	Universidade Federal de Minas Gerais
	Escola de Engenharia – Departamento de Engenharia Eletrônica
	Laboratório de Sistemas Digitais
	Detalhando alguns conceitos em VHDL – Prof. Hermes Aguiar Magalhães

OBJETIVOS:

- Edição de circuitos em VHDL no ambiente Quartus II – Altera.
- Diferenciação de arquitetura estrutural/dataflow e arquitetura comportamental em VHDL.
- Codificação Concorrente e Codificação Sequencial.
- Circuitos combinacionais com tabela verdade incompleta.
- Projeto, simulação e teste de flip-flops e registradores em VHDL

CODIFICAÇÃO CONCORRENTE E CODIFICAÇÃO SEQUENCIAL

Um código VHDL difere radicalmente de uma codificação de software convencional, pois nasce a princípio orientado a eventos. Ou seja, toda sentença tem execução concorrente com as demais sentenças, i.e., suas execuções são “simultâneas”, atentando-se para o fato de que uma sentença só é avaliada se algum dos seus membros à direita da atribuição sofrer modificação. Essa forma de codificação em VHDL leva o nome de codificação concorrente ou DATAFLOW. No entanto, é possível abrir “janelas” de codificação sequencial dentro do código concorrente que se comportam de forma similar à codificação de software que estamos familiarizados, onde várias sentenças são avaliadas uma após a outra em sequência. A estas janelas de codificação para avaliação das sentenças sequencialmente denominamos de codificação comportamental. No entanto, há certas peculiaridades no fato de o resultado final ser um circuito (e não um programa de computador), que alteram radicalmente a forma de codificar o projeto, como veremos oportunamente ¹.

Em VHDL, código comportamental é de execução sequencial (não confundir com o conceito de circuito sequencial em sistemas digitais, que é outra coisa), programado dentro de construções como “PROCESS”, por exemplo. Nele, as sentenças têm uma sequência de execução similar a um programa de computador ¹, declaram-se variáveis (keyword “VARIABLE”) locais para uso interno ao “PROCESS” e os sinais (criados previamente com a keyword “SIGNAL” ou como itens da interface “ENTITY”) são usualmente carregados uma única vez, sendo usados para sinalizar o resultado final do processo sobre os circuitos externos à estrutura. Dentro de um “PROCESS” são válidas as construções sintáticas “IF”, “CASE”, “LOOP” e “WAIT” ². Uma vez que o processamento entrar em um “PROCESS”, as sentenças serão avaliadas em sequência e independente dos estímulos (i.e. independente da existência de eventos nos sinais e variáveis do lado direito das atribuições), onde o fluxo do processamento se dará baseando-se apenas nas decisões lógicas especificadas nas condições de teste codificadas pelo programador. Daí dizer-se que o código nesse caso é “sequencial”.

Já códigos concorrentes estão hierarquicamente “fora” das declarações “PROCESS”, e os próprios blocos “PROCESS” são elementos constituintes de sua estrutura. Toda sentença nesse caso tem execução concorrente com as demais sentenças, inclusive com blocos “PROCESS” eventualmente presentes, ou seja, suas execuções são concorrentes e usam construções como “WHEN”, “SELECT”, GENERATE, etc. Se os eventos nos sinais ocorrerem de forma a acionar tais caminhos concorrentes ao mesmo tempo, as ações de causa e efeito podem ocorrer de forma “simultânea”, originando assim o conceito de

¹ Atente para o fato de que na síntese de circuitos, você não pode contar com os resultados das operações de atribuição onde o destino é um SIGNAL dentro da estrutura PROCESS. Por serem as atribuições dos sinais sempre *concorrentes*, na hora da codificação podemos “imaginar” na prática que a atualização nas modificações dos sinais somente acontecerão ao final da execução do bloco PROCESS. Esta característica diferencia substancialmente esta programação da programação de computadores convencional, daí a palavra “similar”. Para acessar valores intermediários dentro de um PROCESS, é necessário que você os declare como VARIABLE.

² Vale lembrar que a variação “WAIT FOR” da construção “WAIT” não é sintetizável, i.e., não pode ser usada para construir circuitos digitais. Ela é usada apenas na construção de ambientes de simulação ou *testbenches*.

