FEUP Universidade do Porto Faculdade de Engenharia

Departamento de Engenharia Eletrotécnica e de Computadores

EEC0055 - Projeto de Sistemas Digitais

4º ano - 1º semestre

Exame – 19 de janeiro de 2018

Duração máxima: 2h30m, sem consulta.

[1.5 valores]

1 - Considere o ambiente de projeto de sistemas integrados digitais para dispositivos FPGA que foi usado durante as aulas laboratoriais (XILINX ISE). **Explique em que consistem os processos** de síntese RTL e de *Place & Route* e **diga quais são as características relevantes dos modelos** do sistema digital em projeto que são usados como origem e destino para cada um desses processos.

O processo de síntese RTL consiste na tradução automática dos modelos HDL comportamentais que descrevem as funções lógicas a implementar pelo sistema digital em projeto, num circuito lógico (ou netlist*) formado pela interligação de elementos lógicos combinacionais e sequenciais disponíveis na tecnologia alvo. O processo de Place&Route consiste na definição automática da posição física dos blocos lógicos que formam o circuito sintetizado num plano físico onde podem ser construídos (matriz de blocos configuráveis prá-fabricados, no caso de dispositivos FPGA, ou área de silício no caso de um ASIC) e a construção das interligações entre eles, de forma a implementar o circuito lógico criado pela síntese RTL.*

O modelo de entrada para a síntese RTL é o código comportamental HDL (escrito pelo projetista) que descreve de forma abstrata a funcionalidade lógica a implementar. O modelo resultante da síntese RTL é um circuito lógico constituído pelos blocos elementares da tecnologia alvo e que inclui informação temporal (ou atrasos de propagação) associada a esses blocos e estimativas dos atrasos de propagação introduzidos pelas interligações, que nesta fase ainda não podem ser completamente caracterizadas. Este modelo é a fonte para o processo de P&R que produz como resultado um modelo do circuito lógico anotado com a caracterização temporal dos blocos lógicos e das interligações, sendo esta a informação dinâmica mais rigorosa que se pode obter ao longo do percurso de projeto.

[3 valores]

2 - Num dos estágios do projeto de um sistema digital síncrono (com um único sinal de relógio) pode ser calculada uma estimativa para a frequência máxima do sinal de relógio usando as ferramentas de projeto apropriadas.

[1.5 valores]

a) **Diga, justificando, qual é a fase do projeto** em que é obtida a estimativa mais realista **e explique de que forma** é determinado esse valor (apresente um exemplo que ilustre a sua resposta).

Só após o passo de Place&Route é que se dispõe de informação temporal rigorosa sobre o circuito em projeto. Para um circuito digital síncrono com um único sinal de relógio, o período mínimo (ou a frequência máxima) do sinal de relógio que pode ser aplicado ao circuito é igual ao maior tempo de atraso entre todos os caminhos de flip-flop a flip-flop, atravessando os circuitos combinacionais existentes entre eles (ou, de forma mais simplificada, ao maior tempo de propagação dos circuitos combinacionais existentes entre flip-flops).

[1.5 valores]

b) **É possível que o circuito resultante possa** funcionar corretamente com um sinal de relógio com uma frequência francamente superior à máxima estimada (por exemplo 10X superior)? **Em que condições**? **Explique como poderia identificar** uma situação dessas, referindo-se às ferramentas de projeto para dispositivos FPGA que foram usadas nas aulas laboratoriais.

É possível, se os flip-flops a montante e a jusante do caminho combinacional mais demorado que determina a frequência máxima do relógio aplicável ao circuito forem ativados a um ritmo inferior (por exemplo 10X inferior) do sinal de relógio. Além disso, nenhum outro caminho de flip-flop a flip-flop poderá naturalmente condicionar esse aumento da frequência de relógio. Por exemplo, se um bloco combinacional entre os registos A e B tem um tempo de atraso máximo de 100ns e todos os outros caminhos combinacionais entre registos têm atrasos de propagação inferiores a 10ns, o circuito poderá operar com um relógio de 100 MHz se

os registos A e B forem habilitados (enabled) apenas de 10 em 10 (ou mais) ciclos de relógio.

Esta situação poderá ser identificada com base no conhecimento do processo implementado pelo circuito (note que o projetista é que definiu que os registos A e B funcionam da forma referida) e também analisando os relatórios gerados pela ferramenta de análise temporal estática (static timing report) onde se podem consultar os tempos de propagação de caminhos combinacionais selecionados.

