Introdução ao VHDL

José A. Soares Augusto

Março 2006, V 1.0

O VHDL ("VHSIC Hardware Description Language", onde VHSIC significa "Very High Speed Integrated Circuits") é uma linguagem de descrição de hardware. É utilizado quando se pretende, quer simular um sistema digital a um nível (elevado) de abstracção para avaliar se o seu funcionamento corresponde às especificações, quer simplesmente descrevê-lo utilizando uma linguagem hoje em dia generalizada. Por exemplo, relativamente à questão da IP *('Intelectual Property')* de circuitos, que hoje em dia movimenta largas somas de dinheiro em *royalties*, é vulgar um projectista (ou uma companhia) negociar/vender um circuito, ou sistema, que apenas existe como um conjunto de ficheiros com a respectiva descrição em VHDL, os resultados da sua simulação, etc... Existem mais linguagens de descrição de hardware: por exemplo, o VERILOG.

A descrição em VHDL corresponde a um dos primeiros níveis na hierarquia do projecto de circuitos digitais na actualidade. Esta descrição pode ser feita segundo duas filosofias: a ***comportamental*** e a ***estrutural***.

Na descrição (ou especificação) **comportamental** de um circuito ou sistema, este é visto como uma 'caixa negra' que implementa uma determinada funcionalidade. Essa funcionalidade vai ser escrita em VHDL com um conjunto de instruções, de nível mais ou menos elevado (i.e., pode constar apenas de atribuições, mas pode, eventualmente, incluir selecção if-then-else, ciclos for ou while, etc...), que é perfeitamente válida em termos de programa mas que, eventualmente, pode ser impossível de sintetizar num circuito. Normalmente, num projecto o objectivo final é, precisamente, esta implementação a nível de circuito.

Na descrição **estrutural**, o circuito (ou sistema) é visto como consistindo de uma interligação de componentes (ou sub-circuitos) menos complexos que, por sua vez,
podem ser blocos construídos a partir de circuitos ainda mais básicos. Por exemplo, um registo de deslocamento (considerado como uma 'caixa preta' possuindo determinadas funções) é habitualmente construído com um certo número de células de memória (e.g., flip-flops) e com mais algumas portas lógicas. As memórias, por sua vez, são construídas a partir de portas lógicas básicas. Assim, a descrição estrutural desses componentes pode ser feita segundo a seguinte árvore:

Presume-se, neste cenário, que as portas já serão componentes básicos do circuito. Utilizando uma biblioteca de componentes (discretos ou integrados) disponibilizada por um fabricante, que habitualmente incluem dezenas ou centenas de portas típicas, a descrição estrutural do circuito seria facilmente sintetizável em silício.

A dualidade entre as descrições comportamental e estrutural de um circuito será ilustrada mais adiante com vários exemplos.

Uma das exigências frequentemente feitas a uma descrição VHDL é a de que esta seja sintetizável. Isto acontece sempre que se pretende não somente simular e testar um circuito, como também fabricá-lo. Neste caso, apenas um subconjunto das instruções da norma VHDL está disponível ao projectista. Este subconjunto depende, em cada caso, da ferramenta de síntese (e do fabricante do sistema CAD) utilizada. Existe um sistema de CAD, denominado Alliance, desenvolvido numa universidade francesa, que é de utilização livre: o objectivo deste sistema é precisamente a síntese de circuitos, pelo que apenas implementa um subconjunto bastante restrito da linguagem VHDL.

Software

Nesta descrição vai ser exemplificado o simulador de VHDL denominado **VHDL Simili** da companhia *Symphony EDA* (http://www.symphonyeda.com).
Este simulador é disponibilizado sem custos para aplicações não comerciais, o que o torna bastante atractivo. O simulador suporta quase integralmente a especificação do VHDL datada de 1993 (há anterior, datada de 1987). A 'suite' de programas (compilador de VHDL e simulador de VHDL) é simples de utilizar e inclui algumas bibliotecas de fabricantes de chips (e.g. Altera, Synopsis...), para além das bibliotecas 'standard do IEEE'. Em 1999 foi introduzida uma nova extensão ao VHDL, denominada VHDL-AMS, que permite modelar sistemas e circuitos mistos (i.e., analógico-digitais). Esta norma permite, por exemplo, especificar sistemas de equações diferenciais no código VHDL.

Um site com muita informação útil é http://tech-www.informatik.uni-hamburg.de/vhdl que tem links para o Alliance e para outros simuladores (não tem link para o SymphonyEDA). Pesquise também a directoria de VHDL no Google, http://directory.google.com/Top/Computers/Programming/Languages/VHDL. Outra excelente página, recheada de informação sobre VHDL, é http://members.nbci.com/vhdlweb/.

