FEUP Universidade do Porto Faculdade de Engenharia

Departamento de Engenharia Eletrotécnica e de Computadores

Projeto de Sistemas Digitais

4º ano - 1º semestre
Exame – 19 janeiro 2015

CORREÇÃO

Duração máxima: 2h30m sem consulta de apontamentos

[4 valores]

1 - Como certamente pode constatar durante a realização dos projetos laboratoriais, a realização correta dos vários passos de verificação (realizados com recurso a simulação lógica) é fundamental para garantir o sucesso do projeto.

[2 valores]

a) Admitindo que a verificação funcional é bem sucedida, explique porque razão é necessário realizar o processo de verificação pós-sintese, simulando o modelo traduzido do código fonte Verilog. Apresente pelo menos um exemplo que suporte a sua explicação.

A verificação pós-síntese é necessária para confirmar a correção do circuito lógico obtido pelo processo de síntese, que pode não ser funcionalmente equivalente ao código fonte HDL (Verilog, no nosso caso). Isso pode acontecer devido a erros de codificação que não traduzam corretamente as funções lógicas pretendidas, por não conformidade com as regras impostas pela ferramenta de síntese utilizada. O exemplo mais comum é o uso de atrasos explícitos no código fonte Verilog, que são interpretados na simulação funcional mas que não se traduzem para qualquer elemento lógico e por isso pode tornar a funcionalidade do circuito sintetizado diferente do modelo Verilog. Outro erro de modelação acontece quando se usam atribuições blocking *na modelação de processos sequenciais: o simulador pode avaliar essas atribuições de forma diferente do circuito sintetizado.*

[2 valores]

b) Durante o projeto laboratorial não foi realizada a simulação *post-route* do projeto completo (s6base_top.v), tendo-se passado diretamente para o ensaio na FPGA. Isso pode ser feito depois de ter verificado corretamente o bloco que desenvolveu e integrou no projeto principal, e sabendo também que os restantes componentes que compõem esse projeto já haviam sido verificados. No entanto, podem ocorrer erros na integração que comprometam o funcionamento correto do circuito e que seriam facilmente detetados se fosse realizada uma simulação de todo o circuito. Apresente exemplos de erros de projeto que podem acontecer nessa fase de integração e diga com que processo de simulação (funcional, pos-síntese ou *post-route*) poderiam ser detetados.

*O erro mais comum que ocorre na integração de vários blocos no modelo principal (*top-level*) é a interligação incorreta entre blocos, motivada pela declaração errada dos elementos de ligação (wires), ou troca de portos dos blocos que são integrados (por exemplo, fazendo a interconexão de blocos pela posição de declaração dos portos, é fácil passar despercebida uma troca dos portos que são ligados). Estes erros são facilmente detetados com simulação funcional. Outro tipo de erro pode ocorrer se as interligações de blocos forem feitas com atrasos explícitos. Este caso só será identificado numa simulação pos-síntese.*

[4 valores]

2 - Num circuito digital síncrono, a temporização do sinal de relógio e do sinal de inicialização global (*reset*) são críticos para o seu funcionamento. De forma semelhante ao sinal de relógio, também os tempos de atraso associados às ligações que encaminham o sinal de *reset* devem assegurar um estado inicial coerente em todos os elementos síncronos.

[2 valores]

a) Admita que um sinal de *reset* proveniente de um circuito externo é ativado durante um período de tempo correspondente a poucas dezenas de ciclos de relógio e é diretamente usado como *reset* síncrono pelos *flip-flops*. Explique qual poderá ser a consequência negativa de ter diferentes atrasos de propagação entre a origem do sinal de *reset* e os *flip-flops* que o utilizam, mesmo que essas diferenças sejam pequenas face ao período de relógio (por exemplo, da ordem de 1/10 do período do sinal de relógio).

