

1.1) Frontend views são um conjunto de ficheiros que engloba a datasheet, o LEF, e o verilog do modelo (encriptado), a testbench respetiva e os ficheiros Liberty e permitem que várias equipas desenvolvam blocos em paralelo apenas com especificações dos restantes blocos. Nomeadamente o topo pode ser feito em paralelo com os blocos porque os LEFs definem as dimensões esperadas de cada bloco e o posicionamento dos pinos e a datasheet apresenta as especificações e detalhes de cada sinal de um bloco, logo o designer de topo, apesar de não ter os blocos finais, tem o suficiente para desenvolver o seu trabalho.

1.2) Os testes de caracterização são testes feitos num numero reduzido de samples com o intuito de obter as características do circuito e de analisar o seu funcionamento no silício. Os testes de produção são testes feitos em massa, em todas as samples após o seu fabrico, com o intuito de as testar estruturalmente, por exemplo através de scan chains, onde não verificamos o funcionamento correto do circuito, apenas se não há algo que esteja danificado ou tenha sido mal produzido.

1.3) A vantagem de usar os metais mais baixos em cada bloco é facilitar a interligação no topo que pode vir a ser feita com os metais mais altos, passando por cima de outros blocos. Ora, isto tem os seus revés, visto que os metais mais baixos tem normalmente uma resistência superior à dos metais mais elevados.

2

2.1)

```

module my_fsm (clk, rst, start, skip3, wait3, Zot);
input clk, rst, start, skip3, wait3;
output [2:0] Zot;
reg [2:0] Zot;
parameter state0=0, state1=1, state2=2, state3=3;
reg [1:0] state, nxt_st;

always @ (state or start or skip3 or wait3)
    begin : next_state_logic
        case(state)
            state0: begin
                if (start) nxt_st = state1;
                else nxt_st = state0;
                end

            state1: begin
                Nxt_st = state2;
                end

            state2: begin
                if (skip3) nxt_st = state0;
                else nxt_st = state3;
                end

            state3: begin
                if (wait3) nxt_st = stateee;
                else nxt_st = state0;
                end

            default: nxt_st = state0;
        endcase
    end

always @ (posedge clk)
    begin: register_gen
        if(rst) state<=state0;
        else state<=nxt_st;
    end
endmodule

```

2.2)

```
... // Forçar estado 0 e preparar a transição
reset=1;
start=0;
wait3=0;
skip3=1;
#10 //permitir passagem para o estado 1
reset=0;
start=1;

If(state != state1) //Verifica se chega a 1
    $display("Transição mal executada para State1");
    $finish();
end
$display("Transição verificada para State1");

If(state != state2) //Verifica se chega a 2
    $display("Transição mal executada para State2");
    $finish();
end
$display("Transição verificada para State2");

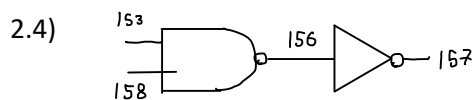
If(state != state0) //Verifica se vai para 3
    $display("Transição mal executada para State0");
    $finish();
end
$display("Transição verificada para State0")
...
```

2.3)

```
module logic(x, y, c, b);
input x,y,c;
output b;
wire d;
```

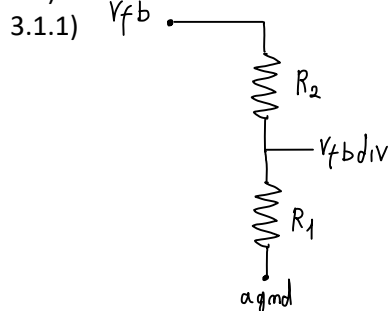
```
assign d = x|y;
assign b = c & d;
```

```
endmodule
```



3

3.1)



3.1.2)

begin

```
    rl_agnd = 0.0;  
    rl_vfb = 0.0;  
    #5  
    rl_vfb=1.0;  
    #20  
    rl_vfb=4.0;  
    $finish
```

end

4)

4.1) O máximo que podemos forçar é $avdd2+vdiod$ e o mínimo é $agnd2-vdiod$.