Convém referir que o objectivo deste documento é o de ensinar a simular (e a testar...) circuitos e sistemas electrónicos, bem como o de ensinar os métodos e algoritmos utilizados em teste e simulação. Esta determinação levou a que fosse escolhido um bom simulador de VHDL que suportasse quase integralmente a linguagem, preterindo-se os objectivos respeitantes à síntese dos circuitos.

Simulação com o sistema VHDL Simili

A utilização do simulador VHDL Simili é feita em dois passos. Primeiro, o ficheiro VHDL é compilado com o programa **vhdlp**; de seguida, se não houver erros, simula-se o modelo em VHDL invocando o simulador **vhdle**. O compilador vhdlp segue de perto (embora não estritamente, a especificação VHDL de 1993 (a mais importante para circuitos digitais).

O simulador vhdle não suporta nenhum mecanismo para a inclusão de estímulos (ou sinais de entrada) na simulação. Assim, tem de ser usado o conceito da "bancada de teste" **('test bench')** para explicitar os estímulos apropriados para a simulação. Estes estímulos podem ser criados por processos na própria descrição VHDL ou podem ser lidos vectores de teste gravados em ficheiros exteriores.

O **compilador vhdlp** é invocado segundo a seguinte sintaxe:

```
vhdlp [opções] ficheiro1 [ficheiro2 ... ficheiroN]
```

O compilador pode compilar mais do que um ficheiro de cada vez. Se estes não estiverem relacionados entre si, a ordem de compilação não é importante; porém, se estiverem relacionados, esta ordem já é importante.

As opções mais importantes são:

- **`-h`** mostra o 'help'.
- **`-work`** `nome-da-biblioteca`
  A biblioteca de trabalho nome-da-biblioteca é utilizada em vez da biblioteca de defeito "work". Por exemplo `vhdlp -work mylib myfile.vhd` compila o ficheiro myfile.vhd para a biblioteca `mylib`.
- **`-ini`** `ficheiro-de-inicialização`
  Esta opção utiliza o `ficheiro-de-inicialização` para especificar as bibliotecas utilizadas, que são especificadas no ficheiro de inicialização de defeito (symphony.ini na directoria ...\bin da instalação).
- As opções **-s** (modo silencioso aparece pouca informação na consola), **-87** (permite alguma compatibilidade com a norma obsoleta VHDL-87), -**strict** ou **-x** (que obrigam a seguir 'rigorosamente' a norma VHDL-93) só são necessárias para aplicações mais complexas (ou utilizadores mais picuinhas...).

O código de saída do compilador é 0 (zero) se a compilação fôr bem sucedida (útil para testar o sucesso de compilações a partir de 'scripts').

O **simulador vhdle** tem um leque de opções mais vasto. É invocado como

```
vhdle [options] top-level-design-unit
```

Por exemplo se a `top-level-design-unit` fôr '`myentity` e se pretendemos utilizar a arquitectura `myarch` dá-se o comando '`vhdle myentity(myarch)`'. Também pode utilizar-se uma configuração '`vhdle myconfig`'.

Alguns 'switches' de invocação do vhdle:

- **`-h`** mostra o 'help'.
- **`-s`** modo silencioso. A maior parte das mensagens é suprimida.

