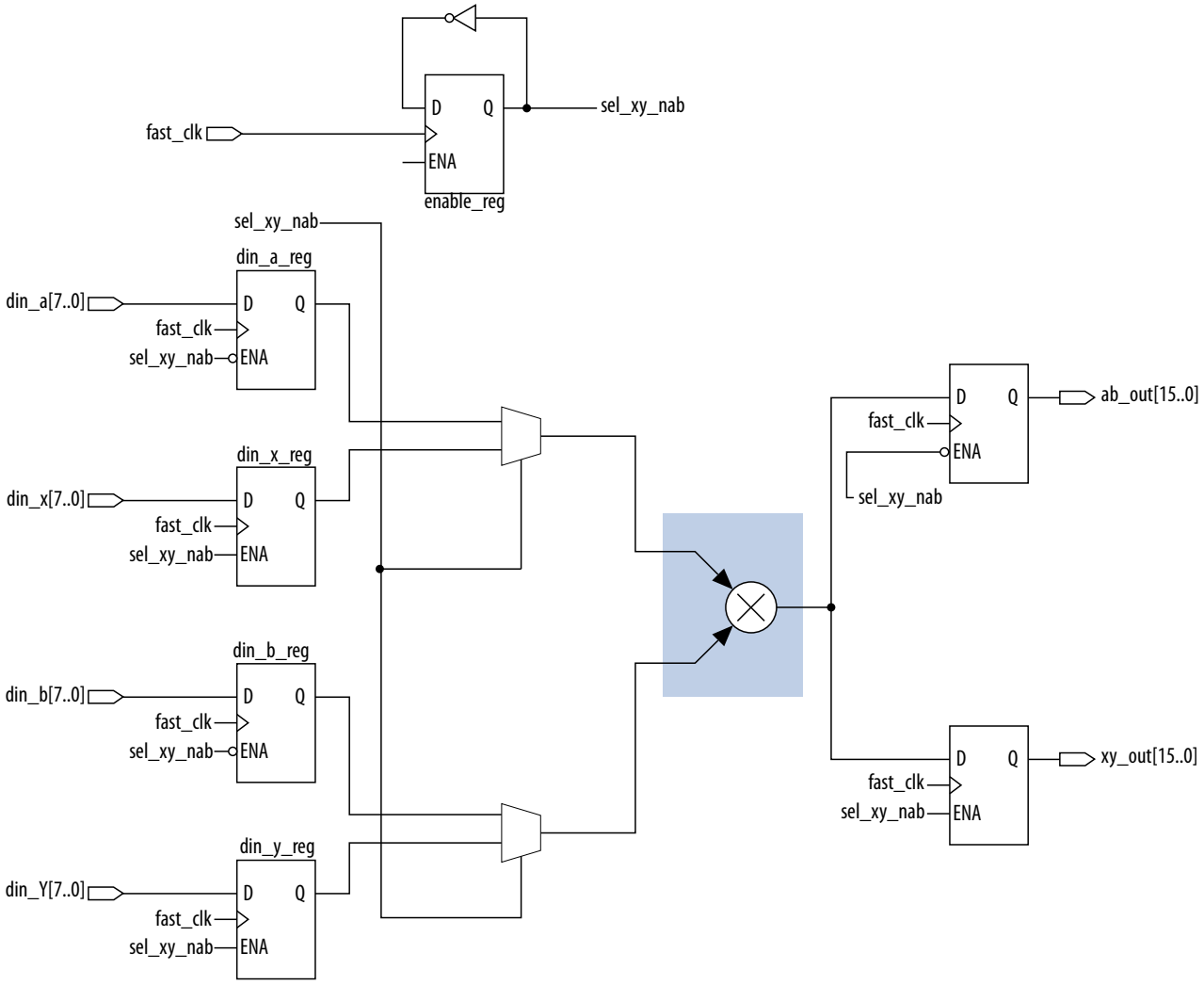
```
set Td_max 1.4
# Calculate the output delay numbers
set output_max [expr $Td_max + $Tsu]
set output_min [expr $Td_min - $Th]
# Create the output delay constraints when clk_a is
# selected.
set_output_delay \
    -clock virtual_dest_clk_a \
    -max $output_max \
    [get_ports dataout]
set_output_delay \
    -clock virtual_dest_clk_a \
    -min $output_min \
    [get_ports dataout]
# Create the output delay constraints when clk_b is
# selected.
set_output_delay \
    -clock virtual_dest_clk_b \
    -max $output_max \
    [get_ports dataout] \
    -add_delay
set_output_delay \
    -clock virtual_dest_clk_b \
    -min $output_min \
    [get_ports dataout] \
    -add_delay
```