Mais que $avdd2+vdiod$ iremos polarizar diretamente os díodos do bulk do PMOS.

Menos do que a $agnd-vdiod$ iremos polarizar diretamente os díodos do bulk do NMOS.

No entanto o máximo que podemos ter em $swB2$ é o menor valor de entre $avdd2+vdiod$ e $avdd1+vdiod$ e o mesmo para o seu mínimo que há de ser o maior valor de entre $agnd2-vdiod$ e $agnd1-vdiod$, como tal, se o switch estiver ligado, em $swA2$ ficamos limitados por $swB2$

4.2) Forçando uma tensão externa no $anatestbus$ podemos comparar com a tensão que queremos medir. O output do comparador sai, e vai a um SAR (successive approximation register) que procederá com as aproximações sucessivas até forçarmos um valor de tensão que seja igual aquele com que estamos a comparar.

Tem a vantagem de não afetarmos os nós mais sensíveis, visto que não vamos estar a colocar ruído nem a adicionar capacidade aos nós. No entanto torna-se difícil medir correntes, já que o comparador está a comparar tensões.

4.3)

Primeiramente queremos $SE=1$ e injectar SI , de forma a que o input dos flip flops seja apenas SI e dando bypass à lógica presente entre registos. SI é colocado em série de forma que todos os flip flops sejam carregados com o valor que quisermos. Se temos n registos vamos demorar $n-1$ ciclos de relógio.

Após darmos load ao vector de teste, temos o teste em si, onde colocamos $SE=0$ de forma a colocar a lógica entre os flip-flops novamente, permitindo a que o sinal SI se propague entre as gates. Este processo só demora 1 ciclo

De seguida damos unload da chain através de SO , colocando $SE=1$ e novamente isto é feito em série e como tal iremos demorar $n-1$ ciclos de relógio. E comparando este vector com o esperado sabemos se a estrutura do circuito está correta. Neste estágio podemos começar logo a dar load do novo vector poupando assim $n-1$ ciclos no teste seguinte.

5)

5.1 e 5.2) Um frontend view deliverable deve conter:

- Verilog do modelo:
Aqui temos apenas uma descrição funcional do bloco que servirá para no design de todo se testar a interligação entre os vários blocos.
- Testbench;
Permite verificar o bom funcionamento do bloco e simular.
- Datasheet;
Dá informação relativas aos detalhes do bloco, o seu funcionamento geral e detalhado, os pinos, valores típicos e considerações a ter em atenção a quando do routing e permite a que um engenheiro que não tenha estado envolvido no design do bloco entenda o seu funcionamento.
- Liberty File;
Contem informação relativa às restrições temporais, potencia e ruído dos inputs e outputs do bloco
- LEF.
É uma vista abstrata do bloco que apenas dá ideia dos limites do PR, as posições dos pinos e as camadas de metal da célula

6)

6.1)

- 1 - Import Design
- 2 - Define Floorplan
- 3 - Placing Pins
- 4 - Power Planning and Interconnection
- 5 - Place
- 6 - Clock Tree Design
- 7 - Route
- 8 - Add fillers

6.2) A análise MMMC permite simular vários corners e vários modos de funcionamento do circuito em simultaneamente. Para isto funcionar é necessário library sets (com informação relativa ao PTV das células) e RC corners (com informação sobre os corners das resistências e dos condensadores da tech), que juntos geram delay corners. Adicionalmente é necessário um ficheiro de constraint modes e com isto definimos os vários corners. Depois caso o nosso chip tenha vários modos de funcionamento podemos adicionar outras libraries para definir qual o modo em que iremos simular.

Apesar de MMMC dar para vários corners, podemos fazer a análise apenas para o melhor e para o pior, visto que depois todas as outras cairão no intervalo de funcionamento destas duas.