	Universidade Federal de Minas Gerais
	Escola de Engenharia – Departamento de Engenharia Eletrônica
	Laboratório de Sistemas Digitais
	Detalhando alguns conceitos em VHDL – Prof. Hermes Aguiar Magalhães

paralelismo. A necessidade de ligações é suprida pela declaração “**SIGNAL**” (e não por “**VARIABLE**” como no caso comportamental). Note que a declaração de um “**SIGNAL**” ou de uma “**VARIABLE**” não equivale à criação de uma “variável” como a conhecemos da programação de computadores (naquele caso a declaração de uma variável resultaria na reserva de uma posição de armazenamento de memória³). No caso do “**SIGNAL**”, trata-se apenas à explicitação de uma ligação elétrica relevante para o programador. Nas construções concorrentes, uma sentença só é avaliada se algum dos seus sinais à direita da sentença sofrer modificação e um “**PROCESS**” presente nessa estrutura só é avaliado se algum item da sua lista de sensibilidades sofrer alteração.

Note que existe uma equivalência entre os construtos **COMPORTAMENTAIS** ⇔ **DATAFLOW** que pode ser explorada: “**IF**” ⇔ “**WHEN**”, “**CASE**” ⇔ “**SELECT**” e “**LOOP**” ⇔ “**GENERATE**”.

Uma categoria à parte de codificação em VHDL que também está no nível dos comandos concorrentes é conhecida como codificação **ESTRUTURAL** e usa construções com a *keyword* “**PORT MAP**” para descrever textualmente dentro de um código VHDL as ligações, de forma equivalente às ligações existentes em um diagrama esquemático. A codificação *estrutural* serve pra descrever as ligações entre componentes instanciados em uma hierarquia superior que necessita deles, o que é usado para construir sistemas de maior complexidade.

Para um melhor entendimento da diferença de funcionamento na atribuição de sinais em construções VHDL concorrentes e comportamentais, leia atentamente o item “*Código Comportamental × Dataflow ou Código Sequencial × Orientado a eventos*” no documento “*Atribuição de sinais em VHDL.pdf*”, disponibilizado pelo professor. Com certeza, o entendimento da “sutileza” no funcionamento desta atribuição de sinais nos diferentes cenários irá economizar muitas horas de depuração de código e muitas simulações aparentemente com resultados sem sentido, mas que com o entendimento de tais conceitos passam a fazer sentido.

CIRCUITOS COMBINACIONAIS COM TABELA VERDADE INCOMPLETA

Não se deve perder de vista que se está codificando hardware e não software convencional. O código a seguir, na Figura 1-(A), visa obter uma porta AND de 3 entradas usando código comportamental. No entanto, ao invés de obter a função desejada, mostrada na Figura 1-(B), obtemos a função incorreta mostrada na Figura 1-(C). Se você duvida, programe no Quartus, simule seu comportamento e verifique o circuito gerado no RTL-Viewer. Note que foi instanciado um “*latch*” não previsto⁴, que não só prejudica o funcionamento esperado para o circuito, mas também ocupa desnecessariamente mais recursos do hardware a ser programado.

Isto acontece porque na realidade o código, como apresentado, apenas especifica uma linha da tabela verdade da função lógica $S = A \cdot B \cdot C$, que seria a linha $ABC = 111$. Portanto o circuito com a presença do latch reflete esta indefinição fidedignamente, pois para $ABC=111$ a saída $S=1$ e para as demais linhas da tabela verdade a saída é indefinida (exatamente como um *latch* não inicializado). Se você especificar todas as possibilidades desejadas, o *latch* irá desaparecer, resultando em um circuito puramente combinacional como o mostrado na Figura 1-(B).

³ A criação de elementos de memória será tratado mais adiante.

⁴ A Seção 19.5 do Livro “Eletrônica Digital Moderna e VHDL” (autor: Volnei Pedroni, Ed. Campus, 2010) traz uma tabela (Figura 19.7) que explicita as condições para a inferência de registradores pelo compilador VHDL na atribuição de **SIGNAL** e **VARIABLE**.

