The Intelligent Technology Company

Technical Note

ELSENA

超安定回路の設計

文書管理番号：ELS0297_S000_10

2006 年 4 月

株式会社エルセナ

超安定回路の設計

目次

1 はじめに

Million Gates CPLD/FPGA 時代を迎えて、益々、高信頼度設計が求められています。従来の PAL デバイスのように、一度入力された信号が、必ず、出力端子に出力されると言った時代は既に終焉を迎えています。あらゆる信号がピン・レベルでモニタできる時代は終わり、暗闇の中（ブラック・ボックス）に如何に整然と確からしいデザインをインプリメントするかと言う時代に突入しています。このようなブラック・ボックス時代に、依然としてデバイスの中を明確に評価する方法は確立されていません。勿論、特別に設けられたテスト端子により回路動作をモニタしたり、新たに ALTERA 社が提唱する SignalTap なる評価方法も今後有効な評価方法になると思われますが、総じて、殆どのデザインは何ら評価の為の方法を講じていないのが実状です。良く聞かれることに、『シミュレーションで動作するが、実機で動作しない』と言うことがあります。これは、単に、信頼度の低い回路がインプリメントされていることに他なりません。そんな中で、旧態依然とした設計手法で設計を行うと、真っ暗闇を手探りで出口を探すようなものであり、このような状態で行われた設計に対するデバッグや評価は、無限の時間を要求し、且つ、信頼度の低いデザインを提供することになるでしょう。そこで、CPLD/FPGA の正しい設計手法を整理・確立し、設計者に深く理解して戴くことで、間接的に ALTERA 社のデバイスが使いやすく、信頼度の高いものであることを知って戴くことを目的としています。それは、デバイス内のデザインを完全に評価され尽くした回路構成でインプリメントすることであり、これにより、デザイン内の不安定要素を限りなく減じることができます。

ALTERA 社の CPLD/FPGA は、高いパフォーマンスと高い耐環境性に優れたデバイスです。しかしながら、ALTERA 社の CPLD/FPGA の動作を取り巻く動作環境は、いろいろ厳しいものがあります。正しく動作させないように、させないようにと外乱が入ります。例えば、使用するシステムの電源が変動したり、周囲の温度も変化します。ALTERA CPLD/FPGA に加えられている電源電圧も常に変化していますし、周囲の動作温度も冬場と夏場ではかなり変化します。しかし、いかなる環境の変化が有ろうともシステムは、正しく動作しなければなりません。そこで、この資料では、各種の変動要素（外乱）とそれに対抗する手段（対策）について記述します。ここに記述されたデザイン・テクニックを設計段階から採用することにより、信頼性の高いシステムを構築することができます。

2 予備知識

CPLD/FPGA が正しく動作することを阻害する要素は、主として、次の点が上げられます。

① 電圧変動
② 周囲温度
③ ロットのバラツキ
④ プロセスの進化（ロットのバラツキとも考えられます。）

上記の外部要因は、極一般的に考えられる要素です。これ以外に身近にある問題として、プリント基板の Vcc/GND パターンの構造やその実装技術が上げられます。特に、GND パターンは、高速デバイスにおけるグラウンド・バウンスの問題を引き起こします。この問題は、デバイス・メーカが低価格化や歩留まり向上の目的でプロセスを微細化することによります。微細化することにより、CPLD/FPGA の動作スピードが増し、従来品では気が付かなかったグラウンド・バウンスが発生したり、ノイズに敏感に反応したりします。これらのことを含めて ALTERA CPLD/FPGA を安心してお使い戴けるように各種のデザイン・テクニックを記述します。

2-1 CPLD/FPGA を不安定にする外部要因

多くの動作不良の原因は、タイミングのバラツキによって起こります。先に上げた様々な要因によってタイミングが変動します。『デバイスを交換したら動作した。』『デバイスを冷やしたら動作した。』と言ったことを良く耳にします。これらは、殆どの場合がタイミングの問題から来ています。ここで、以外と気が付かない問題があります。短期的にはデバイスのロットのバラツキが考えられ、また、長期的にはプロセスの進化による AC タイミングの変化が考えられます。

一方、CPLD/FPGA の AC タイミングの規定方法を見ますと、最悪値（最大値または、最小値）しか定義されておりません。従って、広義の意味では、デバイスのロットが変わったり、プロセス・チェンジを行っても、規定の最悪値に違反しない限り、そのデバイスは正常なデバイスとなります。

これらのことを予め良く理解した上で、設計を行わないと思わぬところで痛い目にあいます。デバイス・メーカがいくら良品と言ってもシステムが動作しなくては元も子も有りません。そこで、デバイス・メーカから出荷されてくる良品（あくまでも、スペック上の）の考え方を整理しておきます。

2-1-1 製造ロット

先に述べたように、CPLD/FPGA の AC パラメータ表には、最悪値しか規定されていません。また、一般的に、あるデバイスは幾つかのスピード・グレードを持っています。例えば、10nS 品、15nS 品、20nS 品と言った具合です。これらのスピード・グレードは個別に最悪値を持ちますが、より遅いデバイスの最悪値はより早いデバイスの最悪値を含んでいます。つまり、15nS 品のデバイスは、10nS 品の性能をも含んでいます。言い換えますと、15nS のデバイスは、10nS 品と 15nS 品の性能を持つ可能性が有ります。これが、最悪値規定の考え方です。このようなことから、CPLD/FPGA の設計時に

曖昧さがあると、タイミングのバラツキが原因で時として誤動作を引き起こすことになります。

2-1-2 プロセス･チェンジ

プロセス･チェンジを行った場合、製造ロットの問題と同様に、旧来品と同じ最悪値は保証されています。しかしながら、プロセス･チェンジによって、AC パラメータは高速化の傾向を持ちます。結果として、新しいプロセス品を使用すると、『ノイズで動かない。』、『グリッチがでる。』、『グラウンド･バウンスによりカウンタが誤動作する。』と言ったことになります。

2-1-3 最悪値とは

ALTERA 社のデータ･ブックのタイミング･パラメータ表には、最大値、あるいは、最小値しか記載されていません。この値は、ALTERA 社が推奨する条件下であれば保証されるものです。ALTERA 社の推奨する条件とは、次のことを指します。

① 使用電圧範囲
② 使用周囲温度範囲
③ ロットのバラツキ（プロセス･チェンジを含む）

ALTERA 社では、上記の条件下で、データ･ブック上の最悪値を保証しています。

ALTERA 社の CPLD/FPGA 開発ソフトウェア MAX+plus II や Quartus II のタイミング･シミュレーションやタイミング･アナリシスをする場合のタイミング･パラメータは、データ･ブックに記載されている最悪値が使用されますので、MAX+plus II ／Quartus II から出力される各計算値は保証された値となります。例えば、タイミング･アナライザから得られた動作周波数はその CPLD/FPGA の最も遅いパスの最低動作周波数を示すものですので、その周波数以下で使用されるシステムの動作周波数は確実に確保できるわけです。同様にして、ディレイ･マトリックスやセットアップ／ホールド･タイム･アナリシスから得られた値をシステム上で確保することにより、ALTERA 社の推奨条件下で確実に動作するようになります。

このような理由から、ALTERA 社の CPLD/FPGA を使用する場合は、実デバイス内に配置配線が完了した後に、タイミング･シミュレータとタイミング･アナライザによる綿密な動作検証を行うことをお勧めします。

以上の事柄を加味して信頼性が高く、安定して動作する回路設計手法を色々な面から解説をして行きます。

3 何を学ぶか？

3-1 何を学ぶか？

ALTERA CPLD/FPGA を使ったシステムの高いデザイン品質を確保するための基本的な考え方は、単一クロックによる同期設計となります。CPLD/FPGA が大規模になるにつれて、高速性を維持しながらも柔軟性を確保するために各ベンダ毎にインターコネクトの方式が提案されています。しかしながら、一つのデバイス全体を通じてあらゆるデバイス内のスキューが均一になるように配置することは現実として不可能です。さらに、これらのスキューは、先に述べた外乱によって度々変動します。このような条件の下で非同期設計を行うとクロック信号とデータ信号との間で競争が発生します。時には、データがクロックを追い越してしまう『すっぽ抜け』の状態が発生します。

このようなことを防ぐ為には、ALTERA 社が推奨する最悪条件を維持しながら、同期設計を行う必要があります。同期設計を行う限り、デバイス内でクロック・スキューが最小になるように調整されていますので、『すっぽ抜け』を心配する必要は有りません。唯一注意することは、外部からの入力信号のセットアップ・タイムとホールド・タイムを確保するだけです。

同期設計を前提にデザインを進める上でも、幾つかの考慮すべき点があります。
それらは：

① マスタ・リセット回路
② プリセット／クリア回路
③ クロック回路
④ 非同期入力回路
⑤ 帰還型 SR フリップ・フロップ
⑥ ディレイ素子回路
⑦ 自己帰還回路
⑧ On-chip XOR 回路とスタティック・ハザード
⑨ イリーガル・ステートからの脱出
⑩ カウンタとグラウンド・バウンス

以上の考慮すべき回路構成について、真剣に回路検討を行った後に対策が施されたデザインは、高い設計品質が維持されることになります。

この資料では、一貫して「同期化回路」について学びます。その前に、同期回路と非同期回路の違いについて触れるとともに、フリップ・フロップ（レジスタ）の非同期制御入力端子の挙動について触れておきます。

3-2 同期回路と非同期回路の違い

『CPLD/FPGA 設計は、同期化設計に尽きる。』と言っても過言ではありません。そこで、まず、同期と非同期の違いと各々の方式のメリット／デメリットを説明します。

3-2-1 同期回路とは

『同一クロックの同一エッジに同期して動作する回路系』を言います。従って、同一クロックであっても逆位相のエッジを使用する場合は、同一クロックと見なしません。また、基本的には、単一クロック同期が望ましいことになります。異なるクロック間での信号の授受は、非同期回路となり、後述する非同期入力の処理が必要になります。

➢ 同期回路のメリットとデメリット…

同期回路のメリットとしては、以下の点が上げられます。

- □ タイミングが取り易い。
- □ グリッチ･フリーのシステムが構成可能。
- □ 高速動作が可能。
- □ デバッグがし易い。

一方、同期回路のデメリットとしては…

- □ 消費電力が大きい。
- □ 回路規模が大きくなる。

同期設計の回路例を図 1、VHDL 記述を図 2、VerilogHDL 記述を図 3 に示します。

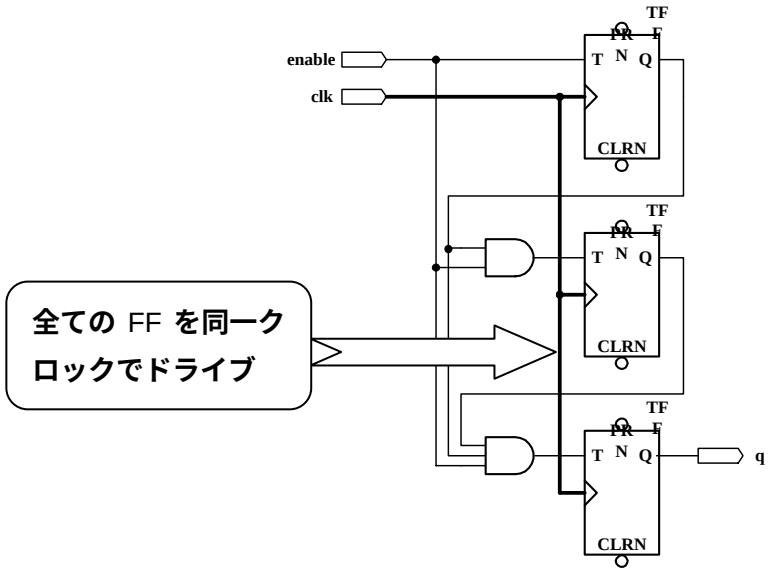


図 1. 同期回路の回路例

図 2. 同期回路のVHDL記述例

```
ARCHITECTURE sync_pld OF s_counter IS

BEGIN
  PROCESS (clk)

  VARIABLE cnt : std_logic_vector(2 downto 0);

  BEGIN
    IF enable = '0' THEN
      cnt := cnt;
        ELSIF (rising_edge(clk)) THEN
      cnt := cnt + 1;
    END IF;
    q <= cnt(2);
  END PROCESS;
END sync_pld;
```

図 3. 同期回路のVerilogHDL記述例

```
module s_counter (clk, enable, q);
  input  clk, enable;
  output   q;

  reg [2:0] cnt;

always @(posedge clk)
  if (enable)
    cnt = cnt+1;
  else
    cnt = cnt;
  assignq = cnt[2];

endmodule
```

3-2-2 非同期回路とは

『同一クロックの同一エッジに同期して動作しない回路系』を言います。従って、同一クロックであっても異なるクロック･エッジで動作する回路は非同期回路となります。リップル･クロックは、その最たる例です。また、内部で分周した出力を再利用することも源振クロックに対し非同期クロックとなります。

- **非同期回路のメリットとデメリット…**

非同期回路のメリットとしては、以下の点が上げられます。

- □ 回路規模が小さい。
- □ 消費電力が小さい。

一方、非同期回路のデメリットとしては…

- □ タイミングが取り難い。
- □ 動作周波数が低い。
- □ シミュレーションが難しい。

非同期設計の回路例を図 4、VHDL 記述を図 5、VerilogHDL 記述を図 6 に示します。

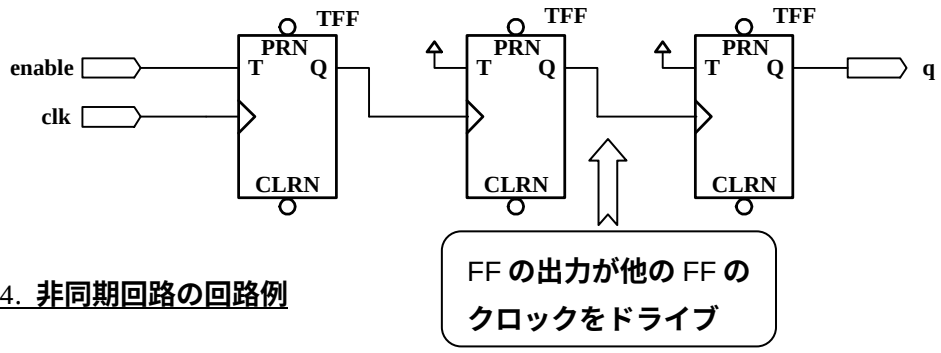


図 4. 非同期回路の回路例

図 5. 非同期回路のVHDL記述例

```
ARCHITECTURE async_pld OF a_counter IS
SIGNAL cnt2, cnt1, cnt0: std_logic;
BEGIN
 q0:PROCESS (clk) BEGIN
    IF (enable='0')          THEN cnt0 <= cnt0;
    ELSIF rising_edge (clk) THEN cnt0 <= not cnt0;
    END IF;
  END PROCESS q0;

 q1:PROCESS (cnt0) BEGIN
    IF rising_edge (cnt0) THENcnt1 <= not cnt1;
    ELSE       cnt1 <= cnt1;
                  END IF;
                END PROCESS q1;

                 q2:PROCESS (cnt1) BEGIN
                   IF rising_edge (cnt1) THENcnt2
               <= not cnt2;
                   ELSE       cnt2 <= cnt2;
                   END IF;
                 END PROCESS q2;
                 q <= cnt2;
               END async_pld;
```

図 6. 非同期回路のVerilogHDL記述例

```
reg cnt0, cnt1, cnt2;

always @(posedge clk) begin
      if (enable) cnt0 = ~cnt0;
      else   cnt0 = cnt0;
      end

always @(posedge cnt0)
      cnt1 = ~cnt1;

always @(posedge cnt1)
      cnt2 = ~cnt2;

assign q = cnt2;

endmodule
```

図 7 に逆位相クロックを使用した回路例を、図 8 に VHDL 記述例、図 9 に VerilogHDL の記述例を示します。

図 7. 逆位相クロックの例

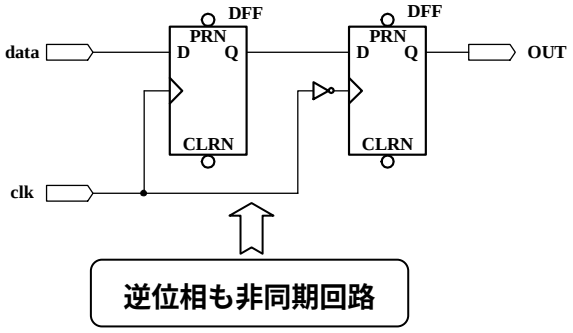


図 8. 逆位相クロックのVHDL記述例

```
ARCHITECTURE rev_clk_pld OF rev_clk IS
SIGNAL int0, int1: std_logic;

BEGIN

  PROCESS (clk) BEGIN
    IF (rising_edge(clk)) THEN
      int0 <= data;
    ELSE
      int0 <= int0;
    END IF;
  END PROCESS;

  PROCESS (clk) BEGIN
    IF (falling_edge(clk)) THEN
      int1 <= int0;
    ELSE
      int1 <= int1;
    END IF;
  END PROCESS;
  q <= int1;

END rev_clk_pld;
```

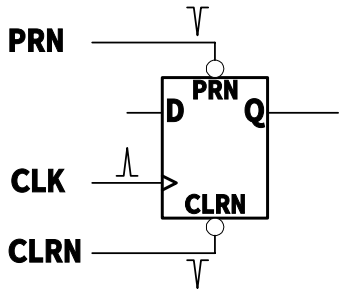
図 9. 逆位相クロックのVerilogHDL記述例

```
module rev_clk (clk, data, q);
  input    clk, data;
  output   q;

reg int, q;

always @(posedge clk)
     int = data;

always @(negedge clk)
     q = int;

endmodule
```

3-3 敏感なフリップ･フロップの入力制御端子

フリップ･フロップのクロックと非同期クリア／プリセット端子における入力信号の品質は、システムに重大な影響を与えます。これらの入力端におけるハザード入力は、システムに予期しない悪影響を及ぼします。図 10 にその概念図を示します。