[3 valores]

3 - A linguagem de descrição de *hardware* digital Verilog suporta dois operadores com comportamentos diferentes para representar a atribuição do resultado de uma expressão a um registo ou a um fio: atribuições *blocking* (=) e atribuições *non-blocking* (<=).

[1.5 valores]

a) **Explique por que razão** um modelo Verilog pode resultar funcionalmente errado se na codificação dos blocos síncronos com sinal de relógio, representados por processos **`always @(posedge clk)`**, forem usadas apenas atribuições do tipo *blocking* (=).

Primeiro, os dois operadores representam a operação atribuição: o resultado da expressão do lado direito é avaliada e o seu resultado é gravado na variável do lado esquerdo (que tem de ser um sinal do tipo reg). Enquanto que as atribuição blocking dentro de um bloco begin...end são entendidas como sendo realizadas na ordem em que aparecem no código, as atribuições non-blocking são avaliadas em paralelo e como tal o resultado atribuído a uma variável não afeta as expressões que existam a seguir. Por exemplo, os excertos de código Verilog i) e ii) representam circuitos diferentes mas ii) e iii) representam corretamente o mesmo circuito:

i)	*ii)*	*iii)*
always @(posedge clock) *begin* *y = a + b;* *z = y * y;* *w = z + y;* *end*	*always @(posedge clock)* *begin* *y <= a + b;* *z <= y * y;* *w <= z + y;* *end*	*always @(posedge clock)* *begin* *w = z + y;* *z = y * y;* *y = a + b;* *end*

Note que não se pode entender como um erro escrever i) em vez de ii), já que pela semântica da linguagem Verilog os dois modelos representam comportamentos diferentes!

No entanto, apesar dos modelos ii) e iii) representaram o mesmo circuito, o modelo iii) pode dar origem a comportamentos errados quando simulado, dependendo dos restantes processos que constituem o modelo do circuito. Por exemplo, se no mesmo ou noutro módulo do circuito existir uma expressão síncrona com o mesmo relógio que usa um dos registos w, z ou y, modificados em iii) através de atribuições blocking, o valor que irá ser realmente usado para a avaliação dessa expressão irá depender de o simulador já ter ou não simulado a atribuição de iii) que altera o registo em causa, porque quando essa expressão for avaliada o registo destino é logo modificado com o valor que lhe é atribuído:

```
always @(posedge clock)
begin
  …
  R <= a + y + b;
  …
end
```

[1.5 valores]

b) **Diga, justificando, em que fase de verificação** (simulação funcional, simulação pós-síntese ou simulação pós-place&route) esse erro pode ser detetado.

O comportamento errado resultante do uso de atribuições blocking em processos síncronos com clock (i.e., escrever o código iii) em vez de ii)) só se pode manifestar na simulação funcional. Depois de sintetizados, os circuitos resultantes dos dois modelos são equivalentes e ambos corretos.

[3 valores]

4 - O uso incorreto da declaração **`always @*`** para codificar o comportamento de um circuito combinacional pode resultar na síntese de elementos de memória do tipo *latch* transparente em vez de circuitos completamente

combinacionais.

[1.5 valores]

a) **Diga, justificando, quais dos processos seguintes** conduzem à síntese desse tipo de elementos de memória, não representando por isso um circuito combinacional.

(i)
```
always @*
begin
  y = output1;
  if ( enable )
    y = output2;
end
```

(ii)
```
always @*
begin
  if ( enable )
    y = output1;
  else
    z = output2;
end
```

(iii)
```
always @*
begin
  if ( enable )
    y = output1;
  else
    y = output2;
end
```

(iv)
```
always @*
begin
  if ( enable )
    y = output1;
  else
    y = y;
end
```

Os processos (ii) e (iv). No processo (ii) o valor do registo y não é especificado para enable==0 e o do registo z para enable==1. No processo (iv) acontece o mesmo para o valor do registo y, já que a atribuição "y=y" representa o mesmo que não a representar (o registo mantém o seu valor anterior).

[1.5 valores]

b) **Escolha um dos exemplos** que tenha referido na alínea a) e admita que se pretendia realmente implementar a funcionalidade (não combinacional) representada pelo modelo Verilog. **Explique como deveria corrigir** o código apresentado de forma a garantir o comportamento descrito mas usando apenas elementos de memória do tipo *flip-flop* (ativos à transição do sinal de relógio). Considere que os sinais **`output1`** e **`output2`** e **`enable`** são provenientes de funções combinacionais cujas entradas são registos síncronos com um mesmo sinal global de relógio.