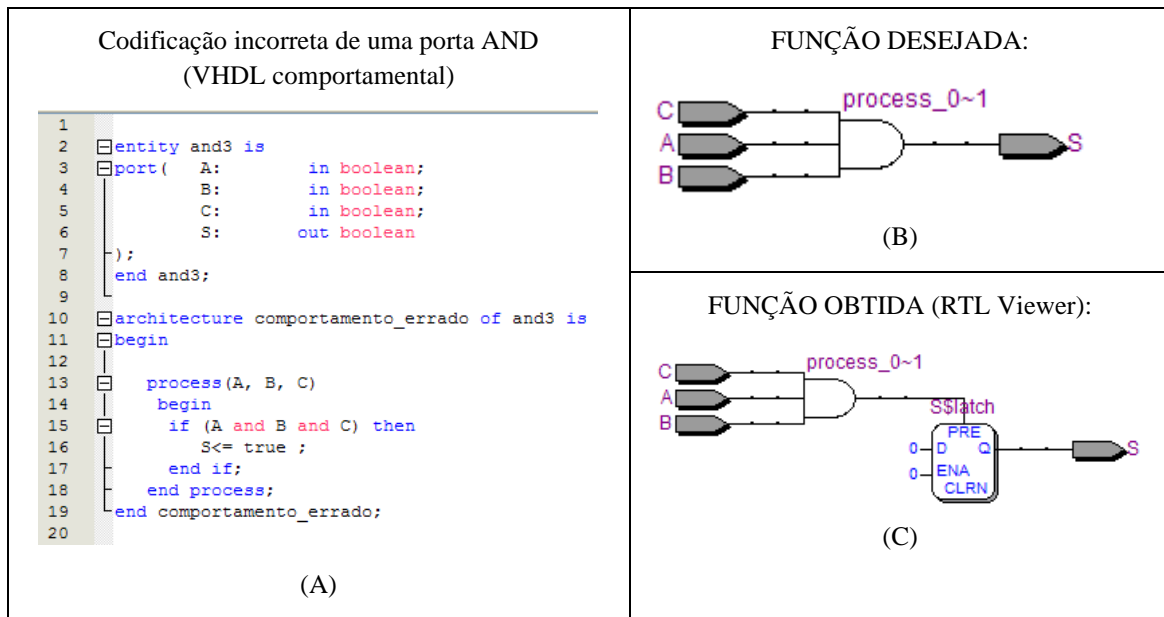


Figura 1: Implementação INCORRETA de uma porta AND de três entradas usando código comportamental.


Este cuidado deve ser tomado em todas as construções de circuitos combinacionais. Portanto circuitos puramente combinacionais são geralmente codificados separadamente de circuitos sequenciais (blocos construtivos em separado dentro de uma arquitetura VHDL). Assim, é possível conferir (usando o RTL Viewer) se foram gerados latches ou flip-flops espúrios nesses blocos, onde só deveriam existir circuitos combinacionais.

PROJETO, SIMULAÇÃO E TESTE DE FLIP-FLOPS E REGISTRADORES EM VHDL:

Podemos descrever os *flip-flops* em linguagem VHDL de forma muito simples. Basta achar uma equação lógica que descreva seu comportamento no tempo, denominada “equação característica” do *flip-flop*. Veja na Tabela 1 abaixo o exemplo para o caso do *flip-flop JK*. Neste exemplo usamos as informações da tabela de transição de estados do *flip-flop*, montamos um mapa de *Karnaugh* onde as entradas são $Q[t]$, $J[t]$ e $K[t]$ e, fazendo a simplificação, obtivemos a equação característica $Q[t+1]=Q[t]K'[t]+Q'[t]J[t]$.

