Más Práticas de Design

Ao realizar o projeto de circuitos ligeiramente mais complexos, que envolvem o uso do *clock* para criar lógica síncrona, você deve estar atento a obedecer um conjunto de boas regras de *design*. As consequências da não observância destas regras de projeto podem se manifestar quando já é tarde e o projeto já está quase concluído (principalmente nos projetos finais). Infelizmente, nesta etapa do projeto, muitas vezes o conserto é penoso e envolve reprojetar muitas partes do circuito. Por este motivo é importante aprender e entender, mesmo que precocemente (antes do projeto final) as regras que vocês devem seguir.

O principal sintoma de um projeto ruim é o comportamento impredizível. Na prática, isto é verificado em situações como:

- A lógica parece estar 100% correta, mas, quando se monta o protótipo na placa, o comportamento não é o mesmo;
- O circuito é testado e funciona na placa. Entretanto, ao fazer modificações ao projeto, uma outra parte que não foi modificada tem seu comportamento alterado.

Geralmente não é fácil encontrar a causa destes problemas, principalmente porque o comportamento ilógico parece não fazer sentido. Há quem coloque a culpa na ferramenta de síntese, com a premissa de que, se antes funcionava e após uma modificação, ou mesmo recompilação, não funciona mais, então o sintetizador está com um *bug*. O problema, na verdade, é o comportamento impredizível criado por uma má prática de design (exemplos serão dados na próxima subseção).

Uma regra geral é sempre trabalhar com sinais sincronizados. Com os exemplos abaixo, você irá construir um entendimento melhor da razão da superioridade da lógica sincronizada. Em poucas palavras, como os sinais são amostrados apenas na borda do clock, há pouco espaço de tempo para que transições temporárias (*glitches*, ruídos, etc.) sejam passadas para frente e, ainda mais importante, a ferramenta de análise de timing consegue detectar com precisão as situações em que um circuito fere as restrições de tempo para calcular e estabilizar o sinal, tomando medidas para resolver o problema automaticamente (criando um circuito mais veloz ou, se não for possível, emitir um erro de violação do *timing*).

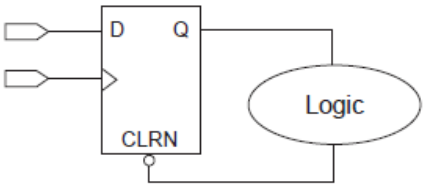
Há uma maneira simples de atestar a qualidade do seu *design*. Basta acessar o menu "*Processing*", submenu "*Start*" e navegar até o item "*Start Design Assistant*". O *Design Assistant* irá apresentar os códigos que seu *design* está violando, acompanho de uma breve mensagem descritiva. Para uma explicação completa, consulte o *Handbook do Quartus*, Capítulo 5, página 16. Se você estiver com o *Handbook* completo (disponível em http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf), esta é a página 208 do PDF.

NOTA: Em alguns exercícios do laboratório, você é convidado a explorar alguns *designs* assíncronos. O objetivo é que você conheça alguns projetos que usam poucas portas lógicas, como o contador assíncrono, utilizando apenas o conceito de *flipflops*. Entretanto, este não é o *design* ideal a ser usado em um projeto maior de FPGA, que não seja puramente para estudo dos efeitos assíncronos. Outro ponto é o uso de componentes legado da biblioteca MaxPlus. Eles devem ser evitados nos projetos, e seu uso é limitado para estudo de algumas características interessantes para um exercício isolado.

Regra **A101 – Design Should Not Contain Combination Loops** e Regra **A102 – Register Output Should Not Drive Its Own Control Signal Directly or through Combinational Logic**

Estas primeiras regras são melhor ilustradas com a figura abaixo, retirada do *Handbook do Quartus*, pág 196.

Figure 5–1. Combinational Loop through Asynchronous Control Pin



Esta figura cobre os dois exemplos, pois há um laço combinacional (A101) e registrador dirigindo seus próprios sinais de controle (A102). Não é tão fácil, entretanto, perceber o problema. Vamos supor que você irá projetar um contador MOD N da seguinte forma:

- Utilização de um projeto de contador tradicional de X estágios, onde $2^X >= N$.
- Um comparador com portas lógicas que, quando o número do contador atinge "N", um sinal de *reset* é gerado para reiniciar todos os flipflops deste contador.

Este design parece perfeitamente válido. Entretanto, se você pensar um pouco (ou consultar o *Design Assistant*), o projeto fere a regra A102. Isto acontece no momento de calcular o sinal de *reset* – uma lógica combinacional compara e vê se o valor é N e, se sim, gera o sinal de *reset* dos próprios *flipflops* que guardam N. Este é um laço combinacional porque, sem nenhuma sincronização (sem esperar o *clock*), o sinal irá percorrer um laço: (1) No momento em que o contador chegar em N, (2) a lógica combinacional irá o gerar sinal de *Reset*, (1) que por sua vez irá zerar o contador, (2) que por sua vez irá desligar o sinal de *reset*. Os números identificam os estágios do laço, em que (1) identifica transição no *flipflop* e (2) identifica transição na lógica combinacional. Perceba que não há espera pelo sinal de clock para passar o sinal para frente.