図 10. ハザードに敏感に反応するFFの入力制御端子



ここまでで、この資料で述べる項目の予備知識を整理しました。以後、順を追って高い信頼性を確保するための設計手法について説明します。

なお、5 項以後に関連した項目について触れていますので、あわせてご一読ください。

4 超安定回路の設計

この項に記載されている安定回路の設計手法を採用することにより、デバイス内で発生する不安定な要素を最小限にすることができます。

4-1 マスタ･リセット回路

ALTERA CPLD/FPGA は、電源投入時に全ての内蔵フリップ･フロップ（レジスタ）を初期化するためのパワー･オン･リセット(POR)回路を内蔵しています。しかしながら、この回路は性悪な電源回路の場合に正しく動作しないケースがあります。従って、一義的、かつ、正確に初期化すために、ボード･レベルでのマスタ･リセット信号を付けることを強くお勧めします。

あるデザインで、フリップ･フロップのリセット端子を使用しない場合を良く見受けます。この場合は、入力信号が確定するまで出力が確定しないことになりますので余りお勧めできません。また、カウンタやステートマシンの場合は、電源投入時にイリーガル･ステートから開始される場合があります。特に、このケースでは、明示的な脱出回路がない限りイリーガル･ステートから脱出することはできません。そのシステムは、電源再投入まで動作することはありません。

このようなことから、

フリップ･フロップ（レジスタ）を使用した回路は、一義的にマスタ･リセット信号でリセットされる回路構成にすることを強くお勧めします。

補足： フリップ･フロップの非同期プリセットや非同期クリア端子は、ハザードに対し非常に敏感ですから、ロジックで構成した信号を非同期制御端子に入力することはお勧めできません。同期プリセットや同期クリアの使用をお勧めします。非同期のプリセット端子やクリア端子は、ボード･レベルのマスタ･リセット信号の入力のみとし、回路出力によるプリセット／リセット信号は同期式を使用することにより、信頼性の高いリセット･システムを構築することができます。

4-2 プリセット／クリア回路

プリセット／クリア信号は、後述するクロック信号と同様に慎重に供給する必要があります。その理由は、これらの非同期入力信号上のハザードに対し、フリップ･フロップが敏感に反応するからです。

次に上げるガイドラインが信頼性の高いロジックを構成する上で役立ちます。

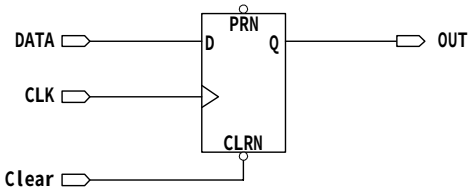
① プリセットやクリア信号を入力する場合は、クロック信号と同様デバイスのピンから直接駆動されるグローバル信号を使ってください。

② もし、ロジックによってプリセット／クリア信号を生成する場合は、1本のピン、あるいは、1つのフリップ･フロップ出力だけで、プリセット／クリア信号を生成してください。その他の信号は、制御信号としてのみ使用してください。(図13参照。) この制御信号は、プリセット／クリア信号がアサートされている間は、ステーブルにして置く必要があります。

③ プリセット／クリア信号は、複数の信号をセレクタで切り替えて使用するようなマルチレベルのロジックを使って生成しないでください。

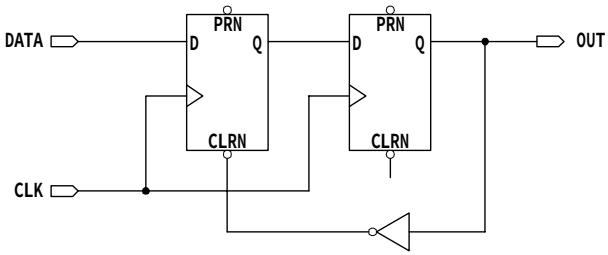
図 11～14 に ALTERA デバイスとして信頼性の高いプリセット／クリア回路を示します。

図 11. ピン･ドライブによるクリア回路



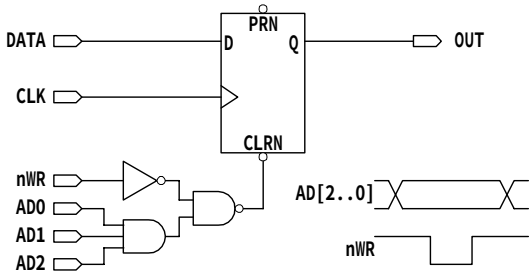
※ この FF のクリア端子は、外部入力 Clear 端子に接続されています。

図 12. レジスタ･ドライブによるクリア回路



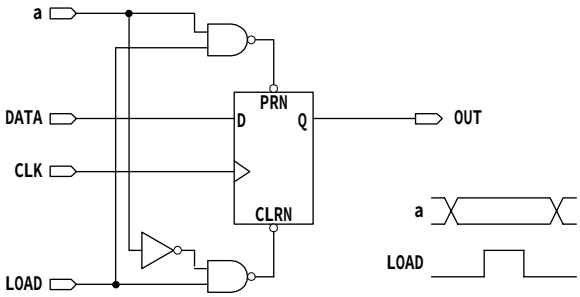
※ 初段の FF のクリア端子は、2 段目の FF 出力に接続されており、この出力にはグリッチはありません。

図 13. ゲーティド･クリア回路



※ クリア端子に接続されているゲート回路入力の ADx 信号が変化している間は、nWR 信号は、”1”であることが必要です。

図 14. 非同期ロード回路



※ 入力信号 a が変化している間は、LOAD 信号は、”0”であることが必要です。

図 15〜16 は、図 12 の VHDL と VerilogHDL 記述例です。図 17〜18 は、図 13 の VHDL と VerilogHDL 記述例です。

図 15. レジスタ･ドライブによるクリア回路のVHDL記述例

```
ARCHITECTURE pld OF a_clr_w_reg IS
SIGNAL   int : std_logic;

BEGIN
  PROCESS (CLK, q) BEGIN
    IF (q='1') THEN int <= '0';
    ELSIF rising_edge (clk) THEN
      int <= data;
    END IF;
  END PROCESS;

  PROCESS (clk) BEGIN
     IF rising_edge (clk) THEN
        q <= int;
    END IF;
   END PROCESS;

END pld;
```

図 16. レジスタ･ドライブによるクリア回路のVerilogHDL記述例

```
module a_clr_w_reg (clk, data, q);
input    clk, data;
output   q;

reg int, q;

always @(posedge clk or posedge q)
if (q)   int = 1'b0;
else int = data;

always @(posedge clk)
   q = int;

endmodule
```

図 17. ゲーティド・クリア回路のVHDL記述例

```
ARCHITECTURE pld OF gated_clr IS
SIGNAL node : std_logic;

BEGIN
 PROCESS (nwr, ad) BEGIN
    node <= not nwr AND ad(2) AND ad(1) AND ad(0);
 END PROCESS;

 PROCESS (clk, node) BEGIN
    IF (node = '1') THEN
       q <= '0';
    ELSIF rising_edge (clk) THEN
       q <= data;
    END IF;
 END PROCESS;
END pld;
```

図 18. ゲーティド・クリア回路のVerilogHDL記述例

```
module gated_clr (clk, data, nwr, ad, q);
 input     clk, data, nwr;
 input  [2:0] ad;
 output    q;

reg q;
reg node;

always @(nwr or ad)
 node = ~nwr & (ad == 3'h7);

always @(posedge clk or posedge node)

 if (node) q = 1'b0;
 else q = data;

endmodule
```

4-3 クロック回路

信頼性の高いクロック・システムは、いかなるディジタル設計においても動作を確実に行う為に非常に重要な問題です。不用意に設計されたクロック構成では、デバイスの AC タイミングが変化すると、エラーを引き起こします。

ALTERA 社では、可能な限りグローバル・クロックを使うことをお勧めします。仮に、デザイン内で単一グローバル・クロックとしてインプリメントできない場合は、ゲーティド・クロック(Gated Clock)のような他のクロック構成を取らざるを得ませんが、次のようなクロック構成はシステムの信頼性を低下させますので、お勧めできません。

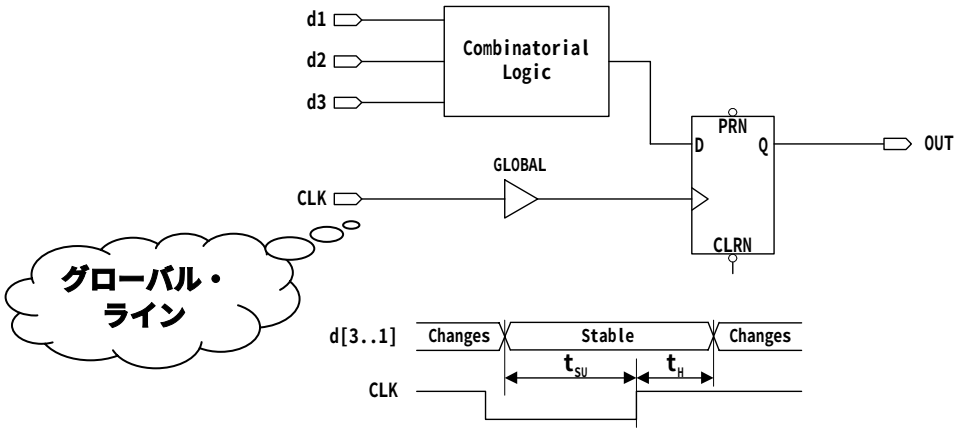
① リップル・クロック
② マルチレベル・クロック
③ マルチ・クロック・システム
④ ステートマシンでドライブされるクロック

以下に各クロック方式の構成の仕方と注意点について記述します。

4-3-1 グローバル・クロック (Global Clock)

ALTERA CPLD/FPGA のクロック・システムの中で、入力専用ピン、あるいは、クロック専用ピンから入力されたグローバル・クロックは、最もシンプルで信頼性の高いクロック・システムとなります。クロック・ピンから入力されたクロック信号は、デバイス内で直接クロック専用ラインに接続されています。また、デバイス内では、デバイスの隅々でクロック・スキューが均一になるように予め調整されています。従って、グローバル・クロック構成を採用することにより、最も高速でしかも信頼性の高いシステムを構築することができます。図 19 にグローバル・クロックを使用した回路の例を示します。

図 19. グローバル・クロック



4-3-2 ゲーティド・クロック (Gated Clock)

ALTERA CPLD/FPGA では、フリップ・フロップ毎にクロックを独立させることができるアレイ・クロック (非同期クロック) を持っています。しかし、アレイ・クロックを使用した場合は、スキューやグリッチの発生に十分注意が必要です。普通、アレイ・クロックは、ゲート回路でクロックを制御するようなゲーティド・クロック構造を取り、マイクロプロセッサのアドレス・ラインをライト・ストローブによってゲートするような場合に良く使われます。

ゲーティド・クロックも次のような条件で使用するとグローバル・クロックと同様に信頼性の高いシステムとして使用することができます。

- クロック・ゲート内で実際にクロックとして動作する入力が1本の場合。
- クロックをドライブするロジックが1つのANDまたはORからできている場合。

図 20 に信頼性の高いゲーティド・クロックの例を示します。図 20 では、1つの AND ゲートでゲーティド・クロックを生成しています。この例では、ピン nWR がクロック・ピンとなり、ピン AD[2..0]がアドレス信号となります。

図 20. ANDゲート・クロック

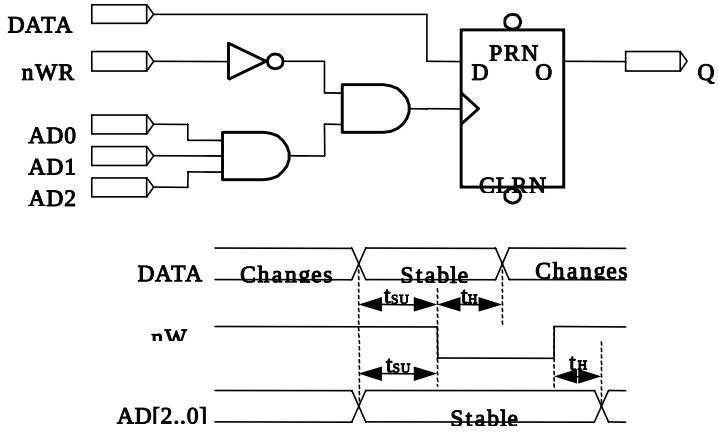


図 20 のタイミング波形は、デバイスが要求するセットアップ／ホールド・タイムを示しています。アドレス信号は、クロック(nWR)がアサートされている間は、ステーブルの必要があります。もし、クロックがアサートされている間にアドレスが変化した場合は、クロックにグリッチが発生し、フリップ・フロップが誤動作する原因となります。一方、データ・ピン DATA は、nWR の立ち下がりエッジに対し、セットアップ・タイムとホールド・タイムを保証する必要があります。

このようなゲーティド・クロックをグローバル・クロック方式に変更することによって、デザインの信頼性を向上させることができます。図 21 に図 20 の回路をグローバル方式に変更した例を示します。ア

ドレス・ラインは、DFFE フリップ・フロップのクロック・イネーブル(Clock Enable)をコントロールしています。ENA がハイの時、D 入力の値をフリップ・フロップ内に読み込みます。ENB がローの場合は、フリップ・フロップの値をホールドします。

図 21. AND ゲート・クロックをグローバル・クロックに変更した例

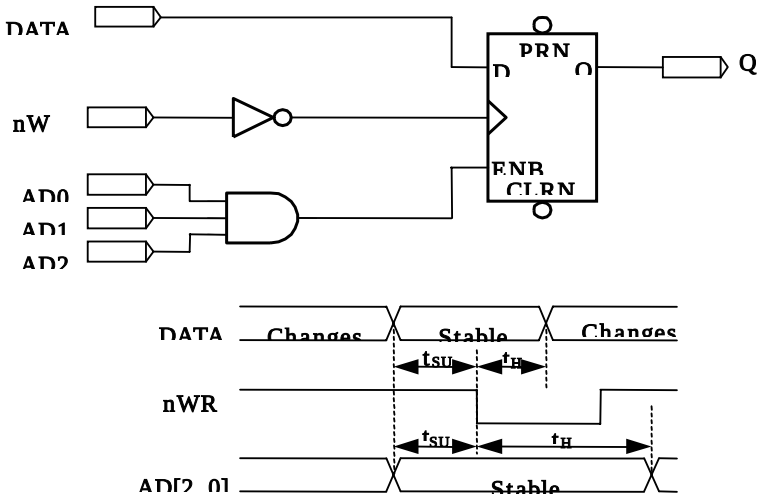


図 21 のタイミングでは、アドレス信号は、nWR がアサートされていてもステーブルである必要はありません。その代わりに、セットアップ・タイムとホールド・タイムだけは守る必要があります。

図 20 を図 21 に改善した場合の VHDL の記述例を図 22 に、VerilogHDL の記述例を図 23 に示します。

図 22. ANDゲート・クロックの改善例 (VHDL)

※ 図 20 のVHDL記述例

```
ARCHITECTURE pld OF and_clk IS
SIGNAL node : std_logic;

BEGIN
    PROCESS (nwr, ad) BEGIN
        node <= not nwr AND ad(2)
                        AND ad(1)
                        AND ad(0);
    END PROCESS;
    PROCESS (node) BEGIN
        IF rising_edge (node) THEN
            q <= data ;
        END IF;
    END PROCESS;
END pld;
```

改善例

※ 図 21 のVHDL記述例

```
ARCHITECTURE pld OF enb_clk IS
SIGNAL node : std_logic;

BEGIN
    PROCESS (ad) BEGIN
        node <= ad(2) AND ad(1)
                      AND ad(0);
    END PROCESS

    PROCESS (nwr) BEGIN
        IF (node = '0') THEN
            q <= q;
        ELSIF falling_edge (nwr) THEN
            q <= data ;
        END IF;
    END PROCESS;
END pld;
```

図 23. ANDゲート・クロックの改善例(VerilogHDL)

※ 図 20 のVerilogHDL記述例

```
reg q;
reg node;

always @(nwr or ad)
    node = ~nwr & (ad
==3'h7);

always @(posedge node)
    q = data;

endmodule
```

改善例

※ 図 21 のVerilogHDL記述例

```
reg q;
reg node;

always @(ad)
    node = (ad == 3'h7);

always @(negedge nwr)
    if (node)
        a = data;

endmodule
```

図 24 にゲーティド・クロックの悪い使用例を示します。この例では、3 ビットの同期カウンタから生成される rco 信号がフリップ・フロップのクロックとして使われています。フリップ・フロップのクロックは、カ

ウンタの出力を AND した後に入力されていますので、カウンタの各ビットのスイッチング・タイムのバラツキで、AND 出力にグリッチを発生することがあります。例えば、図 24 の例では、カウンタの値が 3 から 4 に変化した時に rco 信号にグリッチが発生しています。

図 24. ゲーティド・クロックの悪い例

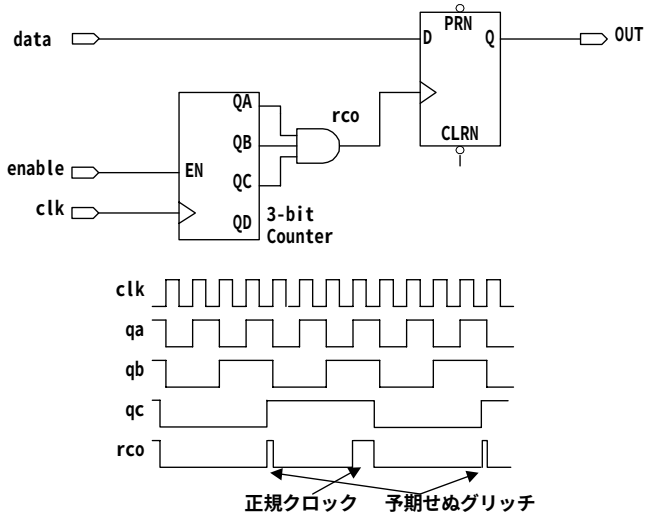


図 25 に図 24 の回路をグローバル・クロック化した信頼性の高い回路を示します。rco 信号は、DFFE のクロック・イネーブル入力をコントロールしています。図 25 の例では、カウンタの値が 6 の時にクロック・イネーブルがハイとなり、次のクロック・エッジでデータをフリップ・フロップに読み込みます。