Se o sinal de reset é assíncrono com o sinal de relógio, uma vez que tem origem num circuito externo, (e pode mudar de estado em qualquer instante), pode acontecer que ao ser libertado em diferentes flip-flops em torno de uma transição ativa de relógio, seja visto como ativo por alguns dos flip-flops e como não ativo por outros, provocando a inicialização incorreta do estado inicial dos elmentos de memória (por exemplo, do registo de estado de máquinas de estado)

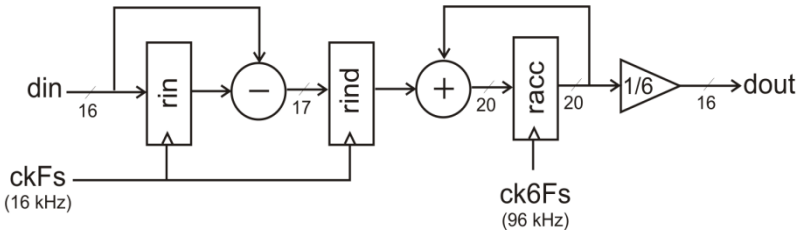
[2 valores]

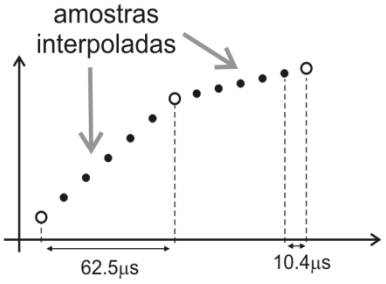
b) Considerando que os elementos síncronos implementam processos de *reset* síncrono, diga como deve ser tratado esse sinal de *reset* para evitar que ocorra o problema referido na a).

Basta passar o sinal externo de reset por um sincronizador formado por um shift-register de 2 andares (2 flip-flops em cascata), e usar como sinal global de reset a saída do 2º flip-flop.

[10 valores]

3 - A figura representa o diagrama de blocos de um circuito digital interpolador, que aumenta em 6X a frequência de amostragem de um sinal de entrada, gerando para cada amostra recebida 5 novas amostras à frequência de amostragem F2 = 6xF1, obtidas por interpolação linear. O circuito é composto por um primeiro andar que opera à frequência F1=16 kHz (diferenciador) e um segundo andar (integrador) que trabalha com um sinal de relógio com a frequência F2 = 6xF1 = 96 kHz. Os sinais de entrada e saída são palavras de 16 bits, representando valores com sinal em complemento para dois. O interpolador deve ter um ganho o mais próximo possível de 1, mas não superior.





[2 valores]

a) Admita que tem disponível o módulo `div6`, que implementa um circuito combinacional para realizar a divisão aritmética por 6. Complete o código Verilog dado, para representar um modelo sintetizável do interpolador. Implemente um mecanismo de inicialização (*reset*) síncrono e ignore, para já, o facto de os dois sinais diferentes de relógio `ckFs` e `ck6Fs` poderem causar situações de meta-estabilidade no registo `racc`

```
module interpol6X( ckFs, ck6Fs, reset, din, dout );
input ckFs, ck6Fs, reset;
input  signed [15:0] din;
output signed [15:0] dout;
reg    signed [19:0] racc;

reg    signed [15:0] rin;
reg    signed [16:0] rind;

always @(posedge ckFs)
if ( reset )
begin
  rin <= 16'd0;
  rind <= 17'd0;
end
else
begin
  rin <= din;
  rind <= din - rin;  // differentiator, 17 bits
end

always @(posedge ck6Fs)
if ( reset )
begin
  racc <= 20'd0;
end
else
begin
  racc <= racc + rind; // integrator, 20 bits
end

div6 div6_out( .in( racc ), .out( dout ) );

endmodule
```

[2 valores]

b) O módulo `div6` deve realizar a divisão pela constante 6, arredondando o resultado através de um processo semelhante ao que foi usado no projeto laboratorial. Para não usar um divisor, cuja realização lógica é mais complexa e lenta do que um multiplicador, optou-se por realizar esta operação como o produto por 1/6 = 0.16(6). Mostre como implementar esta operação como um circuito combinacional construído com base num multiplicador (obtido pela síntese do operador '*'). Apresente claramente a sua solução como um excerto de código Verilog ou um diagrama de blocos e não considere o arredondamento final, que será feito pelo processo referido acima.