- **`-work library-name`**
  altera a biblioteca de defeito *work* para *library-name*. É necessário utilizá-la se o modelo de nível mais elevado não está na biblioteca *work*. Por exemplo *' vhdle -work mylib myentity '* permite simular o modelo *myentity* da biblioteca *mylib*.
- **`-p`** suprime a impressão de mensagens sobre o progresso da simulação (e.g., do tempo) na consola.
- **`-gName=Value`** altera valores dos genéricos (mesmo de valores de defeito) na entidade/arquitectura de nível mais elevado. (Não é válido se a entidade de topo é uma configuração). Exemplo:
  ```
  vhdle -gMyInt=32 -gMyTime=10.0ns -gMyReal=20.0 -gMyStr=hello myentity
  ```
- **`-GName=Value`** altera valores genéricos *a qualquer nível da hierarquia*.
- **`-do command-file`** executa os comandos no ficheiro *command-file*. É útil para monitorizar os sinais durante a simulação.
- **`-list list-file`** utilizada com o comando `-do` para escrever o relatório da simulação para o ficheiro *list-file*. Por defeito este ficheiro de saída é *'simili.lst'*. O switch `-list` é ignorado se não fôr utilizado `-do`.
- **`-t time[unit]`** simula somente durante o tempo especificado (interrompe 'test-benches' infinitas, i.e., cuja actividade não pára.) As unidades são *fs, ps, ns, us, ms, sec, hr* e *min*. A unidade de defeito é o *fs*. Não deve haver espaços entre 'time' e 'unit'. Exemplos:
  ```
  vhdle -t 10ns myentity   -- Simula 10 nanosegundos
  vhdle -t 100us myentity  -- Simula 100 microsegundos
  vhdle -t 100 us myentity -- !!!INVÁLIDO!!!
  vhdle -t 100 myentity    -- Simula 100 femtosegundos
  ```
- **`-stdin fileu`** redirecciona o ficheiro INPUT definido no módulo 'standard' TEXTIO para o ficheiro do utilizador **fileu** (tem que existir!)
- **`-stdout filename`** idem para o ficheiro pré-definido OUTPUT. Um anterior ficheiro **filename** é apagado sem aviso!
- **`-noaccel name`**, **`-nowarn name`** e **`-nocycleopt`** são opções que permitem alterar a aceleração das simulações e a optimização das bibliotecas.
- **`-breakon severity`** permite controlar qual o nível de severidade que faz parar a simulação (quando se usa `assert/report`). Por defeito é *failure*.

Introdução ao VHDL

Para discutir alguns aspectos importantes do VHDL, vai-se considerar um bloco lógico, com 3 entradas X1, X2 e X3 e uma saída Z.

FIG. 1 - Circuito BLOCOX ("caixa preta") com três entradas e uma saída.

ENTITY

Como se disse, podemos encarar o VHDL como sendo uma ferramenta de descrição de um projecto com o fim efectuar a sua síntese (ou compilação), ou utilizá-lo apenas para simulação. Para descrever o bloco da fig. 1 em VHDL utiliza-se o conceito de **ENTITY** (entidade), que nada mais é do que uma declaração da existência de uma entidade e uma especificação dos seus terminais.

O **VHDL não diferencia maiúsculas de minúsculas**. Neste documento vai-se seguir uma notação em que as palavras chave do VHDL (ou das bibliotecas standard) são escritas com maiúsculas, e os nomes específicos ao programa (e.g., os sinais e as variáveis) são escritos com minúsculas.

Então, a descrição em VHDL da entidade BLOCOX é:

```
-- Descrição da entidade BLOCOX

ENTITY blocox IS
     PORT ( x1, x2, x3 : IN BIT;
            z : OUT BIT );
END blocox;
```

Esta descrição apenas veicula a informação relativa à interface da entidade, nada indicando relativamente à sua função. Em VHDL o texto situado entre dois traços '--' e o fim da linha é considerado um comentário. A palavra PORT e a informação entre parênteses que se lhe segue indicam que o BLOCOX tem 3 entradas e uma saída, todas do tipo BIT. A sintaxe da especificação de uma entidade é clara neste pequeno exemplo, pelo que não vamos discuti-la.

ARCHITECTURE

Para implementar a função da entidade utiliza-se uma **ARCHITECTURE** (arquitectura). Em geral, existirão várias maneiras para descrever o funcionamento de uma entidade, a que corresponderão várias arquitecturas. As grandes famílias de arquitecturas, ou seja, tipologias de descrição de circuitos, são denominadas **descrição estrutural** e **descrição comportamental** *('behavioral').*

Para melhor explicar a filosofia que lhes está associada, e as diferenças entre as descrições comportamental e estrutural, vai-se desvendar o interior do BLOCOX, que consiste do circuito lógico na fig. 2. Ora o exame desta figura mostra-nos a estrutura do circuito, em termos de blocos lógicos básicos que, neste caso, são portas do tipo NAND. Então, a partir da figura pode-se escrever a arquitectura estrutural do BLOCOX.