図 25. 信頼性の低いゲーティド・クロックをグローバル・クロックに変更した例

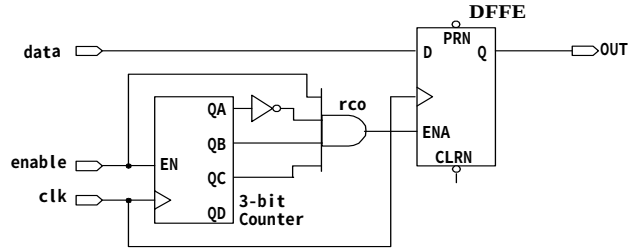


図24を図25に改善した場合の VHDL の記述例を図26に、VerilogHDL の記述例を図27に示します。

図 26. ゲーティド・クロックの悪い例(VHDL)

※ 図 24 のVHDL記述例

```
ARCHITECTURE pld OF gated_clk IS
SIGNAL node : std_logic;
SIGNAL intq : std_logic_vector (2 downto 0);

BEGIN
  PROCESS (clk) BEGIN
    IF (enable = '0') THEN
      intq <= intq;
    ELSIF rising_edge (clk) THEN
      intq <= intq + 1;
    END IF;
  END PROCESS;

  PROCESS (intq) BEGIN
    node <= intq(2) AND intq(1)
                    AND intq(0);
  END PROCESS;

  PROCESS (node) BEGIN
    IF rising_edge (node) THEN
      q <= data;
    END IF;
  END PROCESS;

END pld;
```

改善例

※ 図 25 のVHDL記述例

```
ARCHITECTURE pld OF s_clk IS
SIGNAL node : std_logic;
SIGNAL intq : std_logic_vector (2 downto 0);

BEGIN
  PROCESS (clk) BEGIN
    IF (enable = '0') THEN
      intq <= intq;
    ELSIF rising_edge (clk) THEN
      intq <= intq + 1;
    END IF;
  END PROCESS;

  PROCESS (intq, enable) BEGIN
    node <= enable AND intq(2) AND intq(1)
                    AND not intq(0);
  END PROCESS;

  PROCESS (clk) BEGIN
    IF (node = '0') THEN
      q <= q;
    ELSIF rising_edge (clk) THEN
      q <= data;
    END IF;
  END PROCESS;
END pld;
```

図 27. ゲーティド･クロックの悪い例(VerilogHDL)

※ 図 24 のVerilogHDL記述例

```
module gated_clk (clk, data, enable, q);

input clk, data, enable ;
output   q ;

reg node;
reg q;
reg [2:0]intq;

always @(posedge clk)
      if (enable)
            intq = intq + 1;
      else
            intq = intq;

always @(intq)
      node = intq == 3'h7;

always @(posedge node)
      q <= data;

endmodule
```

改善例

※ 図 25 のVerilogHDL記述例

```
module s_clk (clk, data, enable, q);

input clk, data, enable ;
output   q ;

reg node;
reg q;
reg [2:0]intq;

always @(posedge clk)
      if (enable)
            intq = intq + 1;
      else
            intq = intq;

always @(intq or enable)
      node = enable & (intq == 3'h6);

always @(posedge clk)
      if (node)
            q <= data;

endmodule
```

4-3-3 マルチレベル・クロック (Multi-Level Clock)

コンビナトリアル・ロジックでゲーティド・クロックを生成する場合、1 レベル以上の AND や OR ゲートを使って生成されるクロックのことをマルチレベル・クロックと言います。この場合は、デザインの信頼性を検証することが非常に難しくなります。このようなマルチレベル・クロックを使ったロジックでは、実際の試作回路やシミュレーションで発見できないスタティック・ハザードが発生する場合があります。

☞ : **デザイン内でフリップ・フロップ用クロックを生成するためにマルチレベルのコンビナトリアルを使って生成しないでください。**

マルチレベル・ロジック内に存在するハザードを取り除くためには、一般的にはデザイン内に冗長回路を挿入しますが、MAX+plus II／Quartus II の論理合成機能によってこの冗長回路が取り除かれてしまう場合があります。従って、このような不意の事故を避けるために、次の例を参考にして、明示的に解決することをお勧めします。

図 28 では、スタティック・ハザードを発生する可能性のあるマルチレベル・クロック構成の例です。このクロックは、ピン sel によってコントロールされるマルチプレクサの出力です。マルチプレクサ入力は、clk クロックとそのクロックを半分に分周した div2 クロックです。図 28 に示されるタイミング波形は、両方のクロックがロジック"1"の間に sel 信号が変化するとスタティック・ハザードを発生する例です。

図 29 は、図 28 の回路を 1 レベル・クロック構成に変更した例です。ここでは、ピン sel と div2 信号が、DFFE フリップ・フロップのクロック・イネーブルをコントロールしています。

図 28. スティック・ハザードを発生するマルチレベル・クロック

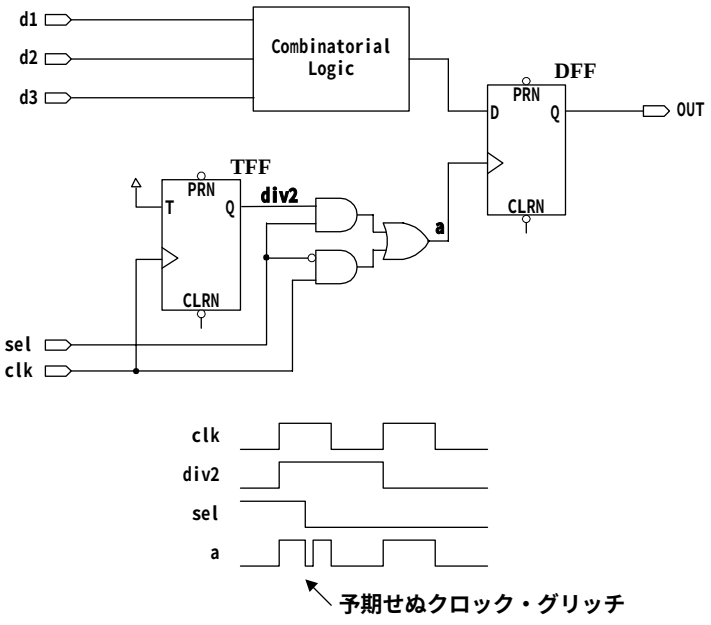


図 29. スタティック・ハザードを発生しない 1 レベル・クロック

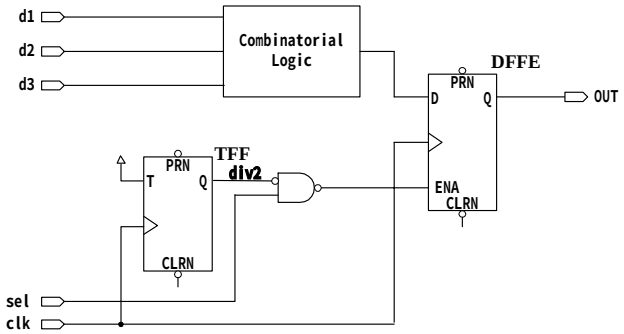


図28を図29に改善した場合のVHDLの記述例を図30に、VerilogHDLの記述例を図31に示します。

図 30. スティック・ハザードを発生するマルチレベル・クロック(VHDL)

※ 図 28 のVHDL記述例

```
ARCHITECTURE pld OF a_ml_clk IS
SIGNAL div2 : std_logic;
SIGNAL node : std_logic;

BEGIN
  PROCESS (clk) BEGIN
    IF rising_edge (clk) THEN
      div2 <= not div2;
    END IF;
  END PROCESS;

  PROCESS (div2, sel, clk) BEGIN
    node <= (sel AND div2)
          OR (not sel AND clk);
  END PROCESS;

  PROCESS (node) BEGIN
    IF rising_edge (node) THEN
      q <= data;
    END IF;
  END PROCESS;

END pld;
```

改善例

※ 図 29 のVHDL記述例

```
ARCHITECTURE pld OF s_ml_clk IS
SIGNAL div2 : std_logic;
SIGNAL node : std_logic;

BEGIN
  PROCESS (clk) BEGIN
    IF rising_edge (clk) THEN
      div2 <= not div2;
    END IF;
  END PROCESS;

  PROCESS (div2, sel) BEGIN
    node <= not(not div2 AND sel);
  END PROCESS;

  PROCESS (clk) BEGIN
    IF (node = '0') THEN
      q <= q;
    ELSIF rising_edge (clk) THEN
      q <= data;
    END IF;
  END PROCESS;
END pld;
```

図 31. スティック・ハザードを発生するマルチレベル・クロック(VerilogHDL)

※ 図 28 のVerilogHDL記述例

```
module a_ml_clk (clk, data, sel, q);

input clk, data, sel;
output q;

reg q;
reg div2;
reg node;

always @(posedge clk)
   div2 = ~div2;

always @(div2 or sel or clk)
   node = (sel & div2) | (~sel & clk);

always @(posedge node)
   q = data;

endmodule
```

改善例

※ 図 29 のVerilogHDL記述例

```
module s_ml_clk (clk, data, sel, q);

input clk, data, sel;
output q;

reg q;
reg div2;
reg node;

always @(posedge clk)
   div2 = ~div2;

always @(div2 or sel)
   node = ~(sel & ~div2);

always @(posedge clk)
   if (node)
      q = data;

endmodule
```

4-3-4 リップル・クロック (Ripple Clock)

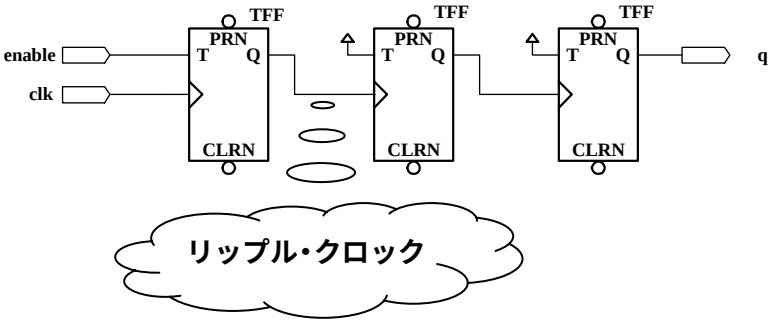
あるフリップ・フロップの出力が他のフリップ・フロップのクロック入力となるようなクロック構成をリップル・クロック(Ripple Clock)方式と言います。リップル・クロックは、グローバル・クロック方式と同様信頼性の高いクロック・システムを構成可能ですが、リップル・チェイン内の複数のフリップ・フロップ出力をコンビナトリアル・ロジックに供給している場合には、後段のコンビナトリアル・ロジック出力でハザードが発生します。また、リップル・クロックによる回路のタイミング計算は、複雑になります。さらに、リップル・クロック方式では、リップル・チェイン内のフリップ・フロップのクロック間で大きなスキューを発生し、ワースト・ケースのセットアップ・タイム、ホールド・タイム、あるいは、クロックからの出力ディレイを大きくする原因となります。結果として、実際に動作するシステムの動作周波数を下げます。

リップル・カウンタを同期カウンタに変更することで、より高速に動作させることができます。図 32～35 を参考にして信頼性の高いデザインに変更してください。

リップル・クロックは、図 32 のT型フリップ・フロップを使ったリップル・カウンタのように、1 つのフリッ

プ･フロップ出力が次のフリップ･フロップのクロック入力として使われます。

図 32. リップル･カウンタ



リップル･カウンタは、同期カウンタに変更することができます。図 33 にグローバル･クロック化した例を示します。この構成では、クロックからの出力が高速に得られます。

図 33. リップル･カウンタをグローバル･クロック方式に変更した例

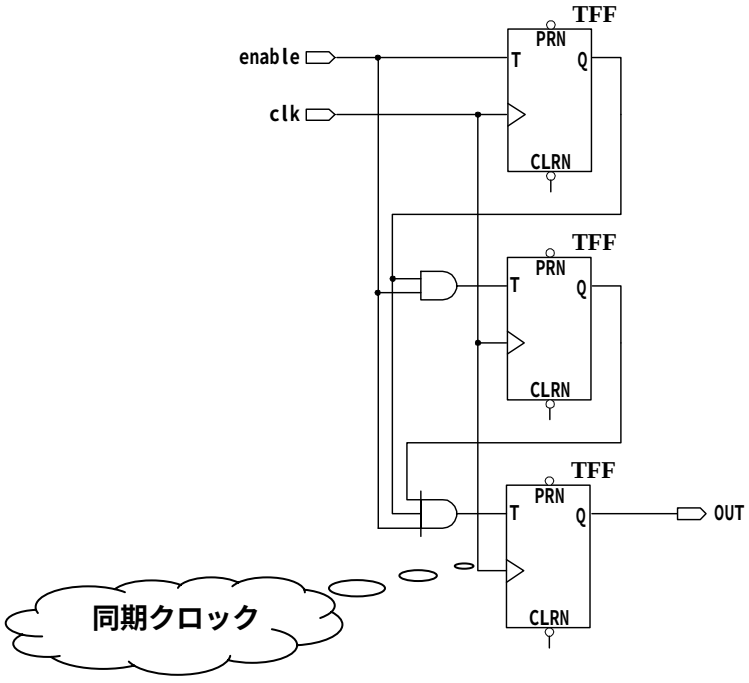


図 32 を図 33 に改善した場合の VHDL の記述例を図 34 に、VerilogHDL の記述例を図 35 に示します。

図 34. リップル・カウンタ(VHDL)

※ 図 32 のVHDL記述例

```
ARCHITECTURE async_pld OF
                        a_counter IS

SIGNAL cnt2, cnt1, cnt0: std_logic;

BEGIN

  q0:PROCESS (clk) BEGIN
    IF (enable='0') THEN
      cnt0 <= cnt0;
    ELSIF rising_edge (clk) THEN
      cnt0 <= not cnt0;
    END IF;
  END PROCESS q0;

  q1:PROCESS (cnt0) BEGIN
    IF rising_edge (cnt0) THEN
      cnt1 <= not cnt1;
    ELSE
      cnt1 <= cnt1;
    END IF;
  END PROCESS q1;

  q2:PROCESS (cnt1) BEGIN
    IF rising_edge (cnt1) THEN
      cnt2 <= not cnt2;
    ELSE
      cnt2 <= cnt2;
    END IF;
  END PROCESS q2;

  q <= cnt2;

END async_pld;
```

改善例

※ 図 33 のVHDL記述例

```
ARCHITECTURE sync_pld OF
                        s_counter IS

BEGIN
  PROCESS (clk)

  VARIABLE cnt : std_logic_vector
                        (2 downto 0);

  BEGIN
    IF enable = '0' THEN
      cnt := cnt;
    ELSIF (rising_edge(clk)) THEN
      cnt := cnt + 1;
    END IF;

    q <= cnt(2);

  END PROCESS;
END sync_pld;
```

図 35. リップル・カウンタ(VerilogHDL)

※ 図 32 のVerilogHDL記述例

```
module a_counter (clk, enable, q);
   input    clk, enable;
   output   q;

reg cnt0, cnt1, cnt2;

always @(posedge clk) begin
      if (enable) cnt0 = ~cnt0;
      else        cnt0 = cnt0;
      end

always @(posedge cnt0)
      cnt1 = ~cnt1;

always @(posedge cnt1)
      cnt2 = ~cnt2;

assign q = cnt2;
```

改善例

※ 図 33 のVerilogHDL記述例

```
module s_counter (clk, enable, q);
   input clk, enable;
   output   q;

reg [2:0] cnt;

always @(posedge clk)
      if (enable) cnt = cnt+1;
      else        cnt = cnt;

assign q = cnt[2];

endmodule
```

リップル・クロック方式におけるクロックの周期は、最下位桁の変化が最上位桁へ伝播するまでの時間に依存します。従って、カウンタの桁数に依存し、桁数が増すにつれて動作速度が低下します。この場合も同期カウンタを使うことにより、システム・スピードは、カウンタの桁数には無関係で、フリップ・フロップのセットアップ・タイムとホールド・タイムによって決定されます。この同期カウンタは、カウンタのスピードを改善し、しかも、カウンタが無効な値を持つことはありません。

4-3-5 マルチ・クロック・ネットワーク

あるアプリケーションでは、1つのデザイン内に複数のクロックを持つ場合があります。一般的な例では、2つの非同期マイクロプロセッサやマイクロプロセッサと非同期コミュニケーション・チャネル間のインタフェース設計の場合です。これらのアプリケーションでは、クロック信号間に必要とされるセットアップ・タイムとホールド・タイムによって拘束されます。

マルチ・クロック・システムにおいては、非同期信号の同期化が必要となります。同期化を行わない信号に対しては、メタステーブル状態を起こす可能性が有ります。図 36 にマルチ・クロック・システムの例を示します。図 37 にその改善例を示します。

図 36 において、clk-a が reg-a をクロッキングし、clk-b が reg-b をクロッキングします。clk-b に関係するセットアップ／ホールド･タイムは、clk-a でドライブされる reg-a によって決ります。しかし、clk-a と clk-b は同期していませんので、clk-b の立ち上がりに対し、reg-b に必要なセットアップ･タイムを満たさないケースが出てきます。