Admitindo que enable, output1 e output2 têm a sua origem em registos síncronos com o relógio e por isso mudam de valor de forma síncrona com o relógio, também y e z irão mudar sincronamente com o relógio. Se for apenas trocado always @ por always @(posedge clock), e admitindo que os sinais origem são sincronizados com a transição positiva do relógio, os sinais destino (y e z) seriam atrasados de um ciclo de relógio em relação ao que determina o modelo assíncrono apresentado. Uma solução seria sincronizar com a transição negativa do relógio, estabelecendo as restrições temporais adequadas para o processo de P&R:*

```
always @(negedge clock)
begin
  if ( enable )
    y <= output1;
  else
    z <= output2;
end
```

[3 valores]

5 - *Pipelining* é uma estratégia de projeto que pode ser usada para aumentar o desempenho de um circuito combinacional intercalado entre registos. Considere que o desempenho do circuito é caracterizado pelo número de resultados produzidos por unidade de tempo. **Diga, justificando, se são verdadeiras ou falsas** as seguintes afirmações:

i. É possível conseguir um aumento do desempenho do circuito *pipelined* mesmo que seja mantida a frequência do sinal de relógio.
Falso. O desempenho, tal como é referido, mede a taxa de resultados produzidos por unidade de tempo e num circuito síncrono com relógio esse valor não pode ser superior à frequência de relógio.

ii. O aumento de desempenho que se obtém com a introdução de N registos de *pipeline* (resultando em N+1 partições do circuito combinacional) é sempre inferior a N+1.

Verdadeiro. O aumento de frequência de relógio é limitado a N+1 vezes e esse valor apenas seria atingido se as N+1 partições do circuito combinacional resultassem com tempos de propagação exatamente iguais, o que na prática se pode considerar impossível de atingir.

iii. O aumento de desempenho resultante é independente da cadência a que os dados são aplicados ao circuito.

Falso. O aumento de desempenho conseguido com pipelining pressupõe que os operandos são aplicados à mesma cadência do sinal de relógio (consequentemente os resultados são também produzidos a esse mesmo

ritmo). Note que é possível conseguir um aumento do desempenho modificando um circuito combinacional para pipelined de maneira a obter um aumento de Fclock de N vezes e aplicando os dados na entrada a um ritmo de Fclock/K (a cada K ciclos de relógio), desde que K (inteiro) seja menor do que N. No entanto, o desempenho é sempre dependente da cadência de aplicação dos operandos ao circuito.

[6.5 valores]

6 - Considere o projeto de uma unidade de processamento de dados para efetuar o cálculo da expressão:

$$y = (a*a + b*b) * c$$

Os operandos **a**, **b** e **c** e o resultado **y** representam valores inteiros de 32 bits sem sinal e sabe-se que os operandos assumem valores que nunca conduzem a situações de *overflow*. O sistema em que este bloco se integra irá processar sequências de dados longas (formadas por várias dezenas de elementos) que lhe são apresentados à frequência de 50 MHz, devendo produzir resultados a essa mesma cadência. O circuito deve ser capaz de operar com um sinal de relógio de 50 MHz.

Uma primeira versão deste circuito consistiu no módulo Verilog seguinte:

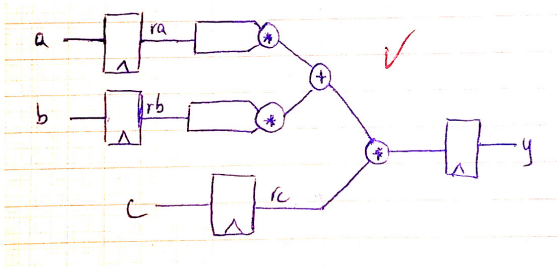
```
module ex1718q6( input clock, input reset,
                 input [31:0] a,
                 input [31:0] b,
                 input [31:0] c,
                 output reg [31:0] y
               );

reg [31:0] ra, rb, rc;

always @(posedge clock)
begin
  if ( reset )
  begin
    ra <= 32'd0; rb <= 32'd0; rc <= 32'd0; y <= 32'd0;
  end
  else
  begin
    y <= ( ra * ra + rb * rb ) * rc;
    ra <= a; rb <= b; rc <= c;
  end
end
```

[1 valor]

a) **Esboce um diagrama de blocos** que represente o circuito lógico em que será traduzido aquele módulo Verilog. Construa esse diagrama com base em blocos ao nível RTL (registos, multiplexadores, somadores, multiplicadores, etc) identificando devidamente todos os sinais envolvidos (pode omitir a representação explícita dos sinais **reset** e **clock**).