6.3) O floorplan não pode ser definido para uma utilização de 100% por que a clock tree tem de ser implementada e esta ocupa espaço, por isso é normal fazer para se ter uma utilização de 70%.

6.4) O objetivo da clock tree é garantir que o clock chega a todos sítios em que ele é necessário, por exemplo registos, ao mesmo tempo, ou seja tem como objetivo minimizar o skew para evitar falhas no setup e no hold. Para tal a ferramenta vai organizar as células de forma a colocar esta clock tree as novas células necessárias como os buffers.

6.5) As fillers cell permitem remover erros de nwell spacing, limpar os problemas de densidade e garantir a continuidade dos rails de supply.

6.6) As filler cells tem diferentes tamanhos para garantir que cobrem as diversas possibilidades de buracos e a grid é sempre múltipla do tamanho mínimo da menor célula de fill para garantir que é sempre possível tapar os buracos

6.7) É necessário importar o Verilog pois a ferramenta tem de saber que circuito é que vai fazer. O ficheiro tem de vir já sintetizado. Ou seja tem de ser uma structural netlist, que contem informação sobre as células usadas tal como as respetivas ligações, detalhes necessários para o PR e que não estão no RTL.

6.8) É importante que a ferramenta tenha acesso aos LEFs da core cells e dos sub-blocos que vai usar, visto que precisa de saber as suas dimensões, o posicionamento dos pinos e as metal blockages que estas células impõem, para conseguir fazer place delas e depois o routing.

7)

7.1) No design de um SoC queremos ter em atenção a área, o ruído e pinos com necessidades especiais. Primeiramente queremos minimizar a área para reduzir o custo e maximizar o yield. Queremos separar fontes de ruído de sinais sensíveis e minimizar o comprimento das ligações desses sinais. Além disso queremos minimizar o comprimento de ligações de potência para reduzir a queda IR e ter em atenção à densidade de corrente dessas ligações para minimizar os efeitos da eletromigração

7.2) Os cores digitais podem ser definidos após termos já uma ideia do floorplan para ocuparem espaços com formatos não retangulares, visto podemos controlar a forma da lógica com recurso a placement blockages.

8)

8.1) A proteção ESD necessita de ground e supply pois tem de dar sink da carga obtida pela descarga para algum sitio. E como não sabemos se o pico vai ser positivo ou negativo temos de dar a oportunidade de dar sink para ambos os lados.

8.2) A proteção ESD com díodos são dois díodos ligados ao supply que entram em condução quando existe uma perturbação de tensão superior á da sua polarização. São bons a dar sink de corrente, mas lentos a responder aos picos.

A proteção ESD com MOS (gate coupled) usa um NMOS com um condensador ligado do pin ao gate e uma resistência do gate para o ground. A malha RC serve para carregar o gate do NMOS permitindo então a passagem de corrente no dispositivo. Tem uma resposta mais rápida que os díodos e se não tiver uma resistência em serie com o dreno podemos danificar o dispositivo pois não há limitação da corrente. Tem a desvantagem de colocar uma resistência em série com o pad.

8.3) As células digitais de IO requerem supply e ground pois tem um buffer que necessita de alimentação, e algumas tem PIOS (Programmable IO on Silicon) que permite configurar o funcionamento detalhado da célula e como tal necessita de alimentação.

Estas células também fazem de level shifter, portanto se for um input, queremos colocar o sinal digital ao num nível lógico suportada pelo chip, como tal a sua alimentação deve ser igual à do core. Se for um output, podemos querer mais tensão, visto que vai para o exterior do chip, por isso as células de digitais de output podem ser alimentadas não com core vdd, mas sim com IO vdd, para fazer drive a capacidades maiores.

8.4) As cut cells são necessárias para separar diferentes domínios de alimentação. Tal como indicado na pergunta anterior o ring pode ter diversos domínios em simultâneo, para diversas funcionalidades do circuito e como tal teremos de separar, por exemplo, alimentação digital da alimentação analógica e isso é feito com recurso a cut cells.