図 36. マルチ･クロック･システム

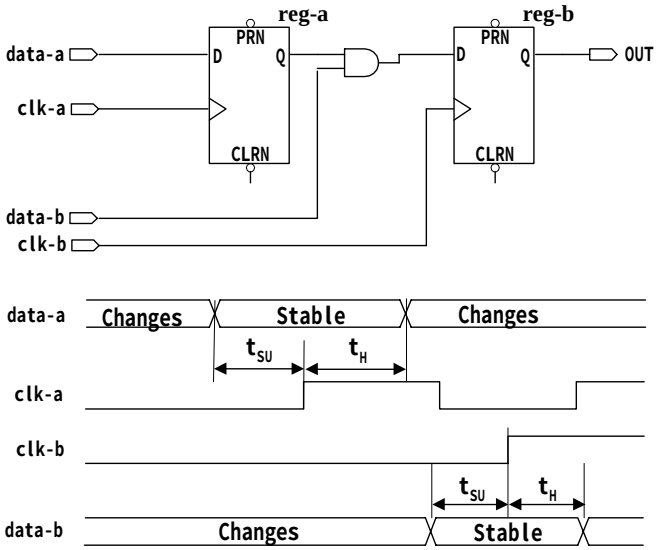


図 36 の回路では、2つの独立したクロックを持っています。このような場合は、両者のセットアップ･タイムとホールド･タイムを確実に保証する方法はありません。そこで、図 37 に示すように reg-a と reg-b の間にフリップ･フロップ reg-c を挿入し、そのフリップ･フロップを clk-b でトリガすることにより同期化を行います。これによって、reg-b のセットアップ･タイムを保証することができます。しかし、この方法では余分なフリップ･フロップが必要なことと1クロック･サイクル分のディレイが発生します。

図 37. 同期用フリップ･フロップを挿入したマルチ･クロック･システム

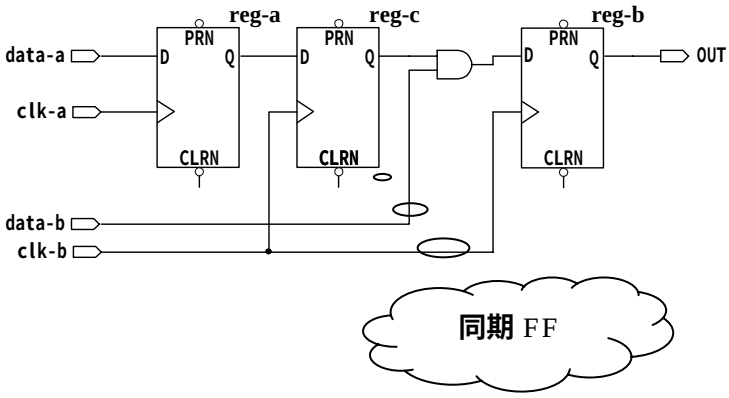


図 36 を図 37 に改善した場合の VHDL の記述例を図 38 に、VerilogHDL の記述例を図 39 に示します。

図 38. マルチ･クロック･システム(VHDL)

※ 図 36 のVHDL記述例

```
ARCHITECTURE pld OF a_m_clk IS
SIGNAL qa, node : std_LOGIC;

BEGIN
  PROCESS (clk_a) BEGIN
    IF rising_edge (clk_a) THEN
      qa <= data_a;
    END IF;
  END PROCESS;

  PROCESS (qa,data_b) BEGIN
    node <= qa AND data_b;
  END PROCESS;

  PROCESS (clk_b) BEGIN
    IF rising_edge (clk_b) THEN
      q <= node;
    END IF;
  END PROCESS;

END pld;
```

改善例

※ 図 37 のVHDL記述例

```
ARCHITECTURE pld OF s_m_clk IS
SIGNAL qa, qb : std_LOGIC;

BEGIN
  PROCESS (clk_a) BEGIN
    IF rising_edge (clk_a) THEN
      qa <= data_a;
    END IF;
  END PROCESS;

  PROCESS (clk_b) BEGIN
    IF rising_edge (clk_b) THEN
      qb <= qa;
    END IF;
  END PROCESS;

  PROCESS (clk_b) BEGIN
    IF rising_edge (clk_b) THEN
      q <= qb AND data_b;
    END IF;
  END PROCESS;

END pld;
```

図 39. マルチ・クロック・システム(VerilogHDL)

※ 図 36 のVerilogHDL記述例

```
module a_m_clk (data_a, clk_a,
                data_b, clk_b, q);

input data_a, data_b;
input clk_a, clk_b;
output q;

reg node;
reg q, qa, qb;

always @(posedge clk_a)
   qa = data_a;

always @(qa or data_b)
   node = qa & data_b;

always @(posedge clk_b)
   q = node;

endmodule
```

改善例

※ 図 37 のVerilogHDL記述例

```
module s_m_clk (data_a, clk_a,
                data_b, clk_b, q);

input data_a, data_b;
input clk_a, clk_b;
output q;

reg q, qa, qb;

always @(posedge clk_a)
   qa = data_a;

always @(posedge clk_b)
   qb = qa;

always @(posedge clk_b)
   q = qb & data_b;

endmodule
```

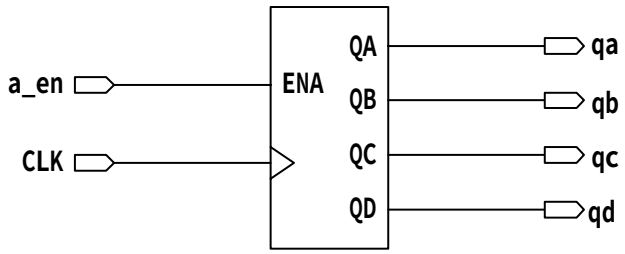
4-4 非同期入力

多くのデザインでは、複数の装置間でデータの送受信を行います。一般に、これらの装置間のクロックが同一クロックで同期されているケースは希です。従って、複数のクロック間の僅かな周波数の違いで、受信側のフリップ・フロップがセットアップ・タイムやホールド・タイム・バイオレーションを起こす場合があります。このような場合、受信側のフリップ・フロップは良く知られるメタステーブルの状態に陥ります。結果として、システムが誤動作します。

フリップ・フロップのデータ入力やクロック・イネーブルに接続されている非同期入力信号は、フリップ・フロップに必要なセットアップ／ホールド・タイム条件を満たさない場合があります。その場合、フリップ・フロップに間違ったデータをセットするようなメタステーブルの状態を引き起こします。結果として、システムの信頼性に深刻な影響を与えます。

図 40 にイネーブル入力(ENA)でコントロールする為に非同期入力を使った2進カウンタを示します。イネーブル信号がカウンタの正常動作に必要なセットアップ／ホールド･タイム･バイオレーションを起こした場合、各カウンタ･ビットはエラーを起こします。この場合、あるビットが反転すべき所が変化せずに現在の状態を保持しているのに対し、他のビットはカウント･アップする。と言った間違ったカウント値を持つ場合があります。さらに場合によっては、あるビットはメタステーブル状態になります。

図 40. 非同期イネーブルを使った2進同期カウンタ



イネーブル信号を同期化する為のフリップ･フロップを追加することで、カウンタのセットアップ･タイムを保証することができます。図 41 に同期化の1方法を示します。この回路の場合は、同期化の為のフリップ･フロップがメタステーブル状態に入っても、次のクロック･エッジ前にステーブルになります。

このような理由から、デバイス内のメタステーブルの問題を回避する為に、**非同期入力信号を2つ以上のフリップ･フロップに供給しないでください。**非同期入力に接続されたフリップ･フロップは、配置されるロジックの位置やフリップ･フロップの特性により、異なる値を持つ場合があります。従って、**1個のフリップ･フロップで受けた後に供給してください。**

図 41. 同期化フリップ･フロップを使った2進同期カウンタ

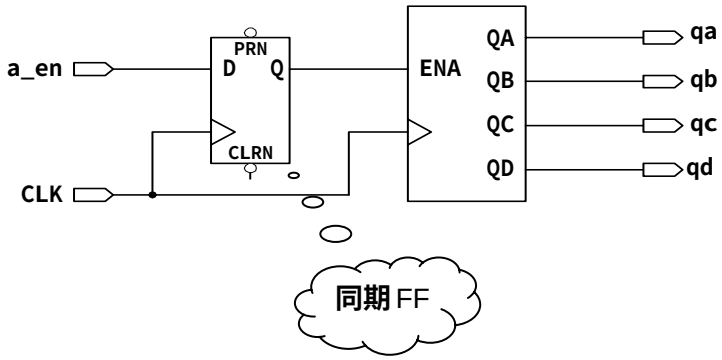


図 40 を図 41 に改善した場合の VHDL の記述例を図 42 に、VerilogHDL の記述例を図 43 に示します。

図 42. 非同期イネーブルを使った2進同期カウンタ(VHDL)

※ 図 40 のVHDL記述例

```
ARCHITECTURE pld OF a_4count IS

BEGIN
  PROCESS (clk, enable)

  VARIABLE cnt : std_logic_vector
                    (3 downto 0);

  BEGIN
    IF (enable = '0') THEN
      cnt := cnt;
    ELSIF rising_edge (clk) THEN
      cnt := cnt + 1;
    END IF;

  q <= cnt;

  END PROCESS;
END pld;
```

改善例

※ 図 41 のVHDL記述例

```
ARCHITECTURE pld OF s_4count IS
SIGNAL intc : std_logic ;

BEGIN
  PROCESS (clk) BEGIN
    IF rising_edge (clk) THEN
      intc <= enable ;
    END IF;
  END PROCESS;

  PROCESS (clk)
  VARIABLE cnt : std_logic_vector
                    (3 downto 0);
  BEGIN
    IF (intc = '0') THEN
      cnt := cnt;
    ELSIF rising_edge (clk) THEN
      cnt := cnt + 1;
    END IF;

  q <= cnt;

  END PROCESS;

END pld;
```

図 43. 非同期イネーブルを使った2進同期カウンタ(VerilogHDL)

※ 図 40 のVerilogHDL記述例

```
module a_4count (clk, enable,q);

input    clk, enable;
output [3:0]q;

reg [3:0]q;
reg enb;

always @(enable)
   if (enable)
      enb = 1'b1;
   else
      enb = 1'b0;

always @(posedge clk)
   if (enb)
        q = q + 1;

endmodule
```

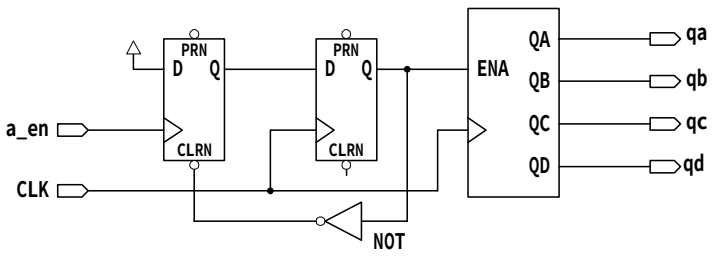
改善例

※ 図 41 のVerilogHDL記述例

```
module s_4count (clk, enable,q);

input    clk, enable;
output [3:0]q;

reg [3:0]q;
reg enb;

always @(posedge clk)
   if (enable)
      enb = 1'b1;
   else
      enb = 1'b0;

always @(posedge clk)
   if (enb)
      q = q + 1;

endmodule
```

図 44 に、非同期入力を同期化する為のもう1つの方法を示します。a-en 入力でフリップ･フロップをセットします。この回路の場合は、非同期入力がクロック周期より短い場合にそのエッジを検出する場合に有効です。

図 44. 2つの同期化フリップ･フロップを使った2進同期カウンタ



非同期入力がステートマシンやカウンタなどのレジスタ･ファンクションの入力として使用される場合は、セットアップ／ホールド･タイム･バイオレーションが発生します。このバイオレーションは、システムとして許容できないステートやメタステーブル状態になったりします。図 45 に非同期入力を使ったステートマシンの例を示します。

図 45. 非同期入力を使ったステートマシン

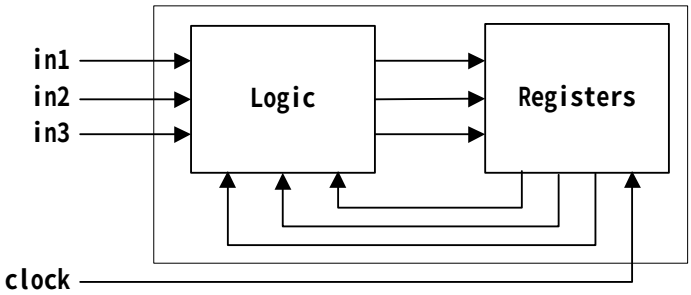
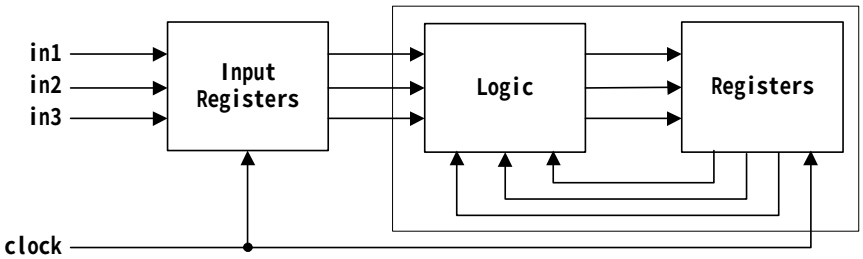


図 46 は、図 45 のステートマシンに入力レジスタを追加し、セットアップ／ホールド･タイムを確実なものにした例です。付加したレジスタがセットアップ／ホールド･バイオレーションによってエラーしても、ステートマシンがメタステーブル状態に入ったり、未定義ステートに入ることは有りません。

図 46. 同期化入力レジスタを付加したステートマシン



4-5 帰還型 SR フリップ･フロップ

ALTERA CPLD/FPGA 内の 2 つロジックセルやプロダクトターム型デバイス(MAX ファミリ)が持つシェアード･エクスパンダをタスキ状に帰還をかけることによって非同期型の SR(セット／リセット)フリップ･フロップを作成することができます。しかし、このフリップ･フロップは、ある条件を満足しない場合に発振現象を起こすことがあります。

☞: **基本的には、ロジックセルやシェアード･エクスパンダによる帰還型の SR フリップ･フロップは使用しないでください。**

帰還型 SR フリップ･フロップのホールド･タイムを確保しない場合は、ある周波数で発振を起こします。元来、フリップ･フロップは、2値を持つ安定した物と考えられていますが、立派な発振回路を製作することができます。図 47 に帰還型 SR フリップ･フロップの例を示します。

図 47. 帰還型SRフリップ･フロップ

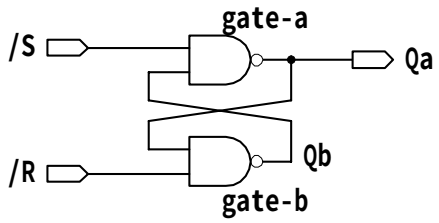


図 47 の回路では、セット入力/Sがローになった場合、その出力Qaは、gate-aのディレイt_{gatea}分遅れて出力されます。その出力がgate-bを通り、そのディレイt_{gateb}分遅れた後にgate-aの入力に到達します。従って、入力/Sに必要なホールド･タイムは、次のようになります。

$$t_H = t_{gatea} + t_{gateb} = 2 \times t_{gate}$$

ここで、入力/S がフリップ･フロップを正しくセットするために必要なホールド･タイムを確保しない場合を考えてみます。図 48 に、ホールド･タイムが短い場合のタイミング･チャートを示します。

図 48. ホールド･タイムを満たさないSRフリップ･フロップ動作

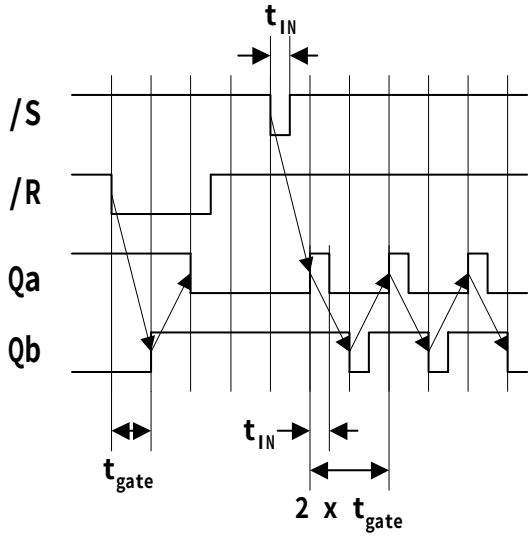


図 48 のように、入力信号/Sのホールド･タイムが $2 \times t_{gate}$より短い場合は、そのパルス幅をハイとし、周期 $2 \times t_{gate}$の発振を起こします。ですから、発振しないはずのフリップ･フロップが見事に発振をしました。同様に入力/Rについても同じことが言えます。

4-6 ディレイ素子回路

ディレイ・チェインとは、故意にディレイや非同期パルスを生成するためにロジックセルやシェアード・エクスパンダを回路内に挿入することです。このディレイは、冒頭にも述べた温度変化、電圧変動、あるいは、製造上のバラツキによって変化します。そのために、度々初期の回路機能を実現できないことが発生します。

図 49～50 にロジックセルやシェアード・エクスパンダを使ったディレイ・チェインの例を示します。図 49 の回路は、セットアップ・タイム、ホールド・タイム、あるいは、クロック対出力の仕様を変えるための例です。この場合のロジックセルやシェアード・エクスパンダのディレイは度々変化しますので、正しく動作しない場合があります。

