FEUP Universidade do Porto Faculdade de Engenharia

Departamento de Engenharia Eletrotécnica e de Computadores

Projeto de Sistemas Digitais

4º ano - 1º semestre
Recurso – 11 fevereiro 2015

CORREÇÃO

[4 valores]

1 - O primeiro processo de verificação que deve ser realizado no ciclo de projeto é a verificação funcional do código HDL, geralmente feita através de simulação lógica.

[2 valores]

a) Diga, justificando, se é possível através da simulação funcional de um sistema digital quantificar o seu desempenho, com base na medida do tempo necessário para realizar uma determinada função.

Em rigor, não, porque a simulação funcional não usa qualquer tipo de informação temporal da tecnologia de implementação alvo, que é necessária para saber a duração admissível para o período de relógio. A simulação funcional permite conhecer o número de ciclos de relógio necessários para realizar uma operação, mas sem se saber o período de relógio que o circuito é capaz de suportar, que só se pode estimar após a síntese lógica, não se pode quantificar o tempo necessário à realização de uma operação. No entanto, se se considerar uma frequência de relógio determinada e também se sabe o tempo correspondente.

[2 valores]

b) As ferramentas de síntese RTL traduzem o código RTL (Verilog) num circuito lógico, seguindo regras bem definidas e suportando apenas um conjunto restrito das declarações das linguagens de descrição de hardware. Por esse motivo, há construções de Verilog que podem ser usadas para modelação funcional mas que não são suportadas pelas ferramentas de síntese porque não é possível construir um circuito lógico que realize essa funcionalidade. Apresente, justificando devidamente, dois exemplos de excertos de código Verilog que não sejam sintetizáveis, sem referir o uso de *system tasks* para controlo da simulação, como `$stop`, `$display` ou `$monitor`.

Um exemplo é o uso de declarações que envolvam atrasos de propagação explícitos, que não são sintetizados, como no caso:

```
assign #10 y = a + b;
```

A simulação deste modelo irá considerar o atraso de 10 unidades de tempo na "ligação" de y a (a+b), mas o circuito real terá um atraso que irá depender da implementação física desta função.

Outro caso que funcionalmente pode simular de forma correta, mas que não é traduzível para um circuito lógico, acontece quando um mesmo sinal é modificado em dois processos (declarações always ou assign) distintos:

```
assign sum = a + b;
always @*
if ( addcy )
  sum = a + b + 1;
else
  sum = a - b;
```

Neste caso teremos um mesmo sinal (sum[6:0]) a ser "ligado" ao mesmo tempo a dois circuitos distintos, um que o liga a (a+b) e outro que o liga a (a+b+1) ou (a-b).

[4 valores]

2 - Quando se codifica em Verilog HDL um bloco combinacional com a declaração `always @*` é necessário garantir que é cumprido um conjunto de regras para que o circuito sintetizado tenha realmente um comportamento combinacional.

[2 valores]

a) Uma possível consequência de não aderir a essas regras é a geração de elementos de memória do tipo *latch* transparente, o que faz com que o circuito resultante deixe de ser combinacional e passa a ter memória. Explique em que situações são criados esses elementos de memória e de que forma se deve elaborar o código para garantir que isso não acontece.

As latch transparentes são inferidas sempre que no modelo de um circuito combinacional com um processo always @, o valor de um sinal não é especificado para alguma condição que seja avaliada nesse processo, como acontece no seguinte exemplo para o sinal Q:*

```
always @*
if ( en )
  Q = y;
```

Para evitar que isso aconteça, deve-se seguir uma regra de codificação que consiste em definir o valor de todos os sinais em todos os casos das condições avaliadas. Uma alternativa consiste em declarar na entrada do processo um conjunto de atribuições a todos os sinais que são posteriormente modificados condicionalmente, o que representará os seus valores por omissão:

```
always @*
begin
  next_state = state;  // por omissão mantém o estado
  if ( state == 1'b0 && enx == 1'b1 )
    next_state = 1'b1
end
```

[2 valores]

b) Diga por que razão a ocorrência de *latches* transparentes, no cenário descrito na alínea anterior, pode comprometer o comportamento de um sistema síncrono em que esse circuito (pseudo-combinacional) se insere.