Isto configura uma situação impredizível, e o circuito pode funcionar ou não. Por quê?

Isto acontece porque o correto resultado depende dos tempos de propagação do circuito combinacional. Suponha que, no projeto de um contador MOD 10 que utiliza 4 *flipflops* (`3 downto 0`), iremos gerar um *reset* assíncrono quando o `bit 3` e o `bit 1` estiverem com o valor `'1'` (situação em que o contador está no valor `10`, em binário, `1010`). Suponha também que o sinal de *reset* está com grande atraso para alcançar o flipflop do `bit 3`. Nesta situação, quando o contador chegar em `10`, o *flipflop* do `bit 1` vai ser zerado **antes** (porque está com pouco atraso), conduzindo o contador ao estado "`1000`". Este bit zerado irá fazer com que o sinal de *reset* seja desativado, pois não estamos mais em "`1010`". E, por causa do atraso, o último bit (`bit 3`) nunca foi reiniciado, porque o *reset* não se estabilizou por tempo suficiente na entrada atrasada de seu *flipflop*.

Esta é uma causa muito comum de impredibilidade nos projetos, devido ao uso de laços combinacionais. No projeto para FPGA, isto está fortemente relacionado à etapa de *placement*, onde a

localização dos componentes é determinada no chip da FPGA. Dependendo da distância entre os componentes, o atraso muda. O *placement* pode ser completamente alterado com uma pequena mudança no seu projeto, que aparentemente "não tem relação com o componente defeituoso".

Outro problema perigoso ao não sincronizar os sinais de controle (como o *reset*), isto é, ao não amostrá-los com um *flipflop*, é a sua baixa tolerância a *glitches*. Um circuito combinacional, quando está propagando um resultado para a saída, possui um período de tempo em que a saída está se estabilizando, em um estado ainda inválido. Isto é exacerbado pelos diferentes atrasos de propagação.

Suponha, por exemplo, que usemos o mesmo projeto do contador "mod 10" comentado anteriormente. Apesar de a lógica parecer implacável em anunciar que o *reset* só irá ser feito quando o contador chegar em `10`, na prática, o *reset* pode ser gerado antes. Isto acontecerá fatalmente se o contador produzir `'1'` no `bit 3` e `'1'` no `bit 1` por um curto período de tempo. E este foi exatamente o problema que um aluno enfrentou neste *design* – o circuito misteriosamente se tornou "mod 8" ao invés de "mod 10". Quando o contador transicionava de `"7" (0111)` para `"8" (1000)`, um atraso grande na propagação do `bit 1` fazia com que este ainda estivesse com o valor `'1'` quando o `bit 3` assumia o valor `'1'`, configurando um falso `"1010"` e efetuando o *reset* do contador. Note que estes problemas de temporização seriam todos evitados se os sinais fossem amostrados por um *flipflop* apenas na borda do clock, e não ligados diretamente a sinais de controle.

Por motivos semelhantes, não se realiza lógica com o *clock* (*gated clock*): Uma falsa transição no sinal de *clock*, se este estiver sendo gerado por lógica combinacional, pode ser longa o suficiente para ativar os *flipflops*. Isto nos leva a próxima regra:

C101 Gated Clocks Should Be Implemented According to Altera Standard Scheme

O *gated clock* (isto é, *clock* que passa por um circuito combinacional) pode ser muito útil para economizar energia. Se o *clock* passar por uma porta AND, você pode facilmente "desligar" o *clock*, silenciando todas as transições para um grande bloco do circuito, economizando energia dinâmica. O *gated clock* é tolerado para este objetivo específico e, mesmo neste caso, a Altera dita um método específico de realizar o procedimento. Entretanto, para a disciplina MC613, vocês não precisam (nem devem) usar lógica que atua no *clock*, com exceção dos divisores de *clock*.

As outras regras críticas do *Handbook do Quartus* dizem respeito a técnicas de geração de pulsos que vocês não usarão no curso. Com isto, todas as regras críticas importante foram cobertas neste pequeno documento. Para as demais regras, você sempre pode consultar o *Handbook do Quartus* ou solicitar ajuda na lista de dúvidas. Um outro princípio muito importante é evitar a inferência de *latches* dentro de circuitos que foram projetados para lógica combinacional, mas isto é coberto no guia "Pensando em Sinais" (basta que você elabore com cuidado o seu *process*, definindo os sinais de saída em todos os fluxos da execução do *process*).