図 49. ロジックセルとエクスパンダの間違った使い方

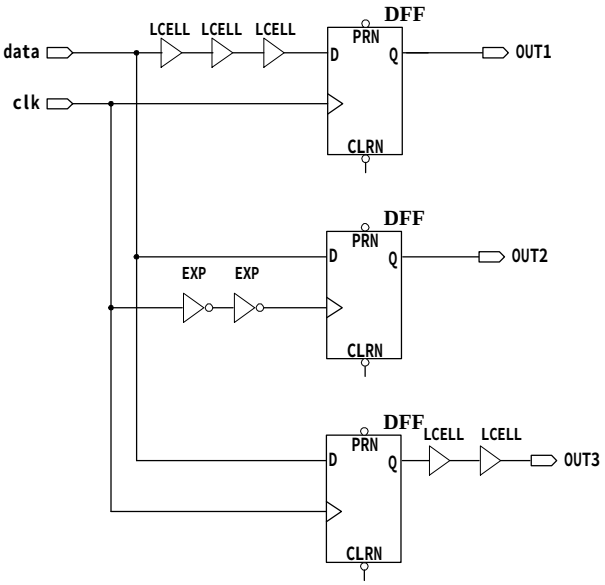


図 50 にシェアード・エクスパンダを使った信頼性の低い非同期パルス発生回路を示します。図 51 は、図 50 を同期回路に変更した例です。

図 50.エクスパンダを使った信頼性の低い非同期パルス発生回路

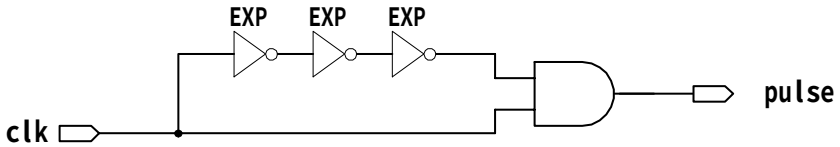
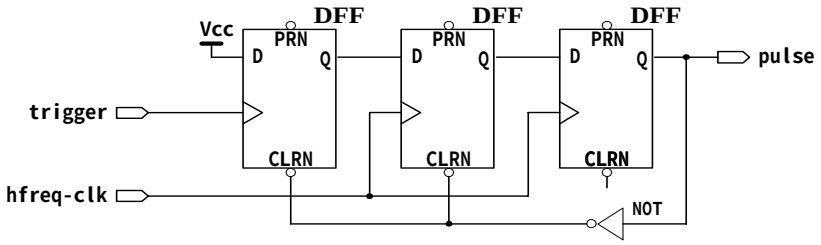


図 51. 同期パルス発生回路



4-7 自己帰還回路

信号が2つ以上のコンビントリアル・ロジックを通って共通回路エレメントに供給されている場合、回路内にレーシング状態が発生します。このような場合は、2つの異なったパス間のスキューによって回路動作に影響します。

最も深刻な例は、フリップ・フロップの出力自体をそのフリップ・フロップ自身のクリア、プリセット、あるいは、クロック・イネーブルに供給する場合です。図 52～53 に説明する例を参考にして正しい回路に変更してください。

図 52 にレーシング状態を持つ非同期パルス・ジェネレータを示します。この回路では、フリップ・フロップが安定して動作する為に必要な最小パルス幅を確保できない場合が有ります。(フリップ・フロップの出力がハイになったら直ちにリセットがかかってしまう為に、その出力パルス幅を確保できなくなります。) この回路の代わりに図 53 で示すパルス・ジェネレータに変更してください。図 53 では、2つのフリップ・フロップと高周波クロックを使ってパルスを発生しています。

図 52. レーシング状態を持つ非同期パルス発生回路

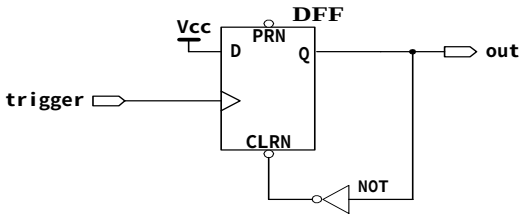


図 53. 同期パルス発生回路

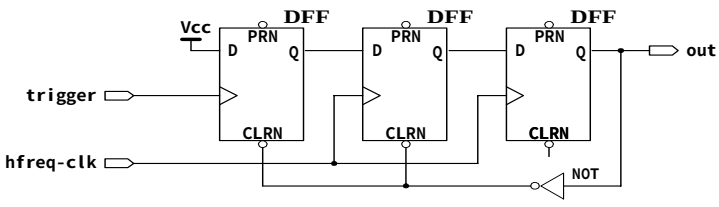


図 52 を図 53 に改善した場合の VHDL の記述例を図 54 に、VerilogHDL の記述例を図 55 に示します。

図 54. 非同期パルス発生回路(VHDL)

※ 図 52 のVHDL記述例

```
ARCHITECTURE pld OF a_clr IS

BEGIN
  PROCESS (trigger, q) BEGIN
    IF (q='1') THEN q <= '0';
    ELSIF rising_edge (trigger) THEN
      q <= '1';
    END IF;
  END PROCESS;

END pld;
```

改善例

※ 図 53 のVHDL記述例

```
ARCHITECTURE pld OF s_clr IS

SIGNAL int0, int1 : std_logic;

BEGIN
  PROCESS (trigger, q) BEGIN
    IF (q = '1') THEN int0 <= '0';
    ELSIF rising_edge (trigger) THEN
      int0 <= '1';
    END IF;
  END PROCESS;

  PROCESS (clk, q) BEGIN
    IF (q = '1') THEN int1 <= '0';
    ELSIF rising_edge (clk) THEN
      int1 <= int0;
    END IF;
  END PROCESS;

  PROCESS (clk) BEGIN
    IF rising_edge (clk) THEN
      q <= int1;
    ELSE q <= q;
    END IF;
  END PROCESS;

END pld;
```

図 55. 非同期パルス発生回路(VerilogHDL)

※ 図 52 のVerilogHDL記述例

```
module a_clr (trigger, q);
input    trigger;
output   q;

reg q;

always @(posedge trigger
         or posedge q)

  if (q)
    q = 1'b0;
  else
    q = 1'b1;

endmodule
```

改善例

※ 図 53 のVerilogHDL記述例

```
module s_clr (trigger, clk, q);
input    trigger, clk;
output   q;

reg int0, int1, q;

  always @(posedge trigger
        or posedge q) begin
    if (q)  int0 = 1'b0;
    else    int0 = 1'b1;
  end

  always @(posedge clk
           or posedge q) begin
    if (q)  int1 = 1'b0;
    else    int1 = int0;
  end

  always @(posedge clk)
    q = int1;

endmodule
```

4-8 On-chip XOR 回路とスタティック・ハザード

ALTERA 社のプロダクトターム形デバイス(MAX ファミリ)は、De-Morgan の定理による負論理演算やアダー回路を構成する目的でハード的な2入力 Exclusive-OR(XOR)回路を内蔵しています。この回路の等価回路を図 56 に示します。

図 56. On-chip XOR等価回路

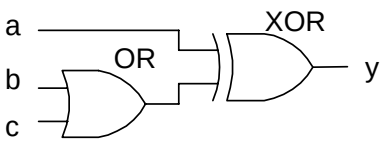
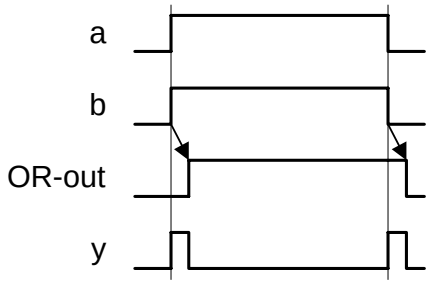


図 56 に示されるように、一方の XOR 回路の入力は、OR 回路を通した後に加えられています。このような回路に同時に入力を加えた場合、ハザードが発生します。その様子を図 57 に示します。

図 57. On-chip XOR回路のハザード波形



b 入力は OR 出力を経た後に XOR 回路に入力されますので、a と b が全く同一信号であっても、OR 回路を構成するトランジスタのスイッチング時間の差でハザードが発生します。このハザードは微少ですが、環境の温度変化によってそのハザードの振幅やパルス幅が変化し、時によってシステムに悪い影響を及ぼす場合があります。このハザードは、XOR 回路を使用した回路では必ず発生します。従って、このハザードを回避するためには、出力にフリップ･フロップを挿入することにより同期化することをお勧めします。

MAX+plus II のオプション Define Synthesis Style 内の XOR Synthesis オプションを OFF でコンパイルすることにより、ハザードを発生しない回路を生成することができますが、この場合の論理圧縮率は低下します。従って、明示的に出力フリップ･フロップの挿入により、ハザード･フリーにすることをお勧めします。

4-9 イリーガル･ステートからの脱出

ステートマシンを設計した場合に、有効なステートで無いステートを持つ場合が有ります。(このステートをイリーガル･ステートと言います。) セットアップ／ホールド･バイオレーションなどによって、イリーガル･ステートに入ってしまう設計では、間違った出力を発生します。また、設計の仕方によっては、このイリーガル･ステートから脱出できなくなる場合があります。この場合は、ステートマシンが無限ループに入りますので、システムが停止する致命的な問題を引き起こします。従って、ステートマシンの設計には、細心の注意を払う必要があります。

ステートマシンがイリーガル･ステートに入る条件は、次のような事柄が考えられます。

① 電源投入時リセット不良 (Power-ONリセット不良)
② セットアップ／ホールド･タイム不良
③ ノイズ

ステートマシンの設計に当たっては、必ず、イリーガル・ステートからの脱出機能を付けなければなりません。この機能を付ける為の最も簡単な手法は、ステートマシンが持つ全てのステートを明示的に定義することです。例えば、n-ビット・ステートマシンでは、2n のステートが存在します。そこで、システムが使用しない全てのステートに対し、明示的にそのステートからの遷移条件を規定します。図 58 に AHDL によるイリーガル・ステートからの脱出機能を持たないデザインを示します。

図 58. イリーガル・ステートからの脱出不能なステートマシン

```
SUBDESIGN il_state (
   clk, go : INPUT ;
   ok      : OUTPUT ;
)

VARIABLE
      ss : MACHINE OF BITS(q[2..0])
   WITH STATES (
      idle,
      one,
      two,
      three,
      four);

BEGIN
   ss.clk = clk ;
CASE ss IS
      WHEN idle =>
   IF go THEN
         ss = one ;
   ELSE
         ss = idle ;
   END IF;
     WHEN one =>
   ss = two ;
     WHEN two =>
   ss = three ;
     WHEN three =>
   ss = four ;
     WHEN four =>
   ss = idle ;
   ok = Vcc ;
END CASE ;

END ;
```

図 58 のステートマシンでは、何らかの理由で idle -> one -> two -> three -> four 以外のステートに入った場合は、そのステートから脱出することは出来ません。図 59 のように明示的にイリーガル・ステートからの脱出条件を記述してください。

図 59. イリーガル・ステートからの脱出機能付きステートマシン

```
SUBDESIGN recover (
   clk, go : INPUT ;
   ok      : OUTPUT ;
)

VARIABLE
      ss : MACHINE OF BITS(q[2..0])
   WITH STATES (
      idle,
      one,
      two,
      three,
      four,
                        illegal1,
                        illegal2,
                        illegal3);
```

全ステートを記述。

```
BEGIN
   ss.clk = clk ;
CASE ss IS
      WHEN idle =>
   IF go THEN
         ss = one ;
   ELSE
         ss = idle;
   END IF;
     WHEN one =>
   ss = two ;
     WHEN two =>
   ss = three ;
     WHEN three =>
   ss = four ;
     WHEN four =>
   ss = idle ;
   ok = Vcc ;
     WHEN OTHERS =>
   ss = idle;
END CASE ;

END ;
```

明示的な記述。

図 59 のように、ステートマシンのステート定義部に取りうる全てのステートを定義します。例では、illegal1,illegal2,illegal3 になります。続いて、CASE ステートメント内にシステムで使用しない全てのステートに対し、明示的に規定します。この例では、OTHERS ステートメントを使って明示的に定義しています。このステートマシンは、何らかの理由で illegal1～illegal3 ステートに入った場合でも必ず次のクロック・サイクルで idle ステートに遷移します。

この手法は、VHDL や VerilogHDL 言語であっても適用されます。

4-10 カウンタとグラウンド・バウンス

一般的にカウンタを構成する場合は、バイナリ・エンコーディング方式を採用します。特に、回路図による設計では、74 ファミリのシンボルを用いて設計を行うことから殆どの場合がバイナリ・エンコーディング方式となります。しかしながら、ご存知のように、バイナリ・エンコーディング方式の場合は、カウンタの桁数が大きくなるにつれて、最上位桁が"0"から"1'に遷移する時点でグラウンド・バウンスが原因でカウンタが誤動作する場合があります。その理由は、下位桁の全てのビットがハイ"1"の状態で出力に接続されている負荷容量（浮遊容量も含む。）に充電が行われています。この状態で、次のクロック・サイクルで最上位桁が"0"から"1"に反転し、全ての下位桁が"1"から"0"に変化します。この時、負荷容量に充電されていた電荷が PLD の"ON"トランジスタを通してグラウンドに放電されます。この時形成される回路網の全インダクタンス成分(L)と放電電流($\Delta i / \Delta t$)の積が起電力となってグラウンド上に現れます。ここで誘起される電圧振幅は、1v を超えることも稀では有りません。結果として、そのカウンタ自身が誤動作したり、負荷回路に悪い影響を与えます。

グラウンド･バウンスを回避する為には、一般的にグレイ･コード･カウンタやワンホット･カウンタを使って、同時に遷移するカウンタ･ビットを最小限に押さえます。しかしながら、グレイ･コード･カウンタの場合は、カウンタ出力をデコードする場合に扱いが複雑ですし、ワンホット･カウンタの場合は、フリップ･フロップを多用したり、不正状態からの復帰が難しくなります。(ALTERA 社の MAX+plusⅡ／QuartusⅡを使用してステートマシンを構成する場合、MAX 系のデバイスではバイナリ･カウンタを、FLEX 系デバイスではワンホット･カウンタを自動的に生成します。マニュアルで設定することもできます。)

ここで、最も設計者が親しんでいるバイナリ･カウンタを使用する上でのグラウンド･バウンス対策用の設計指針を簡単に列記します。

① 多層 PWB を使用し、電源とグラウンドを面パターンとする。
② デカップリング･コンデンサは、0.2μF(積層セラミック等)を接続する。
③ インダクタンス成分が高いソケット実装を避ける。
④ PWB 上のパターン長をできるだけ短くする。
⑤ ファンアウトの大きい回路は、分割ドライブをし1ピンあたりの負荷容量とパターン長を短くする。
⑥ 出力側に 10～30Ωの直列抵抗を挿入する。
⑦ スルーレイト･コントロール･オプションを持つデバイスでは、このオプション使用してΔi/Δtを小さくする。
⑧ コモン･バスにおける終端は、プルアップ方式からプルダウン方式に変更し、バスがハイ･レベルの時の負荷容量に対する充電を可能な限り小さくする。
⑨ プログラマブル･グラウンドの機能を持つデバイスでは、8 ビット毎を基本にプログラマブル･グラウンドを挿入する。

4-11 超安定回路の設計のおわりに

ここに記載した各種の設計手法に基づいて設計を行うことにより、ALTERA 社 CPLD/FPGA を使う上で、信頼性の高いデザインを行うことができます。これらのデザイン手法により、多くの外乱から ALTERA 社の CPLD/FPGA を守ることができます。

ALTERA 社の MAX+plusⅡでは、ここに掲げた各種の設計手法を使って正しくデザインされているかをチェックする機能を持っています。それをデザイン･ドクター(Design Doctor)と言っています。デザインをコンパイルする場合には、このデザイン･ドクター機能を ON にするだけで、デザイン内に潜む不安定要素を自動的に抽出してくれます。そして、その抽出されたメッセージとこの資料を参考にデザインを正しく修正することをお勧めします。また、デザイン･ドクターとここに掲げた各種の設計手法は、MAX+plusⅡ内のオンライン･ヘルプでも参照することができます。

その他にシステムを不安定にする要素について説明します。

5 最悪値タイミング設計

今日、CPLD/FPGAの低価格化と大規模化に伴いこれらのデバイスがASICの代替として使われ始めています。しかしながら、ASIC ユーザが CPLD/FPGA の使用を検討する段階で一つの疑問に遭遇することがあります。それは、CPLD/FPGA の世界ではデータ・ブック内のタイミング・パラメータ表に最悪値である最大値や最小値しか見つけることができないことです。一般に汎用ロジック・デバイスを使用して設計をする場合には、これらの値以外に公称値や代表値を使用して設計をして来たからです。このように、最悪値しか記載されていないデバイスを使用した場合には、次のような疑問が生じます。

1) 如何なる条件下（電源電圧の変動、季節変動や極寒地などへの設置場所に依存する周囲温度の変化、あるいは、製品のバラツキ等）でも正しく動作するのか？
2) なぜ、代表値や平均値を使用しないのか？
3) シフト・レジスタにおいて、なぜ、同相転送が正しく行われるのか？

これらの疑問にお答えするために本資料を用意しました。

5-1 最悪値のおさらい

ALTERA 社のデータ・ブックのタイミング・パラメータ表には、最大値、あるいは、最小値しか記載されていません。この値は、ALTERA 社が推奨する条件下であれば保証されるものです。ALTERA 社の推奨する条件とは、次のことを指します。

1) 使用電圧範囲
2) 使用周囲温度範囲
3) 製造上のバラツキ

これらを考慮して、データ・ブック上の最悪値を保証しています。

ALTERA 社の CPLD/FPGA 開発ソフトウェア MAX+plusII／QuartusIIのタイミング・シミュレーションやタイミング・アナリシスをする場合のタイミング・パラメータは、データ・ブックに記載されている最悪値が使用されています。従って、MAX+plusII／QuartusIIからの算出結果は保証された値となります。例えば、タイミング・アナライザから得られた動作周波数はその CPLD/FPGA の最も遅いパスの最低動作周波数を示しますので、その周波数以下で使用されるシステムの動作周波数は確実に確保できるわけです。同様にして、ディレイ・マトリックスやセットアップ／ホールド・タイム・アナリシスから得られた値は、ALTERA 社の推奨条件下では保証された値となる分けです。

このような理由から、ALTERA 社の CPLD/FPGA を使用する場合は、実デバイス内に配置配線が完了した後に、タイミング・シミュレータとタイミング・アナライザによる綿密な動作検証を行うことをお勧めします。

5-2 最悪値設計

しかしながら、実際のデバイスは全ての条件下で最悪値のタイミング・パラメータを持つ分ではありません。また、購入したデバイスでは、最悪値は保証されていますが、実際の値がその値に対してどの位かけ離れているのかは分かりません。しかし、どのような場合であっても、システム上で安定した動作をさせる必要があります。そこで、最悪値設計（最悪値タイミング設計）において、どのようにして動作が保証されているかを説明します。