A latch transparente memoriza o valor na sua saída quando o sinal de controlo (gate) muda de estado (0-1 ou 1-0, dependendo das condições associadas). Num circuito síncrono os sinais aplicados nas entradas dos circuitos combinacionais provêm de registos e terão, no circuito físico, tempos de propagação que irão depender das ligações elétricas realizadas entre os vários dispositivos lógicos. Se não se estabelecerem restrições temporais específicas associadas aos sinais de controlo das latches que possam resultar daquele erro, os sinais de gate e de entrada podem sofrer de atrasos de propagação que façam com a latch adquira valores errados (o sinal gate pode chegar depois do sinal de entrada). Por exemplo, se o sinal traduzido na latch for formado por vários bits, pode acontecer que as latches que implementam alguns dos bits vejam o sinal de gate mais cedo do que outros e por isso seja memorizado um conjunto de bits não coerente.

[10 valores]

3 - Pretende-se desenvolver um sistema que transforme um número complexo representado na forma cartesiana (parte real R e parte imaginária I) em coordenadas polares (módulo M e ângulo de fase A), realizando as expressões seguintes:

$$\mathbf{M} = \frac{1}{4}\sqrt{\boldsymbol{R}^2 + \boldsymbol{I}^2}$$
$$\mathbf{A} = \tan^{-1}\left(\frac{\boldsymbol{I}}{\boldsymbol{R}}\right)$$

Este sistema irá receber uma cadeia de operandos (R,I) à frequência de 150 MHz (período de 6,6 ns), devendo produzir os resultados (M,A) com a mesma frequência e com uma latência máxima de 500 ns

(atraso entre a entrada e saída). Os valores de entrada R e I são recebidos como valores de 12 bits em sinal e complemento para 2 e os valores de saída devem ter 10 bits. O fator 1/4 apresentado na expressão do módulo é apenas para reduzir o valor do módulo de forma a ser representável em 10 bits (sem sinal); o ângulo deve ser um número inteiro em complemento para 2, representando graus.

O sinal global de relógio é de 150 MHz e todo o sistema deve ser implementado com um sinal de inicialização global (reset) síncrono e ativo alto. Admita que o sistema se destina a ser implementado numa tecnologia digital semelhante à que foi usada para os trabalhos laboratoriais, mas que não tem disponível os multiplicadores *embedded* (blocos DSP48).

[2 valores]

a) Para facilitar a implementação da função arctg( ), apenas é calculado o valor do ângulo reduzido ao primeiro octante, sendo posteriormente corrigido para o octante correto, com base nos valores relativos de R e I. Construa um módulo Verilog de um circuito combinacional que codifique em 3 bits o octante a que pertence o número complexo definido por (R,I), sendo 000_2 o código que representa o primeiro octante e 111_2 o 8º octante. Procure minimizar a complexidade lógica do circuito que propõe.

```
module octant( R, I, oct );
input signed [11:0] R, I;
output [2:0] oct;
reg    [2:0] oct;

wire [11:0] Rabs, Iabs; // the absolute values of I and R
assign Rabs = R[11] ? -R : R; // ABS(R)
assign Iabs = I[11] ? -I : I; // ABS(I)

always @*
begin
  if ( Iabs < Rabs ) // possible quadrants 1, 4, 5, 8
  case( { R[11], I[11] } )
    2'b00: oct = 3'b000; // both positive, Q=1
    2'b01: oct = 3'b111; // I negative, Q=8
    2'b10: oct = 3'b011; // R negative, Q=4
    2'b11: oct = 3'b100; // both negative, Q=5
  endcase
  else               // Iabs >= Rabs, possible quadrants 2, 3, 6, 7
  case( { R[11], I[11] } )
    2'b00: oct = 3'b001; // both positive, Q=2
    2'b01: oct = 3'b110; // I negative, Q=7
    2'b10: oct = 3'b010; // R negative, Q=3
    2'b11: oct = 3'b101; // both negative, Q=6
  endcase
end
```

[2 valores]

b) A função arctg( ) será implementada como uma aproximação por 3 segmentos lineares (ver figura na alínea seguinte), recebendo como argumento o quociente I/R nos quadrantes em que I<R (1º, 4º, 5º e 8º) e R/I nos restantes, e devolvendo um ângulo inteiro, sem sinal, entre 0 e 45º. Como o quociente I/R (ou R/I) será sempre menor ou igual a 1, a divisão inteira será efetuada como (I*4096)/R, resultando dessa forma um quociente inteiro que representa um número com 12 bits fracionários no intervalo [0,1]. Dadas as restrições temporais referidas acima, não será viável implementar um divisor sequencial. Uma realização preliminar de um divisor combinacional de números de 24 bits por 12 bits mostrou um tempo máximo de propagação de 210 ns, o que também não permite cumprir as restrições temporais. Explique como deveria realizar este divisor para conseguir atingir a temporização exigida no projeto.