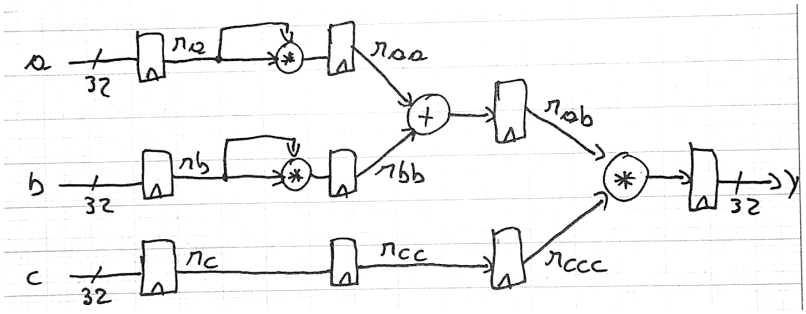


[1.5 valores]

b) No final de um certo estágio do processo de implementação obteve-se uma estimativa para a frequência máxima de relógio de apenas 39 MHz (período igual a 25.6 ns), longe dos 50 MHz pretendidos (período de 20 ns). **Modifique o modelo Verilog** dado de forma a conseguir aumentar a frequência máxima do sinal de relógio suportada pelo circuito **e justifique o aumento estimado para o relógio** com a transformação que propõe.

A solução é transformar o circuito para pipelined. A questão é definir se deve ser usado um ou dois andares de pipeline para seja possível (embora não garantido!) satisfazer a frequência de relógio pretendida. Pode-se estimar que o caminho combinacional mais longo do circuito original tem um tempo de propagação máximo de 25.6 ns e percorre 2 multiplicadores e um somador. Sabe-se que, para um mesmo número de bits um somador apresenta um tempo de propagação que se pode estimar como sendo metade do do

multiplicador. Assim, podemos estimar que 26ns = Tp_{sum} + 2 x (2x Tp_{sum}), e daqui estimar um tempo de propagação para o somador de 5.2ns e para o multiplicador de 10.4ns. Esta análise permite concluir que um único andar de pipeline inserido ante sou depois do somador poderá conseguir reduzir o período de relógio para um valor abaixo dos 20ns pretendidos. Se não for suficiente, então será necessário inserir um segundo andar de pipeline do outro lado do somador, como mostra a figura:



O código Verilog modificado é (considerando apenas um andar):

```
reg [31:0] ra, rb, rc,
           rpipe_add, rpipe_c;

always @(posedge clock)
begin
  if ( reset )
  begin
    ra <= 32'd0; rb <= 32'd0; rc <= 32'd0; y <= 32'd0;
    rpipe_add <= 32'd0; rpipe_c <= 32'd0;
  end
  else
  begin
    rpipe_add <= ( ra * ra + rb * rb );
    y <= rpipe_add * rpipe_c;
    ra <= a; rb <= b; rc <= c;
    rpipe_c; <= rc;
  end
end
```

[1 valor]

c) Considere que a versão do circuito que propôs em b) consegue operar com o relógio de 50 MHz. **Diga quanto tempo é necessário** para completar o processamento de sequências de (i) 10 elementos, (ii) 100 elementos e (iii) 1000 elementos.

Desde o instante em que o primeiro conjunto de dados é carregado nos registos de entrada (ra, rb, rc), o circuito pipelined com um nível de registo interno demora dois ciclos de relógio até que o resultado desses operandos seja carregado no registo destino. Assim, para um conjunto de N operandos aplicados nas entradas à cadência do relógio são precisos (N + $N_{registos\ pipeline}$ + 1) ciclos de relógio para completar o processamento até que o último resultado seja carregado no registo de saída: para 10 elementos são precisos 12 ciclos de 20ns (240 ns), para 100 elementos 2040 ns e para 1000 elementos são precisos 20040 ns.