それでは、シフト・レジスタを例にとって考えて見ましょう。シフト・レジスタは、データとクロックのタイミングによって同一クロックの立ち上がりでデータ転送（同相転送）が正しく行われない場合があります。

- 同相転送とは、シフト・レジスタの各レジスタに同位相クロックを使用し、各ステージの出力を次段に伝搬させる方法です。

- 逆相転送とは、反相転送とも言い、シフト・レジスタの奇数段と偶数段に相互に逆位相のクロックを使用し、各ステージの出力を次段に伝搬させる方法です。クロックとデータが競争し、前段シフト・レジスタの出力が確定する前に、次のクロックの立ち上がりが来てしまうようなレーシングを防ぐ場合に有効な方法です。但し、デューティ・サイクルには注意が必要です。

図 60 は、同相転送方式の典型的なシフト・レジスタのブロック図です。

図 60. 同相転送のシフト・レジスタ

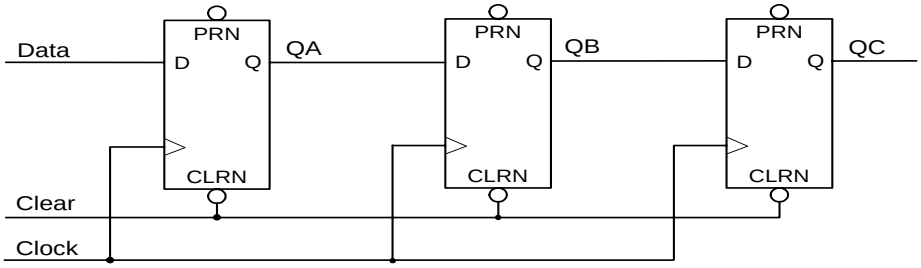


図 60 において、使用されるクロック信号(Clock)は、CPLD/FPGA が持つグローバル・クロックを使用します。

グローバル・クロックとは、クロック専用入力端子から入力されたクロックを言います。このクロックは、デバイス内にクロック専用線を持ち、デバイス全体にわたって遅延とそのバラツキ（スキュー）が最も小さくなるように分配されています。さらに、各フリップ・フロップ間では、均一のクロック・スキューになるように調整されています。図 61 に CPLD/FPGA 内に実現されたシフト・レジスタのブロック図を示します。

図 61. CPLD/FPGA内に実現されたシフト･レジスタ

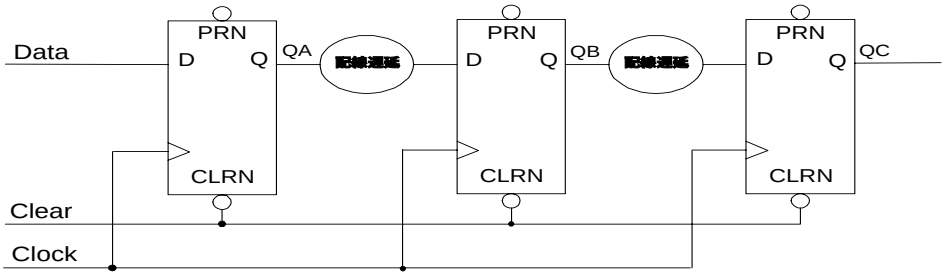


図 61 に於ける配線遅延は、CPLD/FPGA 内のインターコネクトによる遅延を意味しています。ALTERA 社のインターコネクト方式には、2つの方式があります。

- PIA(Programmable Interconnect Array)：ALTERA 社の Classic, MAX の各 Family で採用されているインターコネクト方式で、二次元的に配置された配線用アレイです。非常に高速にできています。

- FastTrack(MultiTrack) Interconnect：ALTERA 社の MAX 9000, MAX Ⅱ, FLEX, APEX, ACEX, Stratix の各 Family で採用されているインターコネクト方式で、三次元的に配置された高速インターコネクト･アレイです。大規模デバイスであっても、非常に柔軟、かつ、高速に配線することができます。

参考: FastTrack(MultiTrack)には、三次元的に3つの配線インターコネクトがあります。

① Row FastTrack (X軸)
② Column FastTrack (Y軸)
③ Local Interconnect (Z軸)

それでは、具体的に数式を使用して解説をして行きましょう。図 61 に於ける2つのレジスタ間のデータ転送を正しく行うために、後段のレジスタに於いてセットアップ･タイム(tSU)とホールド･タイム(tH)を考慮する必要があります。

ここでは、最もインターコネクト遅延の小さい MAX7000 シリーズを例にとって説明します。図 62 に2つのレジスタ間のタイミング･モデルを示します。

図 62. MAX7000 のタイミング・モデル

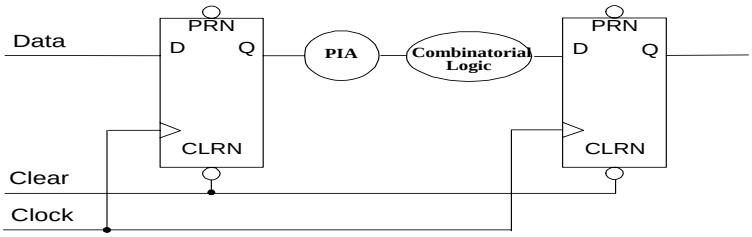


図 62 におけるセットアップ・タイム(tSU)とホールド・タイム(tH)は、各々次式で与えられます。

$$tSU = t_{RD} + t_{PIA} + t_{LAD} + t_{SU} \quad \cdots ①$$
$$tH = t_{RD} + t_{PIA} + t_{LAD} + t_{H} \quad \cdots ②$$

Where, tSU ： 後段レジスタが必要とする総合セットアップ・タイム
tH ： 後段レジスタが必要とする総合ホールド・タイム
t_{RD} ： レジスタ出力ディレイ
t_{PIA} ： インターコネクト (PIA) ディレイ
t_{LAD} ： コンビナトリアル・ディレイ
t_{SU} ： チップ内ハード・マクロ (フリップ・フロップ) が必要とするセットアップ・タイム
t_{H} ： チップ内ハード・マクロ (フリップ・フロップ) が必要とするホールド・タイム

ここで、①式と②式に各値を代入すると、次の解を得ることができます。

$tSU = 1 + 2 + 6 + 4 = 13$ nS
$tH = 1 + 2 + 6 + 4 = 13$ nS

Where $t_{RD} = 1$nS, $t_{PIA} = 2$nS, $t_{LAD} = 6$nS,
$t_{SU} = 4$nS, $t_{H} = 4$nS

従って、後段のレジスタが正しく前段のデータをセットするためには、セットアップ、ホールド・タイム共に 13nS 以上を確保すれば良いことになります。

話は変わって、上記のような値を持つCPLD/FPGAがなぜ、同相転送シフト・レジスタで正しく動作するのかを考えて見ます。各CPLD/FPGAファミリ毎のグローバル・クロックの最大遅延とデバイス内に於けるスキューのバラツキは、ALTERA社の最も大きなデバイスであっても数 100pS (注) 以下と言われていますので、セットアップ／ホールド・タイムに比べてグローバル・クロックのスキューのバラツキを無視して考えることができます。

注 ここにあげたスキューのバラツキは参考値であり、メーカとして保証していません。しかしながら、ALTERA 社の CPLD/FPGA では、デバイスの全てのフリップ・フロップに対し、クロック・スキューが均一になるようにディレイ調整をしていますので、個々のフリップ・フロップに於けるグローバル・クロックのスキューは無視することができます。

ここで、グローバル・クロックを使用した同相転送について考えて見ましょう。クロック・スキューを無視すると、2つのレジスタには、全く同時にクロックが供給されると考えられますので、前段のレジスタからの出力(t_{RD})は1つ前のクロックでセットされた後、PIA(t_{PIA})を通り次段のコンビナトリアル・ロジック(t_{LAD})を通過して 9nS後に後段のレジスタ入力に到達します（ホールド・タイムの保証）。同様に、次のクロックによって変化した前段レジスタ出力も 9nS後に次段レジスタの入力に到達します（セットアップ・タイムの保証）。このようにして、同一クロックでトリガされたデータが次段のレジスタで有効になる前に有効クロック期間が終了していますので、同相転送が保証される分です。従って、このCPLDを使用する場合のグローバル・クロックの最高周波数は、1／tSU = 1/13nSで求められますので 77MHzとなります。この周波数以下で使用する場合は、動作が保証されます。

しかしながら、もし仮に、CPLD 内で非同期クロック（アレイ・クロックとも言う）を使用した場合は、クロック自身が論理ブロックとインターコネクトを介して各レジスタに分配されます。この場合は、各レジスタ毎にクロック・スキューが発生し、しかも値をコントロールすることもできません。特に、セグメント方式を持つデバイスでは、配置されるロジックセルの位置によってインターコネクト・ディレイが異なりますのでこのことが顕著になります。このようなシステムでは、前段のレジスタ出力が後段レジスタ入力に到達した後に同一タイミングのクロックがやってくる。と言ったことが起きます。結果として、システム的には同一クロックで一段飛び越したデータ転送が行われることになりますので、誤動作することになります。業界では、このような現象を『すっぽ抜け』と言っています。この事故を防ぐ為にも同期設計が必要です。同期設計(グローバル・クロックを使用した設計)を行うことによって、インターコネクト・ディレイに比べて無視可能なスキューで全てのフリップ・フロップを駆動できますので、信頼性の高いシステムを構築することができます。

多少曖昧な言い方になりますが、現在の ALTERA 社を含めた CPLD/FPGA ベンダの製品は、グローバル・クロックのスキューがインターコネクト・ディレイに比べて無視できる大きさであることを理由に、明言は避けながらも最悪値表現のみでデータの規定を行っています。しかしながら、高速化の一途をたどるデバイスは、今後、最小値タイミングをも規定せざるを得ない時代を迎えています。その為に、一般的なロジック・デバイスのように min, typ, max 表現に向けて研究が進められています。

5-3 同期設計

CPLD/FPGA が大規模になるにつれて、高速性を維持しながらも柔軟性を確保するために各ベンダ毎にインターコネクトの方式が提案されています。しかしながら、デバイス全体を通じてあらゆるデバイス間のスキューが均一になるように配置することは現実として不可能です。このような条件の下に、非同期設計を行うとクロック信号とデータ信号との間でレーシングが発生します。時には、データがクロックを追い越してしまい『すっぽ抜け』が発生します。

このようなことを防ぐ為にも、同期設計が必要となります。同期設計を行う限り、デバイス内でクロック・スキューが最小になるように調整されていますので、『すっぽ抜け』を心配する必要は有りません。唯一注意することは、データ・ラインのセットアップ・タイムとホールド・タイムを確保することだけです。

今までに述べた理由で、CPLD/FPGA ベンダの技術者は、『同期設計』『同期設計』.....と口癖のように言う分です。

5-4 非同期クロックでシフト･レジスタは正しく動作するか？

CPLD/FPGA 内でロジック部を通して生成されたクロックを非同期クロック（アレイ･クロックとも言う）と言います。ピンから入力されたクロックで有っても、クロック専用ピンであるグローバル･クロック･ピンを使用しない限り非同期クロックと見なされます。クロック専用ピン以外から入力されたクロック信号（クロック信号では無く単なる信号と見なされる。）は、必ず、論理ブロックとインターコネクト部を通して各フリップ･フロップに分配されます。従って、それらのクロック･スキューはロジックセルの位置によって異なります。さらに、一度シミュレーションによって確定されたタイミングであっても、設計変更などによってロジックセルの位置が変わると問題になる場合があります。因みに、非同期クロックを使用したシフト･レジスタであってもクロック･スキューとそのクロックとデータ･ラインとの間のセットアップ･タイムとホールド･タイムを十分考慮した設計であれば正しく動作します。しかし、このような設計は、実験装置における採用であれば大きな問題は起きませんが、実機搭載にはお勧めできません。一般的に、実機の動作環境は、電源電圧や周囲温度も度々変動しますし、多くの生産ロットを繰り返す度に購入するデバイスの特性が変化します。結果として、『時々誤動作する。』『夏は動かない。』『冬は動かない。』と言った事故に遭遇します。これらの事故は、殆どの場合がタイミング･パラメータのバラツキによって発生するものです。

このような不安定な要素を回避する上でも、動作が保証された同期設計を強くお勧めします。

5-5 非同期設計とシミュレーション

前項で述べたように、各レジスタ間において、セットアップ･タイムとホールド･タイムが保証される限り、CPLD/FPGA は同期設計で無くても正しく動作します。従って、MAX+plus II／Quartus II のタイミング･シミュレータとタイミング･アナライザを使用して、各ノード間（レジスタ間）のタイミングを厳密に解析してください。しかしながら、この場合のタイミング･マージンは十分な余裕を持つ必要があります。即ち、使用する環境条件（電圧変動や周囲動作温度）やデバイスのバラツキを考慮する必要があります。低消費電力化の目的で非同期回路を使うことが見受けられますが、各種の環境変数を考慮して余裕を持った仕様でお使いいただくことをお勧めします。非同期回路の利用は、数 MHz 位の動作周波数が限度かも知れません。多くのシステムでは、勿論低消費電力化は永遠のテーマですが、“機器の信頼性を取る”か“低消費電力を取る”かは経済性を含めてトレードオフの関係を持ちます。

エルセナでは、信頼性の高い同期設計を強くお勧めします。

5-6 クロックのデューティ比とシミュレーソン

最近のデバイスでは、デバイスの内部でクロックの位相を反転させるオプションが付いています。この信号をグローバル･クロックとして使用する場合も、最も少ないスキューで正相･逆相のクロックを得ることができます。得られた正相･逆相クロックともデューティ比が忠実に再現されます。

ここでは、入力されたグローバル･クロックのデューティ比が 50%で無い場合やアレイ･クロックを使用した場合でデューティ比が異なる場合を考えてみます。ALTERA 社の MAX+plusⅡ／QuartusⅡでは、正相･逆相クロックを考慮してシミュレーションを行うことができます。しかしながら、この場合のデューティ比は、50%を想定しています。従って、実際に入力されるクロックのデューティ比が 50%で無い場合は、正しくシミュレーションを行う為に、その比率に合わせてデューティ比を変えてやる必要があります。これによって、非対象デューティ比クロックに於けるシミュレーションも正しく行うことができます。

5-7 最大値と最小値の落とし穴

各CPLD／FPGAベンダのデバイスには、各種のスピード･グレードがあります。このスピード･グレードとタイミング･パラメータの最大値、あるいは、最小値との間にどのような関連があるのでしょうか？

例えば、入力／出力ピン間ディレイに 10nS、15nS、20nS のデバイスが有ったとします。この場合の常識的な理解としては、20nS 品であれば、ピン間ディレイの最大値は 15nS～20nS の間にあり、15nS 品であれば、10nS～15nS にあると考えます。しかしながら、CPLD/FPGA の世界では必ずしもこの仮説が成り立ちません。タイミング･パラメータ上で最小値と最大値が同時に定義されている分ではありませんので、20nS 品であれば 20nS を超えなければ全て条件を満たすことになります。結果として、実力的には 10nS 品や 15nS 品であっても、20nS 品として出荷される場合があります。(勿論、マーキングは 20nS 品のまま) これは、デバイス･ベンダの生産管理や在庫管理に依存します。特別にスピード･グレードを分けて生産せずに、あるデバイスを一括生産し、必要な注文数に応じて必要なスピード･グレードのものを選別すればよい分けです。こうすることにより、柔軟に生産調整をすることが出来ます。また、微細化することによってスピード性能が向上しても何ら古い製造プロセスのものと区別する必要がありません。

このような背景から、データ･シートに記載されている保証データと実デバイスが持つ実力データとの間には開きがあります。そして、これらのデータのバラツキが回路上のグリッチや信号のレーシング等の現象となって現れます。そして、システムを不安定な領域に引き込んで行きます。奈落の底へとです。CPLD/FPGA 内でロジックセルをディレイ素子として使用したタイミング調整回路は、最も危険と言わざるをえません。

5-8 最悪値タイミング設計のおわりに

CPLD/FPGA で最悪値のみでタイミングを定義している理由は、デバイス全体にわたるクロックのスキューがインターコネクトのディレイに比べて無視できる値であることを利用しているからに他なりません。しかしながら、今後、微細化のためのダイ･シュリンクや配線層の多層化等の技術進歩により、インターコネクトによる遅延がクロック･スキューの値に近接してくると、クロック･スキューを無視して考える分にはいきません。現在、ASIC 設計で行われているような各信号間のスキューの調整が必要となります。

従って、将来にわたってシステムを安定して生産･動作させるためにも;

ディジタル回路において、『同期設計』に勝る方法はありません。

6 非同期信号のシステムへの影響

極、当たり前に設計をして、極、当たり前に評価をして装置を出荷したにも係わらず、他の装置と接続して動作をさせていたら、『時々誤動作する』とか、夏場の暑い時に『誤動作する』とか言ったことを聞くことがあります。これらの原因は、タイミングに起因することが殆どです。

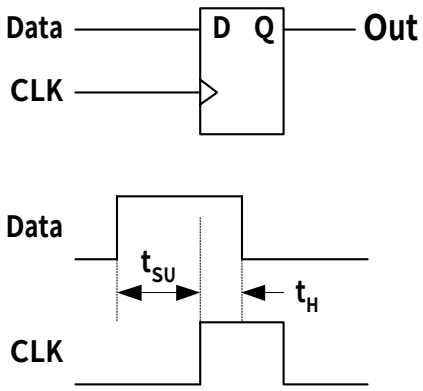
これらは、システムに同期をしていない入力信号（非同期入力信号）や回路上で発生するハザードに起因しています。そこで、この資料では、システムに同期していない入力信号がシステムに与える影響とその解決方法について説明し、ハザードがシステムに与える影響に関しては後述します。