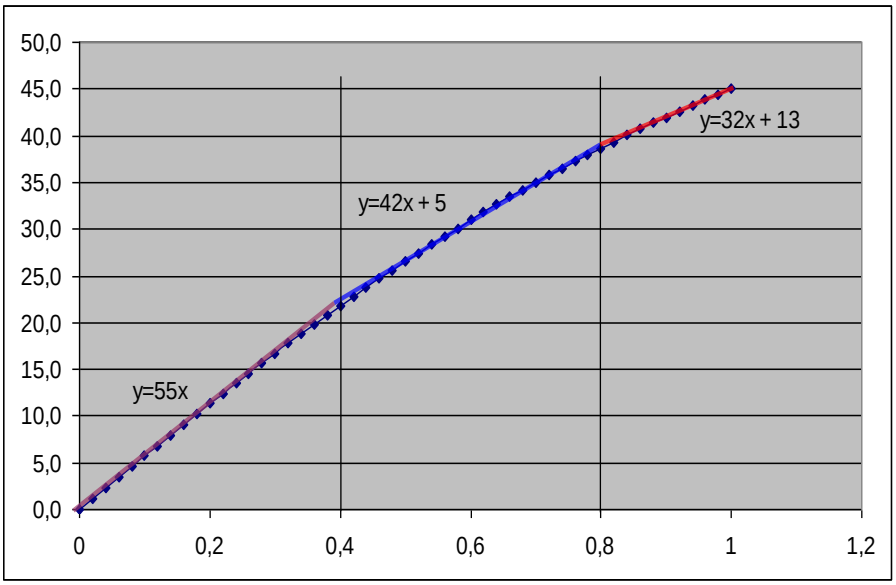
Uma vez que o projeto admite uma latência de 500 ns (500 ns equivale a 74 ciclos de relógio de 150 MHz), a solução seria implementar este divisor como uma versão pipelined do divisor combinacional. Um divisor combinacional, implementado como um array celular, tem um tempo máximo de propagação que corresponde a um caminho crítico com 24x12 = 288 células. Com um nível adequado de andares de pipeline seria possível, em princípio, atingir a frequência de relógio de 150 MHz. Por exemplo, dividindo o circuito combinacional em 58 andares de pipeline (5 células do divisor em cada

andar), seria de esperar que o tempo de propagação em cada andar de pipeline ficasse abaixo dos 6.6 ns requeridos (210 ns / 58 andares pipeline = 3,6 ns). Como a latência máxima corresponde a 74 ciclos de relógio, o restante sistema ainda poderia ter 12 andares de pipeline adicionais.

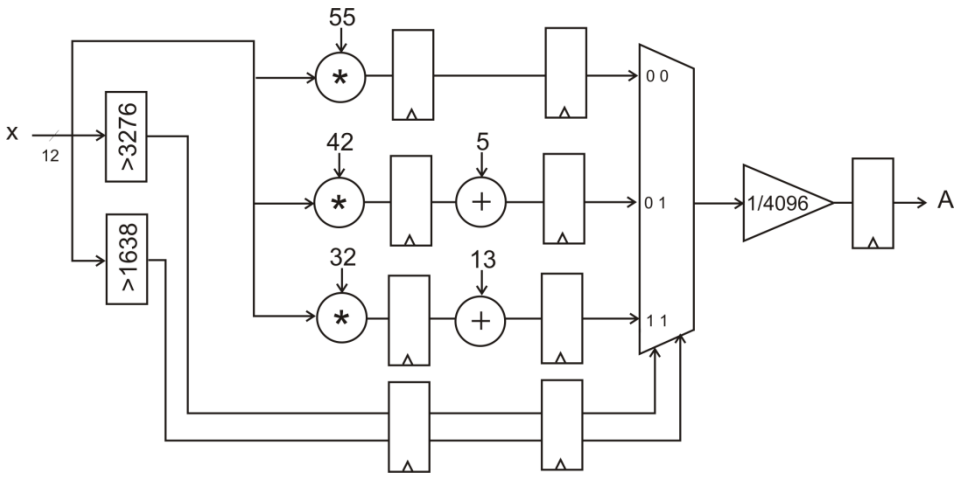
[2 valores]