*Em vez de codificar diretamente o produto pela constante fracionária 0.16(6), o que não é suportado pelas ferramentas de síntese RTL (i.e., assign dout = racc * 0.16666), pode-se fazer a mesma operação recorrendo apenas a aritmética inteira, realizando o produto por uma constante inteira obtida por arredondamento de $2^k/6$, em que k representa o número de bits fracionários significativos efetivamente utilizados, realizando a seguir a divisão por 2^k, o que não é mais do que desprezar os k bits menos significativos do resultado. Por exemplo, com k=10 (2^k=1024), a divisão por 6 pode ser representada como o produto por 170 (170 = floor(1024/6)) seguido pela divisão por 1024. Em Verilog HDL isto pode ser escrito como:*

```
wire signed [27:0] tdiv;   // tdiv requires 8 extra bits
assign tdiv = racc * 170;  // multiply by 170 = floor(1024/6)
assign dout = tdiv[25:10]; // divide by 1024, truncate 10 least significant bits
```

[2 valores]

c) Tendo em conta o valor de F2 (96 kHz), acha que se justifica implementar o módulo `div6` de forma combinacional, como proposto na alínea anterior? Fundamente a sua resposta e, caso entenda que não, indique alternativas.

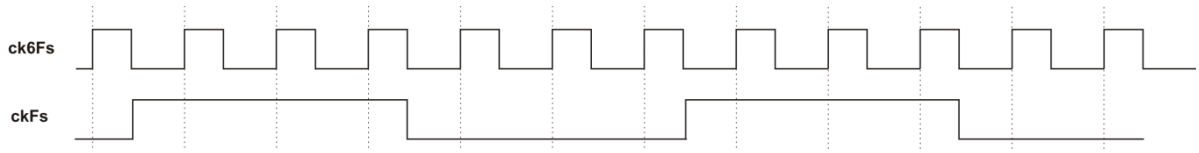
A resposta pode ser sim ou (não-exclusivo) não, desde que devidamente justificada:

***Não**: Nesta aplicação, aquela operação é realizada a um ritmo muito mais lento do que o permitido por qualquer technologia digital atual. Mesmo admitindo um clock de apenas 1 MHz, a mesma operação poderia ser feita com um multiplicador sequencial do tipo shift-add que precisaria de apenas 8 ciclos de relógio para realizar o produto por 170, com uma complexidade lógica muito menor do que o multiplicador combinacional. Esta opção é relevante se o objetivo do projeto for minimizar a área global do circuito, como será o caso se a tecnologia alvo for um ASIC.*

***Sim**: Se a tecnologia alvo não tiver constrangimentos para acomodar a complexidade lógica do projeto, como por exemplo a FPGA usada nos trabalhos laboratoriais que tem disponíveis algumas dezenas de multiplicadores combinacionais, usar um desses multiplicadores para implementar aquela operação não acarreta qualquer aumento de "área" e por outro lado permite libertar recursos reconfiguráveis da FPGA para outras funções.*

[2 valores]

d) Para garantir a correta transferência de dados entre os dois domínios de relógio, os sinais de relógio `ckFs` e `ck6Fs` são gerados a partir de um sinal global de 2.4 MHz, de forma a cumprir o diagrama temporal mostrado na figura (ambos os sinais de relógio são usados apenas no seu flanco ascendente e por isso o seu *duty-cycle* não é relevante). Apresente um modelo Verilog sintetizável que produza os sinais `ckFs` e `ck6Fs` a partir do relógio global de 2.4 MHz (`ck`), que serão usados como sinais de relógo do circuito interpolador.




```
module clockgen( ck, reset, ck6Fs, ckFs );
input ck, reset;
output ck6Fs, ckFs;
reg ck6Fs, ckFs;
reg [4:0] count25;  // Counter modulo 25 to generate the 96 kHz clock
reg [2:0] count6;   // Counter modulo 6 to further divide the 96 kHz clock by 6

always @(posedge ck)
begin
  if ( reset )
  begin
    count25 <= 5'd0;
    count6  <= 3'd0;
    ck6Fs <= 1'b0;
    ckFs  <= 1'b0;
  end
  else
  begin
    if ( count25 == 5'd24 ) // counts 0 to 24 to divide the 2.4 MHz clock by 25
    begin
      count25 <= 5'd0;
      ck6Fs <= 1'b0;        // reset counter to zero, set clock ck6Fs to low
      if ( count6 == 3'd5 ) // this event also increments the count6 counter
      begin                 // this counts from 0 to 5, with the falling edge
        ckFs <= 1'b0;       //     of the 96 kHz clock: reset counter and ckFs
        count6 <= 3'd0;
      end
      else
      begin
        count6 <= count6 + 1;                 // when count6 increments to 3
        if ( ckFs == 1'b0 && count6 == 3'd2 ) // (when current state is 2),
          ckFs <= 1'b1;                       // set ckFs to 1 (16 kHz)
      end
    end
    else
    begin
      count25 <= count25 + 1;                   // when count25 increments to 12
      if ( ck6Fs == 1'b0 && count25 == 5'd11 )  // (when current state is 11),
       ck6Fs <= 1'b1;                           // set ck6Fs to 1 (96 kHz)
    end
  end
end
endmodule
```