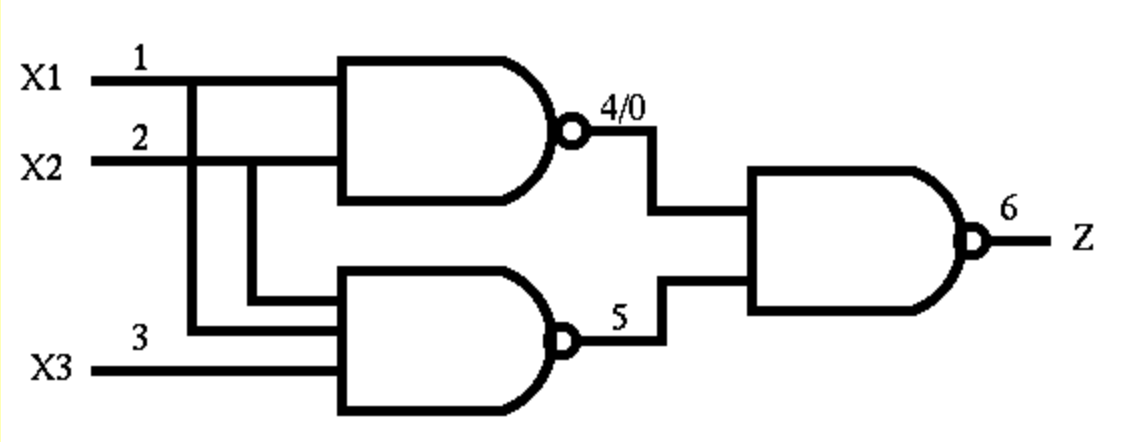

FIG. 2 - Esquema lógico do BLOCOX.

```
-- Arquitectura estrutural do BLOCOX

ARCHITECTURE estrutural OF blocox IS
     COMPONENT nand2
          PORT ( a, b : IN BIT ; y : OUT BIT);
     END COMPONENT;
     COMPONENT nand3
          PORT ( a, b, c : IN BIT ; y : OUT BIT);
     END COMPONENT;
     SIGNAL x4, x5 : BIT;
BEGIN
     U1: nand2 PORT MAP ( x1, x2, x4 );
     U2: nand2 PORT MAP ( x4, x5, z );
     U3: nand3 PORT MAP ( x1, x2, x3, x5 );
END estrutural;
```

Nesta pequena peça de código são visíveis alguns aspectos importantes. A área entre as linhas de ARCHITECTURE e BEGIN servem para declarar os componentes (neste caso nand2 e nand3) e os sinais (neste caso, os sinais internos no circuito da fig. 2, que denominamos de x4 e x5). Estas declarações utilizam, respectivamente, as palavras chave COMPONENT e SIGNAL do VHDL. No código VHDL situado entre BEGIN e END encontra-se o ‘esquemático’ do blocox, ou seja, a sua descrição estrutural. Este circuito consiste de três portas, U1, U2 e U3, instanciadas com a instrução PORT MAP.

Obviamente, existe uma correspondência posicional entre os sinais reais x1,...,x5 e z do esquemático e os sinais abstractos a, b, c, y que são utilizados na declaração dos componentes nand1 e nand2, semelhante à que se usa quando se invoca uma função ou um procedimento numa linguagem de programação comum. No entanto, pode-se fazer uma instanciação mais concreta (e não posicional) de componentes no 'port map'; por exemplo a linha

```
U1: nand2 PORT MAP ( x1, x2, x4 );
```

é equivalente a

```
U1: nand2 PORT MAP ( y => x4, b => x2, a=> x1 );
```

Se introduzir a entidade BLOCOX e a respectiva arquitectura estrutural num ficheiro denominado, obviamente, blocox.vhd e o compilar invocando o vhdlp (do VHDL simili) obtém:

```
C:>vhdlp blocox.vhd
Symphony EDA (R) VHDL Compiler/Simulator Module VhdlP, Version 1.5, Build#17.
Copyright(C) Symphony EDA 1997-2001. All rights reserved.
Reading G:\VHDLSymph\bin\symphony.ini ...
Library 'ieee'        ==> $SYMPHONY/Lib/Ieee/Ieee.sym (readonly)
Library 'work'        ==> work.sym
Parsing Entity:blocox @ line blocox.vhd:3
Writing  work.sym\blocox\prim.var
Parsing Architecture:blocox(estrutural) @ line blocox.vhd:11
Writing  work.sym\blocox\_estrutural.var
Elapsed Time: 00h:00m:00s:286ms
```

O circuito compilou sem erros nem avisos (*'errors'* ou *'warnings'*), o que significa que a sintaxe está correcta. Além disso, criou-se a sub-directoria *work.sym* (que pode ser vista como uma biblioteca de VHDL proprietária do projectista) que contém algums ficheiros resultantes da compilação do ficheiro blocox.vhd. O compilador foi buscar uma série de definições à biblioteca 'ieee.sym' que é uma biblioteca standard nos compiladores/simuladores de VHDL.

Se o sistema de CAD onde este código é escrito possuir bibliotecas que dão a representação eléctrica (por exemplo, em termos de MOSFETS) ou mesmo física (em termos de máscaras para implantar em silício) das portas nand2 e nand3, o nosso BLOCOX poderá ser de imediato utilizado. Em resumo, esta descrição estrutural é extremamente adequada para a síntese de circuitos.