c) Tendo calculado o quociente I/R (ou o inverso R/I), a função arctg( ) pode ser aproximada, dentro dos requisitos de precisão exigidos pelo projeto, pelos 3 segmentos lineares mostrados na figura 3, onde x é um número positivo representado por 12 bits fracionários:

x entre 0,0 e 0,4:	y = 55 x
x entre 0,4 e 0,8:	y = 42 x + 5
x entre 0,8 e 1,0:	y = 32 x + 13



A figura mostra um circuito lógico que realiza esse cálculo (as constantes 1638 e 3276 são, respetivamente 0,4x4096 e 0,8x4096). Apresente um módulo em Verilog (sintetizável) que represente o circuito da figura, considerando que os produtos por constante são implementados através da síntese do operador *.



```
module angle( clock, reset, X, A );
input clock, reset;
input  [11:0] X;
output [9:0] A;
reg    [9:0] A;

wire Xgt08, Xgt04; // results of comparisons X>0.8, X>0.4
// Registers for the first pipeline stage:
reg [17:0] rmul55, rmul42, rmul32 // regs with output of mults
reg r0Xgt08, r1Xgt08; // regs in the mux select lines
reg r0Xgt04, r1Xgt04;
// Registers for the second pipeline stage:
reg [18:0] raddmul55, raddmul42, raddmul32; // after the adders

// Comparators:
assign Xgt08 = X > 3276;
assign Xgt04 = X > 1638;

always @(posedge clock)
if ( reset )
begin
  rmul55 <= 0; rmul42 <= 0; rmul32 <= 0;
  r0Xgt04 <= 0; r1Xgt04 <= 0;
  r0Xgt08 <= 0; r1Xgt08 <= 0;
  raddmul55 <= 0; raddmul42 <= 0; raddmul32 <= 0;
  A <= 0;
end
else
begin
  // The delays in the mul select lines:
  r0Xgt04 <= Xgt04; r1Xgt04 <= r0Xgt04;
  r0Xgt08 <= Xgt08; r1Xgt08 <= r0Xgt08;
  // The multipliers:
  rmul55 <= X * 55;
  rmul42 <= X * 42;
  rmul32 <= X * 32;
  // The adders:
  raddmul55 <= rmul55;
  raddmul42 <= rmul42 + 5;
  raddmul32 <= rmul32 + 13;
  // The output multiplexer and output register:
  case ( {r1Xgt08, r1Xgt04} )
    2'b00: A <= raddmul55 >> 12; // divide by 4096
    2'b01: A <= raddmul42 >> 12;
    2'b11: A <= raddmul32 >> 12;
    2'b10: A <= 10'hxxx; // synthesized as don't care
  endcase
end
endmodule
```

[2 valores]

d) Depois de sintetizado verificou-se que o circuito anterior não permitia atingir a frequência de relógio de 150 MHz, tendo-se verificado pela análise temporal que todos os caminhos críticos que limitavam o período do relógio envolviam o multiplicador por 55, limitando a frequência de relógio a 120 MHz. Explique como poderia ultrapassar essa limitação.

*A solução seria implementar aquele multiplicador de forma pipelined. Para isso já não poderia ser realizado como a síntese do operador * mas teria de ser codificado explicitamente á custa das somas dos produtos parciais. Como 55 é representado pelo código binário 110111, o produto por*

55 pode ser representado pela adição de apenas 5 produtos parciais, recorrendo a 4 somadores. Dividindo essa rede de somadores em 2 ou 3 andares de pipeline seria, em princípio, suficiente para atingir a frequência de relógio pretendida.