6-1 非同期入力

6-1-1 非同期入力の持つ意味

フリップ･フロップの入力信号とクロック信号との間には、安定して動作するための要求条件としてセットアップ･タイムとホールド･タイムの規定があります。回路設計者はこれらの規定を厳守しなければなりません。この要求条件は、各々のデバイス･ベンダから提供されるデバイス･データ･シートの中に規定されています。図 63 に概念図を示します。

図 63. セットアップ／ホールド･タイム



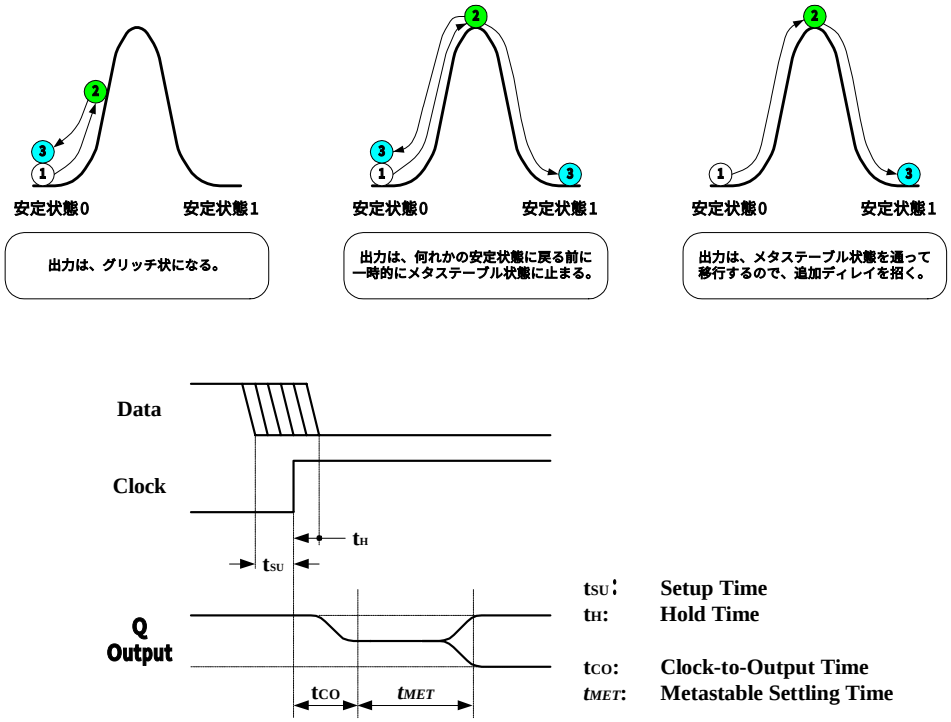
このセットアップ／ホールド･タイム規定に違反した場合には、メタステーブルと呼ばれる一時的に不安定な状態やシステム上定義された状態と異なる状態になる場合があります。このことが要因でシステムの信頼性に大きく影響を及ぼします。異なる機器間でデータを授受するような場合は、非同期システムとなり、受け取る信号とシステム･クロック間では、このセットアップ･タイムとホールド･タイムの規定を保証することができません。これらの装置間での信号の授受時には、信号の同期化が必要となりこの同期化が行われない場合メタステーブル状態を引き起こします。

6-1-2 メタステーブルとは

メタステーブルとは、本来規定されているクロックからの出力時間 tco を超えても出力状態が安定しない状態を言います。安定するまでに必要な時間は、周囲条件やデバイスの製造技術に依存します。

図 64 にメタステーブルの 3 つの状態を示します。

図 64. メタステーブルの 3 つの状態



レジスタの動作は、フリップ･フロップのデータ入力が規定のセットアップ･タイムとホールド･タイムを持つ限り、出力は如何なる追加ディレイも無しに、正しく入力を保持します。しかし、フリップ･フロップのデータ入力がセットアップ･タイム、あるいは、ホールド･タイムの何れかを保証しなかった場合は、フリップ･フロップは限界すれすれにトリガされ、出力は規定時間内に 2 つの安定状態の何れかに直ぐには確定しない場合があります。この限界トリガは、グリッチ出力やハイ･レベルとロー･レベルの間でメタステーブル状態に一時的に止まったり、あるいは、クロックの立上りから出力が安定するまでに余分な時間がかかったりします。

図 64 の左の図は、十分トリガされずに元の状態に戻ってしまう状態を表します。真中の図は、結果として安定状態に落ち着きますが、再び元の状態に戻るか、あるいは、他の状態に遷移します。しかしながら、その状態に落ち着くまでに規定値以上に時間がかかります。右の図は、他の状態に移行しますが、メタステーブルの状態を通過しますので、規定のt_{CO}を守ることはできません。図 64 に示されるタイミング図内のt_{MET}は、セトリング･タイムと呼ばれ、本来の規定のクロックから出力が安定するまでの時間(t_{CO})を超えて出力が確定するまでの時間を言います。

ここで、メタステーブルはその状態が発生する確率関数として表され平均発生間隔時間となります。

$$MTBF = \left[f_{CLOCK} \times f_{DATA} \times C_1 \times e^{(-C_2 \times t_{MET})^{-1}} \right] - 1$$

Where: f_{CLOCK}: クロック周波数

f_{DATA}: 入力データの周波数

t_{MET}: フリップ･フロップが安定するまでの時間

C_1,C_2: デバイスの製造技術によって決まる定数

上式からも分かるように、周波数が高くなると発生し易くなります。

図 65 は、デバイス･メーカから示されるメタステーブルの特性の一例です。

図 65. メタステーブルと製造係数C_1,C_2とt_{MET}の関係

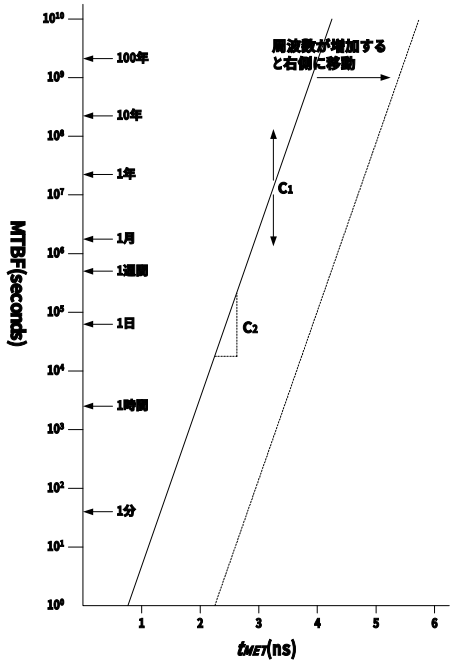


図 65 でも分かるように、C_1が大きくなるとメタステーブルの発生確率が下がり、C_2が大きくなるとt_{MET}が小さくなります。また、動作周波数が増すとカーブが右方向に押しやられ、メタステーブルの発生確率が高くなります。

メタスタビリティの影響を減らすための最も一般的な方法は、次の 2 つの方法があります。

1) 同期用フリップ･フロップを使用する
2) FIFO バッファーを使用する

6-2 メタスタビリティのシステムへの影響

先にも述べたとおり、非同期入力はシステムに悪影響を及ぼします。基本的に、システム内のフリップ･フロップが一度メタステーブルの状態に陥りますと、単に不安定となるばかりでなく最悪の場合はシステム･ハングアップの状態となります。最早、電源を切る以外に復帰の方法はありません。このように、メタスタビリティの状態は、システム上からは人為的に状態定義ができません。半導体の物性のみが状態を制御できます。“神のみぞ知る”です。

特に陥り易い例は、同一クロック系システムの場合です。例えば、相互の機器が 50MHz のマスタ･オシレータで作動している機器同士での信号の授受です。このように公称同一周波数であっても、2 つのマスタ･オシレータには周波数偏差や温度係数等も異なっており、微妙に周波数は変動します。結果として、信号が相互に非同期となります。ご注意ください。

6-3 メタスタビリティの回避

6-3-1 同期用フリップ･フロップの使用

単一信号を同期化する場合には、非同期入力信号とシステム･クロックとを同期化するために、その信号と後段ロジックとの間に同期化フリップ･フロップを複数挿入します。挿入されるフリップ･フロップの数によってシステムの信頼性は格段に向上しますが、挿入したフリップ･フロップの段数分だけシステムのレイテンシが増しますので、全体システムでのパフォーマンスの低下を引起します。

非同期入力信号の繰り返し周波数とシステム･クロックの周波数との差が無視できる程にある場合は 1 段程度で十分ですが、2 つの周波数が隣接している場合は数段入れる場合もあります。

図 66 に例を示します。

図 66. 1 段のフリップ･フロップを使用したメタステーブル回避回路

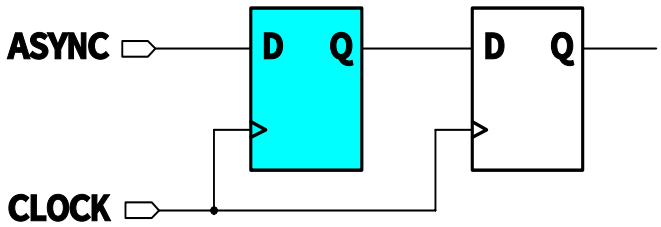


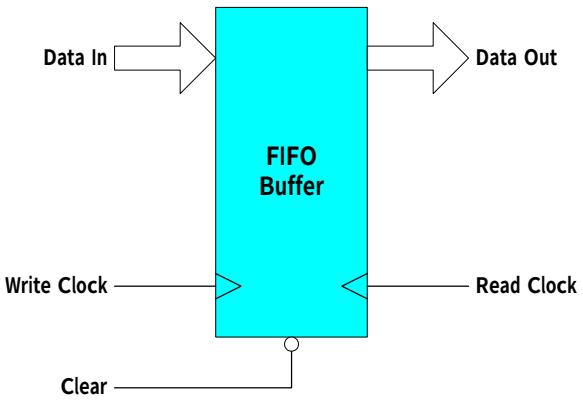
図 66 において、同期化フリップ･フロップ (前段のフリップ･フロップ) がメタステーブル出力を生成しても、その後、メタステーブル信号は 2 番目のフリップ･フロップがトリガされる前に安定する可能性が出てきます。この方法は、2 番目のフリップ･フロップが不安定出力をトリガしないと言う保証は有りませんが、そのデータが回路の他の部分に到達する前に有効な状態になる確率は向上します。このようにして、システムに要求される信頼性から挿入するフリップ･フロップの段数を決定して行きます。

何れにしても、1 本の非同期入力は、複数のフリップ･フロップに供給しないでください。1 本の非同期入力を複数のフリップ･フロップに供給すると、メタステーブルの状態によるシステム･エラーの確率が増加します。なぜならば、非同期入力信号が複数のフリップ･フロップに入力された場合、接続されたフリップ･フロップ毎に異なる状態になる可能性があり、システム上一義的な状態定義ができなくなります。このような場合は、同期化フリップ･フロップを入れ、システム上一義的な状態定義をした後の出力を後段フリップ･フロップやロジックに供給してください。

6-3-2 FIFO バッファーの使用

データ･バスなどの複数ビットの入力を同時に同期化するてるためには、FIFO(First-In First-Out)バッファーを使用します。図 67 にその概略図を示します。

図 67. FIFOバッファーを使用したメタステーブル回避回路



6-4 非同期信号のシステムへの影響のおわりに

システムに非同期の信号が入力されますと、メタステーブルと呼ばれるシステム上予測不可能な状態に陥ります。最悪の場合、システムがハングアップし電源を切断しないと復帰しない事態を引起す場合があります。システム全体で同期化されていない機器間での信号の授受の場合には、メタステーブル状態を引起す可能性があります。従って、システム全体の信号系統を精査し、同期化システムを構築し、不安定要素を取り除くことをお勧めします。

7 ハザード信号のシステムへの影響

電気的に正規化されたロジック回路は、カスタム LSI、スタンダード・セル、ASIC、CPLD/FPGA、古くは、TTL と言ったいろんな実現手段で 構築されてきました。アナログ信号に比べロジック信号は正規化されていますので、考え方が非常にシンプルです。そのシンプルさが現在の文明を飛躍的に向上させました。自然界の殆どの物理現象がディジタル化され、さらにはコンピュータ化される時代になって来たといっても過言ではありません。

多くの便利さが提供される中で、実際に半導体回路を使ってシステム化される段階では、マクロ的にはディジタル回路（ロジック回路）として正規化された考え方をすることができますが、ミクロ的にはアナログ信号となります。特に、最近の半導体デバイスは高速化され、ロジック回路として単純な“1”と“0”の世界だけで閉じて考えられなくなって来ています。例えば、ギガ・ヘルツ(GHz)帯の高速伝送や低消費電力化のために取り扱い信号の微弱化等が挙げられます。

このように単純な“1”と“0”の世界だけで閉じられない世界に、古くから“ハザード”と呼ばれる期待以外の信号が本来のロジック信号の中に含まれており、ロジック回路設計者は度々も悩まされて来ました。

そこで、この資料では、何故“ハザード”信号が発生するかの簡単なメカニズムとその“ハザード”信号がシステムに与える影響について簡単に説明します。

7-1 ハザード信号

7-1-1 ハザード信号とは

ハザードとは、1 つ以上の入力が変化した時、出力が正規の値と異なる値を出すような回路の誤動作をハザード（Hazard）と言います。ハザードには、Dynamic Hazard と Static Hazard が有ります。

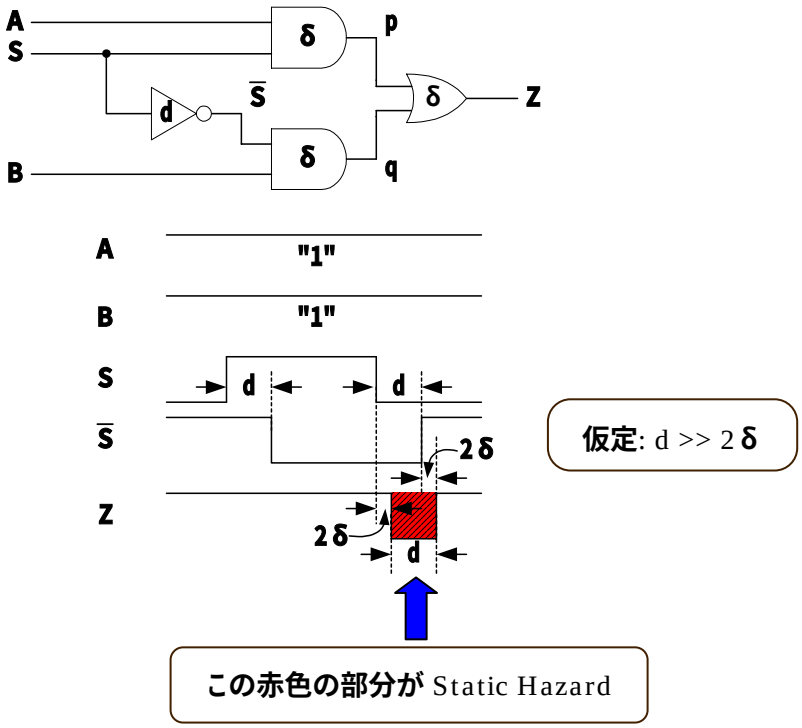
- Dynamic Hazard とは、1 個以上の入力の変化によって出力が変化する時、3 回以上変化して過渡的な値の系列を発生させる現象を言います。例えば、“1”から“0”に遷移する場合に：“1” → “0” → “1” → “0” と変化する場合です。また、“0”から“1”に変わる場合に：“0” → “1” → “0” → “1” と変わる場合です。

- Static Hazard とは、1 個以上の入力が変化した時、同一値をとるべき出力が正しい値と異なる値を一時的に取ることを言います。この Static Hazard には、2 つのケースが有ります。

  1 Hazard：“1”出力が一時的に“0”になる誤り
  0 Hazard：“0”出力が一時的に“1”になる誤り

ここでは Static Hazard について説明します。図 68 に良く使われる信号切換回路(MUX)を示します。

図 68. Static Hazardを発生するMUX回路



注　図 68 の例では、説明を理解し易くするために切換信号用のインバータのディレイ(d)を他の AND-OR ゲートのディレイ(δ)よりも非常に大きいものとしています。

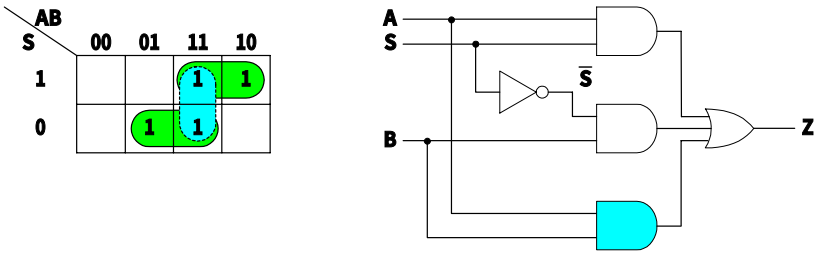
この例で考えますと、例えば、入力 A と B が“1”の時、切換信号 S を切換えても出力には何も発生しない筈ですが、仮に、/S を作るインバータのディレイ(d)がゲートのディレイ(δ)に比べて十分に大きい場合、信号 S が変化しても/S が変化するまでにはかなりの時間のずれが生じます。そうした場合に、S が “1” → “0”に変わって p 点出力が“0”になっても、/S 信号はまだ“1”になっていませんので、結果として q 点出力も“0”のままです。従って、最終出力 Z 点には“0”が出力されます。この負のパルス(ディレイ d に相当)が Static Hazard です。このハザードは、複数の信号のタイミングのずれによって発生します。このタイミングのずれは、後でも述べますが、一義的に決まるものではなく、色々な環境によって影響を受けます。今、この瞬間に発生していなくても何時なんどきに姿を見せるか分かります。従って、明示的にその発生原因の抑制をした設計をする必要があります。