TABELA 1: Estados de um Flip-flop JK

Estado atual $Q[t]$	Entradas $J[t]K[t]$			
	00	01	11	10
0	0	0	1	1
1	1	0	0	1



Próximo Estado = $Q[t+1]$


“Store”	“Reset”	“Toggle”	“Set”
---------	---------	----------	-------

Mapa de Karnaugh: $Q[t+1]$

		$J[t]K[t]$			
		00	01	11	10
$Q[t]$	0	0	0	1	1
	1	1	0	0	1

$$Q[t+1]=Q[t]K'[t]+Q'[t]J[t]$$

Uma possível codificação do flip-flop JK acima usando VHDL é mostrada na Figura 2, onde a equação característica está na linha 28. Ao analisarmos o circuito gerado pelo código mostrado na parte de baixo da mesma figura, vemos que a tecnologia do chip constrói uma lógica combinacional na sua entrada para

	Universidade Federal de Minas Gerais
	Escola de Engenharia – Departamento de Engenharia Eletrônica
	Laboratório de Sistemas Digitais
	Detalhando alguns conceitos em VHDL – Prof. Hermes Aguiar Magalhães

transformar o comportamento do *flip-flop D* no comportamento de um *flip-flop JK*. Note também que o código apresentado conta com uma entrada “*clr*” assíncrona - ou seja, independente da borda do *clock* - para zerar a saída do *flip-flop*.