[2 valores]

e) Enquanto a sua equipe se debatia com a função arctg(), os seus colegas estavam também a ter dificuldades em conseguir calcular a função módulo dentro dos 6,6 ns requeridos. Recordando-se do que tinha estudado em PSD, lembrou-se que uma das funções do algoritmo CORDIC era converter coordenadas cartesianas para polares, realizando o cálculo do módulo e ângulo em paralelo e requerendo apenas um número de iterações pouco maior do que o número de bits pretendido para o resultado (neste caso 10 bits, sendo necessário 13 iterações). Diga, justificando, se seria viável recorrer a uma implementação do algoritmo CORDIC, no contexto do problema apresentado, indicando o tipo de implementação (combinacional, sequencial, *pipelined*) e uma estimativa dos recursos lógicos necessários.

*Curiosamente, estava mesmo a pensar nisso! O algoritmo CORDIC seria mesmo **a** solução. No entanto, não poderia ser usada uma implementação sequencial, já que queremos que os resultados sejam produzidos à mesma frequência do sinal de relógio. Uma alternativa seria implementar o CORDIC como uma unidade pipelined em que cada iteração do algoritmo é realizada por um andar independente da pipeline. Seria assim necessário ter pelo menos 13 andares (para 13 iterações), tendo cada um 2 somadores/subtratores e dois registos de 13 bits (ou mais, se fossem usados mais andares). A tabela com os valores de arctg, necessária para o CORDIC, seriam nesta implementação traduzidas em constantes aplicadas diretamente em entradas dos andares respetivos.*

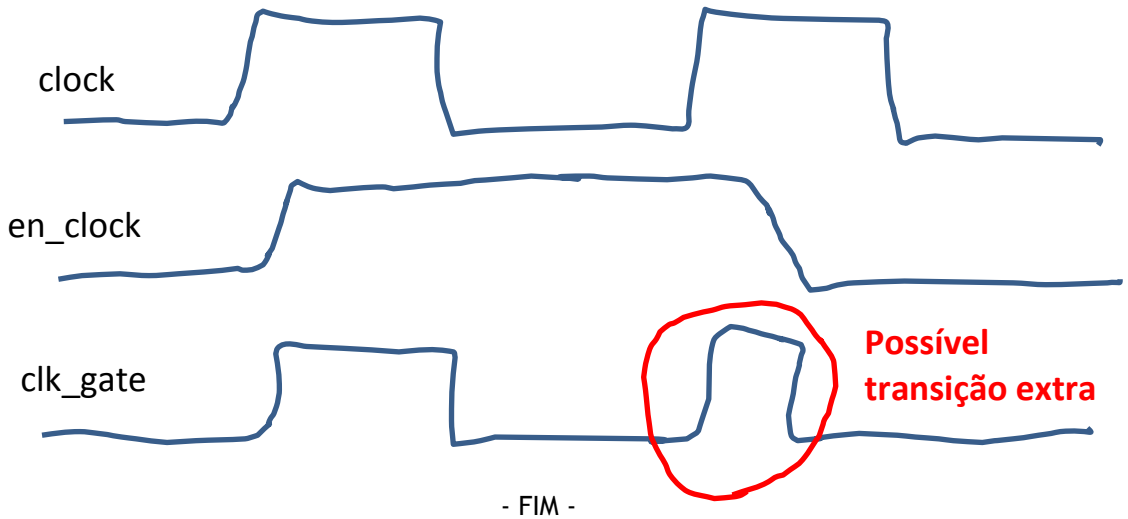
[2 valores]

4 - *Clock gating* é uma estratégia importante para reduzir o consumo de energia em circuitos digitais CMOS, permitindo desligar a atividade do sinal de relógio quando não é necessário. Explique por que razões não se pode simplesmente ligar e desligar o sinal de relógio sintetizando o circuito descrito pelo código Verilog seguinte:

```
assign clk_gate = clock & en_clock

always @(posedge clk_gate)
  reg <= new_reg;
```

Considerando que o sinal en_clock provém de uma máquina de estados sincronizada com o próprio sinal de clock, esse sinal pode provir de um circuito combinacional e por isso ter "glitches" quando ocorrem mudanças do seu estado lógico (motivadas por fenómenos transitórios resultantes dos atrasos de propagação). Com a função apresentada, esses potenciais "glitches" propagam-se para o sinal clk_gate e podem assim ser vistos como transições extra, que não correspondem às transições ativas do relógio principal "clock".
Mesmo que não ocorram "glitches" no sinal en_clock, esta terá um tempo de atraso em relação ao relógio principal e a função AND entre os dois poderá originar uma transição adicional:



- FIM -