[2 valores]

e) Uma forma alternativa de construir este circuito interpolador consiste em implementar todo o sistema síncrono com um único sinal de relógio com frequência igual ou superior a 96 kHz, definindo o ritmo de funcionamento das partes que trabalham mais lentamente à custa de sinais de *clock enable*, produzidos por um processo semelhante ao referido na alínea d). Compare esta solução com a que foi inicialmente apresentada (com 2 domínios de relógio), em termos de rapidez, recursos

lógicos necessários (ou área ocupada) e consumo de energia.

Em termos de rapidez não há qualquer diferença entre as duas formas de implementação, uma vez que qualquer uma das soluções deverá cumprir os requisitos temporais da aplicação e assim garantir o mesmo débito de resultados.

Em relação à complexidade lógica, a solução usando um único relógio será um pouco mais complexa uma vez que os flip-flops necessitam de suportar a função de enable síncrono, o que é geralmente implementado com um multiplexador. No entanto, para implementações em FPGA essa diferença não será relevante porque os flip-flops já possuem essa função implementada. Os circuitos necessários para gerar os sinais de clock enable terão uma complexidade equivalente aos necessários para gerar os dois sinais de relógio considerados.

O consumo de energia será potenciamente mais elevado na versão com clock enable. Apesar dos registos terem o mesmo ritmo de transições em ambas as implementações, o sinal de relógio terá mais atividade de comutação na versão com clock enable e isso conduzirá necessáriamente a um maior consumo energético dos recursos lógicos envolvidos na geração e distribuição do relógio.

[Apesar de não ter sido considerado na questão, é de referir também que a versão com clock enable será mais simples de implementar porque é mais fácil garantir as restrições temporais, uma vez que será um circuito síncrono com um único relógio e não se coloca o problema de assegurar o sincronismo da transferência de dados entre os dois domínios de relógio]

[2 valores]

4 - Uma técnica para aumentar o desempenho de um circuito digital síncrono consiste em transformar os circuitos combinacionais que limitem a frequência de relógio em circuitos *pipelined*, dividindo-os em secções e acrescentando registos entre elas. No entanto, apesar desta técnica permitir aumentar a frequência de relógio, porque reduz o tempo de propagação dos circuitos combinacionais entre registos, também aumenta o número de ciclos de relógio necessários para processar um conjunto de dados. Explique em que situações é vantajoso recorrer a *pipelining* como forma de aumentar o desempenho de um circuito síncrono e em que casos essa técnica não deve ser usada por não contribuir para melhorar a rapidez.

Apesar de pipelining permitir aumentar a frequência máxima do sinal de relógio que pode ser aplicado a um circuito, apenas tem interesse se isso conseguir aumentar efetivamente o desempenho de um sistema, medido como a taxa de produção de resultados (throughput). Isso só acontece se o restante sistema em que se insere puder aplicar nas entradas sequências de operandos à mesma cadência do sinal de relógio, sendo os resultados produzidos ao mesmo ritmo a que são aplicados os operandos.
No caso geral, a introdução de N andares de pipeline num circuito combinacional conduz a um aumento da frequência máxima de relógio que é inferior a N vezes a frequência suportada pelo circuito combinacional original (ou se o período do relógio original é To o período mínimo do circuito pipelined será Tp > To/N). No entanto, o circuito passa a necessitar de N ciclos de relógio para propagar para a saída o resultado da operação de um conjunto de operandos e por isso o processamento de um único conjunto de dados isolado fica mais lento.
No entanto, como a cada ciclo de relógio podem ser aplicados novos conjuntos de operados e recuperados novos resultados, o processamento de uma sequência de M dados demorará um número de ciclos de relógio igual a M+N-1. Esta abordagem só compensa se conduzir a um menor tempo de processamento, o que é verdade se Tp.(M+N-1) (no circuito pipelined) for menor do que M.To (no circuito não pipelined).

- FIM -