7-1-2 ハザード信号の抑制

Static Hazard が発生する原因は、2 つ以上の入力の変化に時間差がある場合に発生します。従って、Static Hazard を無くすためには、1 個の入力のみが変化するようにすれば良くなります。一般的に、Static Hazard を回避する方法として、冗長回路の追加が有ります。この適切な冗長回路を探すためにカルノー図を使います。

図 69 に、カルノー図と図 68 に冗長回路を付加して Static Hazard を抑制した例を示します。

図 69. カルノー図とStatic Hazard抑制回路付MUX回路



カルノー図の作り方は、X 軸と Y 軸に入力変数を書きます。先に延べたように、入力が 1 個のみの変化の場合はハザードを発生しませんので、それを調べるために、変数が 1 個だけ変化するように並べます。先の切換回路(MUX)であれば、変数は A, B, S の 3 変数ですので、X 軸に AB の 2 変数を 1 ビットの変化で有るように、“00” → “01” → “11” → “10”と並べ、Y 軸には S を“1” → “0”と並べます。続いて、各入力に対する出力 Z を表の中に埋めます。この例で言えば、入力切換信号 S が“1”の場合は入力 AB が“11”の時と“10”の時、及び、S が“0”の場合は AB が“01”と“11”の時となります。

図 69 の左図(カルノー図)で、切換信号Sの欄で“1”が横に 2 個並んだ項が正規出力項ASとB/Sを示す必須項です。今、出力が“1”となる入力状態の内、入力の値が 1 つだけ異なる隣り合った状態をみると、Sが“1”の場合、Bが変化してもASがZを“1”に定義し続け、Sが“0”の場合は、Aが変化してもB/SがZを“1”に定義し続けますのでハザードは生じません。しかし、Sの変化については正規項に含まれていません。これがハザードを発生する原因です。そこで、AB項(“11”)を追加すれば、Sの変化に対しても、AB項がZを“1”に定義し続けますので、ハザードを除去することができます。この項は、必須項ではないので、一般の論理合成では削除されますが、ハザードの防止と言う観点からすれば必要となります。従って、**ハザードを防止する目的で明示的に冗長機能をソースに記述する場合は、その個所を論理合成しないように設定することが必要です。**

7-2 ハザード信号のシステムへの影響

正規化されたロジック信号は、実現された電子回路内で、目に見えない形で存在します。この信号は、計測器でも観測が困難な場合があります。特に、ロジック回路観測用に使われるロジック･アナライザでは観測が不可能です。オシロスコープなどの波形観測用の機材でもかなり高価なものを使用する必要があります。また、ハザード観測のためには知恵と熟練が必要となります。

特に、始末が悪いのは、実験室や製造工場などの恵まれた環境（安定した電源環境、空調の効いた部屋等）で試験された装置は、大概にして安定動作してしまいます。“動作してしまう”と表現したのは、恵まれた環境内では、悪魔（ハザード）が出てこないことが殆どです。実験室や工場での検査段階で問題が発生してくれれば、市場に出荷する前に対処できます。しかし、この悪魔（ハザード）は意地悪で、

彼らが住み易いとされる不安定電源や高･低温などの悪い環境に行くと直ぐに顔を出します。皆さんは、“夏場になったら機器が不安定になった”とか、“製造ロットが変わったら機器が不安定になった”等の経験はありませんか？この主たる原因がハザード（悪魔）です。

このハザードは、先にも述べましたが、環境の変化によって素子のタイミングが変化することに起因しています。特に、CPLD/FPGA の世界では、同一型名でありながら製造プロセスがどんどん進化することです。他の半導体ロジック素子では、一生同一製造プロセスが採用されるのが一般的ですが、CPLD/FPGA の世界では、歩留まり改善や高速化のためにプロセスの微細化が行われ、これによって素子の遅延が早まる傾向にあります。結果として、信号間のタイミングに差異が生じハザードを誘発します。特に、生涯ロットが大きく、永年にわたって繰り返し生産されるが行われる機器では、“過去の製造ロットでは問題が発生しなかった”が、“最新ロットは不安定だ”というようなことが起こります。このような場合に陥り易い問題解決のアプローチとして、“過去に正常動作していたから設計には問題無い”といって、設計レビューに目を向けないことがあります。このような場合は問題解決手法がその場凌ぎとなり、後に同じ問題を引起す場合があります。

このタイミングのずれは、電源電圧の変動や装置が設置される環境での使用温度変化でも“ハザード”発生の原因となりますが、それらの原因を纏めますと、主として次の3つとなります。

1) 電源電圧の変動
2) 周囲温度の変動
3) 製造プロセス変更

ハザードがシステムに与える影響は計り知れないことが多いですが、これらのシステム上における不安定要素を解決するためには、先に述べた“ハザード抑止策”を含めて以下の方法を採用することをお勧めします。

1) クロック同期によるレジスタ回路でのハザード回避策
2) 出力信号のレジスタ･ラッチ出力
3) 冗長回路によるハザード抑止策

注　ハザードの抑止を含むシステムを設計する上での、様々な高い動作信頼性を確保するための方法を巻末の参考文献に紹介していますので、是非ご覧ください。

7-3　ハザード信号のシステムへの影響のおわりに

一般的には、ハザードの現象はシミュレーションや計測器でも殆どの場合が発見できません。ですから、「電子回路で生成される回路はハザードを常に発生するもの」として、「発生しても問題無い回路」構成にすることが肝要です。そのためには、**同期設計に勝る方法はありません。**

8 デバイス構造とその設計手法

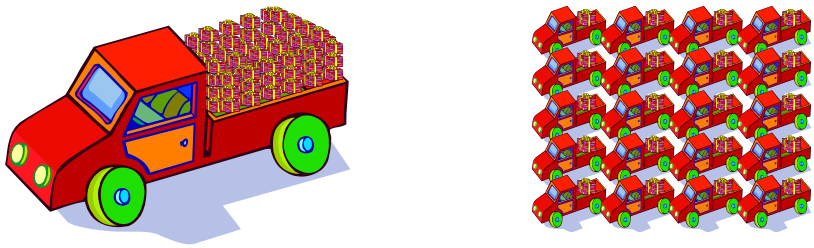
CPLD/FPGA を使用する場合、使用するデバイスのアーキテクチャによって設計方法に留意した方がパフォーマンスや集積度に改善が見られ場合があります。それは、基本ロジック演算部の構成の仕方が異なるためです。これらのアーキテクチャが持つ違いを理解することでより洗練された設計をすることができます。多くの場合は、MAX+plus II や Quartus II のコンパイラがより良い解を提供してくれます。しかし、更なる磨きを掛けてパフォーマンスを改善したり、デバイスの使用効率を改善したり、さらには、コンパイル時間を短縮したりする場合には、この資料による概念が役立ちます。

そこで、この資料では、ALTERA 社のデバイスを使って設計を行う場合にアーキテクチャに依存する部分の設計の仕方の概念について説明します。

8-1 CPLD と FPGA の構造

プロダクトターム方式を採用している MAX Family 系のデバイス(CPLD)や LUT(Look-Up Table)方式を採用している FLEX/Stratix Family 系のデバイス(FPGA)とでは、基本論理ブロックの違いから設計手法が若干異なります。その主たる要因は、入力ゲートの Fan-in の大きさに依存しています。CPLD は、基本論理演算部に大きな Fan-in を持つ AND-OR-XOR 回路を持っています。その Fan-in は、30 本を超えています。従って、他入力の演算を得意としています。一方、FPGA は、4 入力演算を基本としたセル･ブロックとなっていますので、レジスタ系の回路構成を得意としています。これらを比喩した例で説明しますと図 70 のようになります。CPLD は、大きな荷台を持つトラックで一度に沢山の荷物を運ぶことができます。一方、FPGA は軽トラックで一度に多くの荷物は扱えませんが小回りがききます。

図70. FPGAとCPLDの基本演算部イメージ図



先に述べたように、CPLD は大型トラックに沢山の荷物を載せて一度に運びことができます。しかしながら、小さな路地には入って行けませんので小回りがききません。一方、FPGA の方は、小型トラックに荷物を分散して運ぶ方法ですので、一度に多くの荷物を運ぶことは出来ませんが、小さな路地まで入っていくことができます。ちょうど、軽トラによる宅配便を思わせるものがあります。このように、基本演算部に特徴がありますので、この基本演算部のアーキテクチャの特徴を活かした設計をすることにより、パフォーマンスの改善やデバイス内の内部リソースを効率よく使用することができます。また、コンパイル時間を短縮することもできます。

8-2 CPLD と FPGA の設計上の留意点

8-2-1 CPLD 設計上の留意点

CPLDは、Fan-inが大きいので、大きなカウンタやステートマシンを構成するのが得意です。桁上げ演算や条件判定用の回路を 1 つの基本演算部だけで実現することができます。一方、Fan-inの小さいランダム・ロジックを構成することが不得意です。元々大きなFan-inを持つのがCPLDの特徴ですので、小さなFan-inの場合、使用しない残りの入力に付属しているロジックは全て無駄となりますので、結果として内部の未使用リソースが死んでしまいます。各ベンダによって、このような場合の未使用リソースを有効利用するように工夫はされていますが、FPGAに比べるとまだまだ改善の余地があります。また、CPLDの基本演算部は、それ 1 段で全ての論理を構成することを基本概念としていますので、ロジックのカスケード接続を想定しておりません。従って、ロジックのカスケード接続を行った場合は、大幅な遅延を招きます。クリティカル・パス内でロジック・カスケード接続が行われている場合は、その個所がボトル・ネックとなり高速化できません。このような場合は、ロジックがカスケード接続されている部分を**パイプ・ライン化**することをお勧めします。図 71 の左図にカウンタの桁上げ部がFan-inサイズを超え、ロジックが追加された例を示しています。このような場合は、カウンタの速度が半分になる場合もあります。

図 71. CPLD先見桁上げカウンタの例

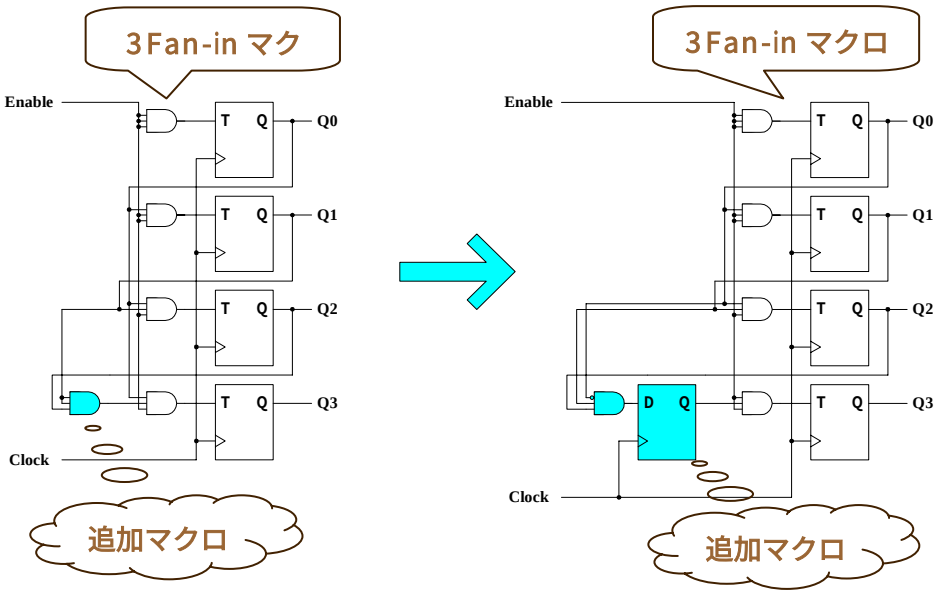


図 71 では、説明のために 1 段当たりの Fan-in を 3 本と仮定して描かれています。左図では、Q3 への桁上げ用ロジックでは、下位からの桁上げ信号と Enable 信号を合わせると 4 本の Fan-in が必要となりますので、1 つの追加マクロセルが必要となります。このマクロセルは、基本マクロセルのほぼ 1 段分の遅延がありますので、この桁上げ回路によって大きく速度を下げる結果となります。そこで、右図のように、追加マクロセルによって Fan-in を増やすのでは無く、Q3 への下位桁からの桁上げ信号を本来の 1 つ前のカウント値でその桁上げに相当する信号をラッチして置き、次のクロックで遅滞無く Q3 に桁上げ信号を出すように改良します。これにより、同じセルの使用数で速度を落とさずに高

速カウンタを組むことができます。この方式は、先見桁上げカウンタと言いますが、カスケードされたロジックをパイプ・ライン化した例となります。

8-2-2 FPGA 設計上の留意点

FPGA は、先に述べましたように、4 本の入力を持つ基本演算部を多数持っています。従って、Fan-in の小さなランダム・ロジックを構成するのが得意となっています。Fan-in を多く必要とする大きなカウンタやステートマシンは不得意です。図 72 は、大きな Fan-in の回路を記述した場合、デバイス内部でどのようにインプリメントされるかを示したものです。

図 72. FPGAへの多入力ゲートのインプリメントの様子

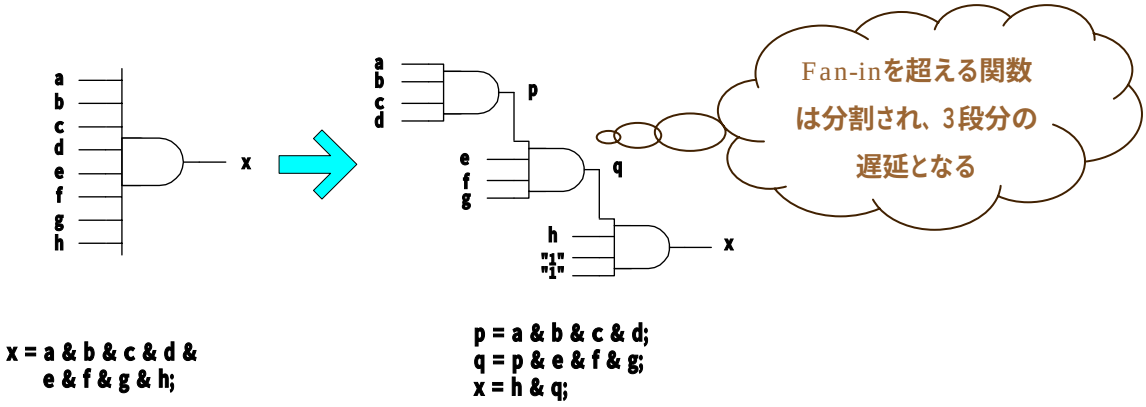


図 72 の右図に示すように、4 本のセル毎に分割インプリメントされますので、設計段階で 4 入力の論理に予め分割して記述しておくと、コンパイル時間が短く、しかも、コンパクトにインプリメントされる可能性があります。ALTERA 社のデバイスでは、このような場合の素晴しいソリューションとして、高速カスケード接続用専用パスを持っています。一般に、ALTERA 社のコンパイラを使用すると多入力 Fan-in パスの処理にこの専用高速パスをアサインするようになっています。しかし、時としてこの非常に有効な高速パスが使われないことがあります。従って、クリティカル・パスにおける多入力 Fan-in の設計に関して明示的にこの高速パスを使用するようソース・ファイル上に定義するとより良い結果が得られます。図 73 に明示的に高速パスを使用するように記述した回路例を示します。

図 73. 多入力ANDを明示的に分割記述した回路例

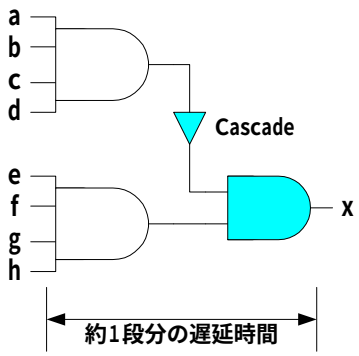


図 72 の右図では、3 段の AND ゲートがカスケード接続されていますので、AND ゲート 3 段分の遅延時間が生じています。しかし、図 73 で示される“Cascade”機能を使用することにより、約 AND ゲート 1 段分の遅延で済んでいます。“Cascade”機能におけるパナルティ遅延は殆どありません。

蛇足ですが、最近の論理合成ツールは、改善に改善を重ねた結果、作為的に分割設計をしなくても満足の行く結果が得られるようになってきています。

8-3 いつデバイス構造に合った設計手法を使うか？

ALTERA 社の CPLD/FPGA 開発ツールである MAX+plus II や Quartus II を使用することにより、殆どのアプリケーションで十分満足する解を得ることができます。しかしながら、時にはその解に飽き足らない場合があります。もうちょっと「パフォーマンスを改善したい」。「もうちょっとリソースの使用率を下げたい」。特に、パフォーマンスの改善は、デバイスを使用する上で度々遭遇する要求条件です。このような場合のクリティカル・パスのタイミングを改善する時にこの手法をお使いください。使用するデバイスのアーキテクチャに適した設計手法を採用することにより、歴然としたパフォーマンス改善を見ることができる場合があります。

8-4 デバイス構造とその設計手法のおわりに

CPLD/FPGA を使用する場合に、使用するデバイスのアーキテクチャを考慮した設計手法を採用するとパフォーマンスの改善、リソースの有効利用、あるいは、コンパイル時間の短縮等に役立ちます。これらの手法は、特に遅延時間のクリティカルなパスに採用すると特に有効です。

改版履歴

Version	改定日	改定内容
1.0	2006 年 03 月	・新規作成

本社
〒163-0928 東京都新宿区西新宿 2 丁目 3 番 1 号 新宿モノリス 28F TEL 03-3345-6205 FAX 03-3345-6209
松本営業所
〒390-0815 長野県松本市深志 1-1-15 朝日生命松本深志ビル 1F TEL 0263-39-6134 FAX 0263-39-6135
大阪営業所
〒532-0003 大阪市淀川区宮原 3 丁目 4 番 30 号 ニッセイ新大阪ビル 17F TEL06-6397-1090 FAX06-6397-1091
名古屋営業所
〒450-0002 愛知県名古屋市中村区名駅 3 丁目 11 番 22 号 IT名駅ビル 4F TEL 052-566-2513 FAX 052-566-2514