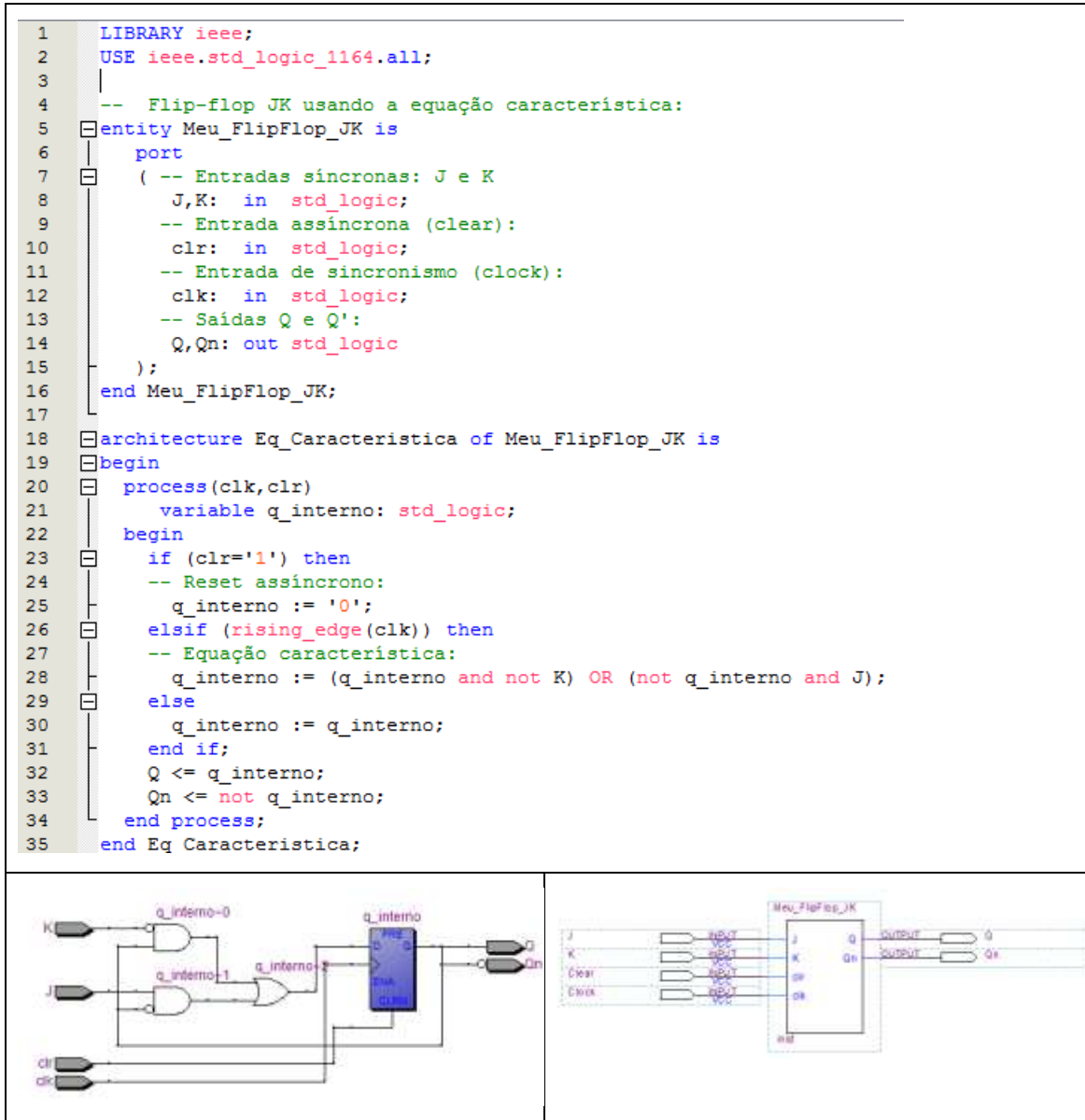



Figura 2: Flip-flop JK com entrada *clr* assíncrona codificado em VHDL usando sua equação característica

Observe que foi necessário criar uma VARIABLE ⁵ *q_interno* dentro do bloco PROCESS para nos possibilitar a ligação da saída do *flip-flop* de volta à rede combinacional de sua entrada sem ter de recorrer a uma interface do tipo “*buffer*” (bidirecional) desnecessariamente. Adicionalmente, como na linha 26

⁵ Lembre-se que na síntese de circuitos, você não pode contar com os resultados das operações envolvendo SIGNAL dentro da estrutura PROCESS, pois é como se a atualização nas modificações dos sinais somente acontecesse ao final da execução da estrutura. Esta característica diferencia substancialmente esta programação da programação de computadores convencional. Na síntese de circuitos, apenas as modificações nas variáveis (de caráter temporário, declaradas como “VARIABLE”) acontecem dentro da estrutura PROCESS.

	Universidade Federal de Minas Gerais
	Escola de Engenharia – Departamento de Engenharia Eletrônica
	Laboratório de Sistemas Digitais
	Detalhando alguns conceitos em VHDL – Prof. Hermes Aguiar Magalhães

condicionamos a execução da sentença de avaliação de $q_interno$ à ocorrência de uma borda de subida do sinal “ clk ” usando a função “ $rising_edge$ ”, e posteriormente usamos seu resultado para atualizar os sinais Q e Q_n (linhas 32 e 33), o compilador VHDL automaticamente instancia um flip-flop para a implementação da função desejada ⁶. Isso porque toda vez que atribuímos a um sinal um valor que depende de uma borda ou transição de outro sinal, o VHDL instancia um flip-flop para realizar isso.

Note também que o identificador “ $q_interno$ ” refere-se na realidade a diversos componentes e ligações do circuito, que irão depender da tecnologia e arquitetura interna do chip utilizado. Não se trata, portanto, de uma “variável” como aquela que estamos habituados em programação de software.

O tipo de flip-flop mais usado atualmente é o *flip-flop* tipo D, especialmente para construir registradores. Registradores são utilizados para armazenar um valor binário qualquer, converter dados paralelo-serial ou serial-paralelo ou ainda deslocar uma palavra binária.

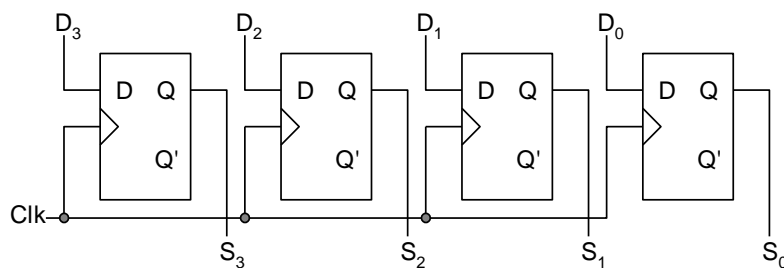


Figura 3: Registrador de carga paralela de 4 bits

O armazenamento de um valor de mais de um bit codificado em binário é realizado por um registrador de carga paralela, que é um circuito sequencial que armazena dados (Figura 3). Na ocorrência de um evento de *clock*, os dados D_3 , D_2 , D_1 e D_0 no exemplo são memorizados e ficam disponíveis nas saídas S_3 , S_2 , S_1 e S_0 dos respectivos *flip-flops*.