## クロック・イネーブル・マルチサイクル

レジスターのイネーブルされたポートに基づいて、クロックがイネーブルされたマルチサイクルを使用してマルチサイクルを指定することができます。たとえば、次の図はレジスターの `din_a_reg[7..0]`、`din_b_reg[7..0]`、`din_x_reg[7..0]`、`din_y_reg[7..0]`、`a_times_b`、および `x_times_y` に対して登録された有効な信号を使用する目的でレジスター `enable_reg` が使用されている単純な回路を表しています。

`enable_reg` レジスターは、レジスターのクロック周期の 2 倍のイネーブルパルスを生成します。したがって、正しい解析を行うためには、マルチサイクル例外を適用する必要があります。`enable_reg` レジスターによって供給されるイネーブル駆動レジスターに対してマルチサイクル・セットアップ 2 とマルチサイクル・ホールド 1 を適用する必要があります。このマルチサイクル例外は、デスティネーション・レジスターが `enable_reg` によって制御されているレジスター間パスにもに適用されます。すべてのイネーブル駆動レジスターに `set_multicycle_path` 例外を適用することで、これを達成することができます。これは、すべてのイネーブル駆動レジスターを指定しなければいけないため、時間のかかる作業となります。また、`set_multicycle_path` と `get_fanouts` の組み合わせを使用することも可能です。

**図 18: クロック・イネーブル・マルチサイクル・ホールド**




**例 23: クロック・イネーブル・マルチサイクル制約**

```
#Setup multicycle of 2 to enabled driven destination registers
set_multicycle_path 2 -to [get_fanouts [get_pins enable_reg|q] \
-through [get_pins -hierarchical *|ena]] -end -setup
```

`set_multicycle_path` 例外のターゲットは、レジスターのイネーブルポートを供給する `enable_reg` レジスターのすべてのファンアウトに制限されています。

```
[get_fanouts [get_pins enable_reg|q] -through [get_pins -hierarchical *|ena]]
```

セットアップとホールドの関係は、`enable_reg` レジスターで開始し、2 と 1 のいずれかのイネーブル駆動レジスターで終了します。

**関連情報**

**インテル Quartus Prime タイミング・アナライザー**

マルチサイクル例外の詳細については、 *インテル Quartus Prime ハンドブック* vol.3 のタイミング・アナライザーの章を参照してください。

# 改訂履歴

**表 2: 改訂履歴**

| 日付 | ソフトウェア・バージョン | 変更内容 |
|---|---|---|
| 2017 年 11 月 | 17.1.1 | デバイス JTAG コントローラーを非同期にリセットするオプションを含めるため、JTAG 信号制約のサンプルコードを更新しました。 |
| 2017 年 11 月 | 17.1.0 | Intel 表記に変更しました。 |
| 2016 年 10 月 | 16.1.0 | カスタマーフィードバックを反映し、マルチサイクル例外のトピックを更新しました。 |
| 2016 年 2 月 | 16.0.0 | • JTAG 信号の SDC 例を更新しました。<br>• パックされた FF の OE の Unateness に関するセクションを追加しました。<br>• クロック・イネーブル・マルチサイクルのトピックを加筆修正しました。<br>• スクリプトの例と回路図の誤植を修正しました。 |
| 2011 年 1 月 | 11.0.0 | • トグルレジスターによるクロックの生成とトライステート出力のセクションを新しく追加しました。<br>• 編集上の軽微な変更を行いました。 |
| 2010 年 3 月 | 10.0.0 | スクリプト例の誤植を修正しました。 |
| 2008 年 8 月 | 8.1.0 | 初版 |