[1.5 valores]

d) Admita agora que este circuito deve continuar a operar com um relógio de 50 MHz, mas que só tem de produzir resultados a um ritmo de 10 MHz (i.e. a cada 5 ciclos do relógio de 50 MHz). **Modifique o modelo Verilog dado** de forma a satisfazer estes novos requisitos temporais, **minimizando a sua complexidade lógica** (área).

A estratégia para minimizar a área é a mesma que foi usada no projeto laboratorial: partilhar operadores (hardware) ao longo do tempo disponível. Como temos 4 operações para realizar a cada 5 ciclos de relógio, a solução mais simples e mais económica em termos de área consiste em realizar uma dessas operações em cada ciclo de relógio, usando apenas um multiplicador e um somador, e multiplexadores para selecionar os diferentes operandos para o multiplicador. Um modelo Verilog que implementa esta solução é:

```
reg [31:0] ra, rb, rc;
reg [31:0] opa, opb;
wire [31:0] out_mult;
reg [2:0] state;
```

```
always @(posedge clock)
begin
  if ( reset )
  begin
    ra <= 32'd0; rb <= 32'd0; rc <= 32'd0; y <= 32'd0;
    state <= 3'd0;
  end
  else
  begin
    case ( state )
      3'd0: if ( enable_datain ) // synch arrival of new data in a, b, c
            begin
              ra <= a; rb <= b; rc <= c; // load input registers
              state <= 3'd1;
            end
      3'd1: begin
              ra <= out_mult; // reuse register ra to store ra*ra
              state <= 3'd2
            end
      3'd2: begin
              rb <= out_mult; // reuse register rb to store rb*rb
              state <= 3'd3
            end
      3'd3: begin
              y <= out_mult;
              state <= 3'd0
            end
    endcase

  end
end

// O único multiplicador:
assign out_mult = opa * opb;

always @*
case ( state )
      3'd1: begin
              opa = ra; // durante o estado 1 as entradas do mult
              opb = ra; // sao ra e ra
            end
      3'd2: begin
              opa = rb; // durante o estado 2 as entradas do mult
              opb = rb; // sao rb e rb
            end
      3'd3: begin
              opa = ra + rb; // durante o estado 3 as entradas do mult
              opb = rc;      // são a soma de ra com rb (que têm ra*ra e
            end              // rb*rb, respetivamente) com rc
      default: begin
                 opa = 32'd0;
                 opb = 32'd0;
               end

endcase
```

[1.5 valores]

e) Num cenário de aplicação diferente, pretende-se que aquela função produza resultados a uma cadência de apenas 1 MHz (i.e. a cada 50 ciclos do mesmo relógio de 50 MHz). Diga como poderia implementar uma solução que consiga reduzir ainda mais a área do circuito resultante (nota: não se pretende que construa um modelo Verilog descrevendo o circuito proposto, mas deve justificar a descrição apresentada recorrendo a diagramas de blocos que ilustrem a sua solução).

*Se existirem 50 ciclos de relógio entre cada conjunto de dados, poderemos tirar partido do tempo disponível para realizar (alguns) produtos com multiplicadores sequenciais que necessitam de muito menos área do que um combinacional (um somador acumulador e um shift-register). Como se sabe que um multiplicador sequencial para operandos de 32 bits necessita de 32 ciclos de relógio para completar um produto, não será possível realizar as duas multiplicações em cascata com multiplicadores deste tipo, mas usando dois a operar em paralelo poderemos realizar os dois produtos ra*ra e rb*rb gastando para isso 32 ciclos de relógio. Como o produto final não pode ser realizado desta forma porque não há tempo (ciclos de relógio suficientes), uma solução poderia ser usar para o produto final um multiplicador combinacional. No entanto, essa solução seria possivelmente pior do que a proposta em d), que necessita de apenas um multiplicador combinacional!*

*Uma solução poderia ser implementar esta operação em duas fases, de forma pipelined mas em que cada andar da pipeline trabalharia durante o período de aplicação dos dados. Assim, num primeiro andar teríamos os dois multiplicadores sequenciais que calculavam em paralelo ra*ra e rb*rb, colocando o resultado num registo. O segundo andar da pipeline usaria um outro multiplicador sequencial para realizar a segunda parte do cálculo. Se a aplicação permitisse que os resultados fossem apresentados na saída ao fim de 100 ciclos de relógio, esta seria a solução mais económica. Note que depois de produção do primeiro resultado, os seguintes seriam apresentados à mesma cadência de entrada dos operandos (1 MHz).*

-◖= Fim =◗-