A conversão de dados de serial para paralelo pode ser realizada por um registrador de deslocamento (Figura 4), que possui uma única entrada serial S e quatro saídas paralelas P_3 , P_2 , P_1 , P_0 . Na ocorrência de um evento de *clock*, a entrada serial S é armazenada na saída Q do flip-flop mais à esquerda (ponto P_3). O mesmo acontece com os *flip-flops* seguintes, onde o dado na entrada D de cada um é armazenado nas respectivas saídas Q , ficando disponíveis em P_2 , P_1 e P_0 .

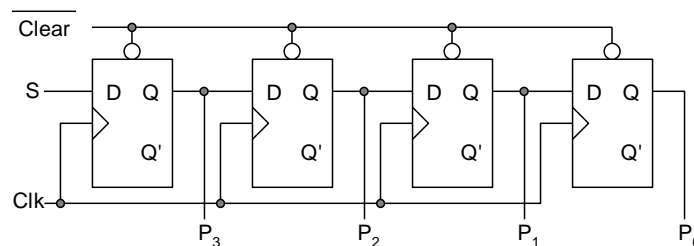


Figura 4: Registrador de deslocamento de 4 bits

⁶ A Seção 19.5 do Livro “Eletrônica Digital Moderna e VHDL” (autor: Volnei Pedroni, Ed. Campus, 2010) traz uma tabela (Figura 19.7) que explicita as condições para a inferência de registradores pelo compilador VHDL na atribuição de SIGNAL e VARIABLE.

	Universidade Federal de Minas Gerais
	Escola de Engenharia – Departamento de Engenharia Eletrônica
	Laboratório de Sistemas Digitais
	Detalhando alguns conceitos em VHDL – Prof. Hermes Aguiar Magalhães

ATIVIDADES PARA TREINAR O PROJETO DE FLIP-FLOPS E REGISTRADORES (OPCIONAL):

Parte I – Implementação dos *flip-flops*:

- Determine as tabelas de estado dos *flip-flops* D e T. A partir delas, faça as simplificações necessárias e determine as respectivas equações características. Programe estas equações em VHDL e simule ambos os *flip-flops* no Quartus II. No seu projeto, cada flip-flop deverá ter apenas a saída Q. Para cada tipo de *flip-flop*: apresente a tabela de estado, o processo de simplificação, a equação característica, o código VHDL e as formas de onda da simulação (estímulos e saídas), devidamente comentadas.
- Uma vez testados e funcionando, documente a estrutura interna da CPLD gerada pelo Quartus II (disponível em Tools > Netlist Viewers > RTL Viewer) PARA CADA UM dos tipos de *flip-flop* do item acima.

Parte II – Implementação de sinal de “CLEAR” e de registradores

- Programe em VHDL e simule temporalmente um *flip-flop* D com entrada de CLEAR síncrona. Apresente o código VHDL, acompanhado do respectivo circuito gerado (disponível após a compilação em Tools > Netlist Viewers > RTL Viewer) e das formas de onda de simulação (estímulos e saídas), devidamente comentadas.
- Programe em VHDL e simule temporalmente um *flip-flop* D com entrada de CLEAR assíncrona. Apresente o código VHDL, acompanhado do respectivo circuito gerado e das formas de onda de simulação (estímulos e saídas), devidamente comentadas.
- Escolha um dos tipos de *flip-flop* D acima – itens (c) ou (d) – e programe em VHDL o registrador de deslocamento de 4 bits da Figura 4. Dica: você pode usar o construto COMPONENT da linguagem VHDL para reaproveitar o módulo flip-flop D recém criado. Simule temporalmente o registrador. Apresente o código VHDL, acompanhado do respectivo circuito gerado e das formas de onda de simulação (estímulos e saídas), devidamente comentadas.

Modifique o registrador de deslocamento do item (e) acima, de forma que ele se comporte como um registrador que realiza uma rotação da palavra armazenada para a direita, de uma posição a cada evento de *clock*. Monitore as saídas P do circuito. Apresente o código VHDL, acompanhado do respectivo circuito gerado e das formas de onda de simulação (estímulos e saídas), devidamente comentadas.

- Modifique o circuito do item (e) para que apresente o comportamento temporal cíclico mostrado na Tabela 2. Apresente o código VHDL, acompanhado do respectivo circuito gerado e das formas de onda de simulação (estímulos e saídas), devidamente comentadas.

TABELA 2

0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0