Porém, o BLOCOX poderia ser descrito através da função lógica que implementa

$z = \sim (\sim(x1\ x2) \sim(x1\ x2\ x3))$

onde o '~' representa a negação lógica. Assim, uma arquitectura comportamental possível para o BLOCOX seria a seguinte

```
-- Arquitectura comportamental do BLOCOX

ARCHITECTURE comportamental OF blocox IS
BEGIN
    z <= NOT ( NOT (x1 AND x2) AND NOT (x1 AND x2 AND x3 ) )
         AFTER 2 ns;
END comportamental;
```

Esta arquitectura consiste precisamente da escrita da função lógica do BLOCOX. Existem dois pontos a realçar: primeiro, usa-se o sinal '<=' para fazer a atribuição de sinais; segundo, para modelar o bloco lógico mais de acordo com a realidade, introduziu-se um atraso de 2 ns na

atribuição de z. Este atraso é do tipo inercial (isso será discutido posteriormente). Se adicionarmos esta arquitectura ao ficheiro blocox.vhd e o compilarmos, obtém-se o seguinte resultado:

```
C:\>vhdlp blocox.vhd
Symphony EDA (R) VHDL Compiler/Simulator Module VhdlP, Version 1.5, Build#17.
Copyright(C) Symphony EDA 1997-2001. All rights reserved.
Reading G:\VHDLSymph\bin\symphony.ini ...
Library 'ieee'        ==> $SYMPHONY/Lib/leee/leee.sym (readonly)
Library 'work'        ==> work.sym
Parsing Entity:blocox @ line blocox.vhd:3
Writing  work.sym\blocox\prim.var
Parsing Architecture:blocox(estrutural) @ line blocox.vhd:11
Writing  work.sym\blocox\_estrutural.var
Parsing Architecture:blocox(comportamental) @ line blocox.vhd:27
Writing  work.sym\blocox\_comportamental.var
Elapsed Time: 00h:00m:00s:426ms
```

Comparando com a anterior compilação, verifica-se que neste caso foram analisadas ('parsed') duas arquitecturas (a estrutural e a comportamental) para o BLOCOX. Como resultado desta tarefa, foram escritos alguns ficheiros na directoria de trabalho *work.sym*.

Simulação

Vai-se agora discutir a simulação de um circuito ou sistema descrito em VHDL. Particulariza-se obviamente para o VHDL Simili, o simulador recomendado. Porém, este aspecto varia de simulador para simulador: por exemplo o Alliance permite criar ficheiros de estímulos num formato próprio. Recorde-se que o Alliance suporta apenas um subconjunto restrito da especificação VHDL pois é, fundamentalmente, um programa para a **síntese** de circuitos.

Voltamos a considerar de novo o BLOCOX. Queremos simulá-lo aplicando nas suas entradas X1, X2 e X3 os estímulos mostrados na fig. 3. Para isso, vai-se modificar a descrição comportamental do BLOCOX e introduzir directamente as formas de onda na arquitectura. No entanto, a entidade BLOCOX também tem que ser alterada para que seja possível "escrever" internamente nos terminais de entrada: x1, x2 e x3 vão passar a ser do tipo BUFFER BIT:

```
-- Descrição da entidade BLOCOX com entradas do tipo BUFFER BIT

ENTITY blocox IS
      -- Permite escrever nas entradas internamente
     PORT ( x1, x2, x3 : BUFFER BIT;
            z : OUT BIT );
END blocox;
```

A arquitectura incluindo a descrição dos sinais de entrada é:

```
-- Arquitectura comportamental do BLOCOX com estímulos

ARCHITECTURE comportamentalest OF blocox IS
BEGIN
     x1 <= '1' , '0' AFTER 10 ns, '1' AFTER 25 ns,
          '0' AFTER 30 ns;
     x2 <= '0' , '1' AFTER 5 ns, '0' AFTER 20 ns,
           '1' AFTER 35 ns, '0' AFTER 45 ns;
     x3 <= '0' , '1' AFTER 10 ns, '0' AFTER 30 ns,
           '1' AFTER 40 ns;
     z <= NOT ( NOT (x1 AND x2) AND NOT (x1 AND x2 AND x3 ) )
         AFTER 2 ns;
END comportamentalest;
```

É óbvia a forma de atribuição de valores lógicos (do tipo BIT) a sinais. Os únicos valores admitidos são '0' e '1' . Na descrição de cada forma de onda começa-se por especificar o seu valor inicial e, depois, registam-se os instantes de mudança no valor binário de cada sinal utilizando a palavra AFTER..

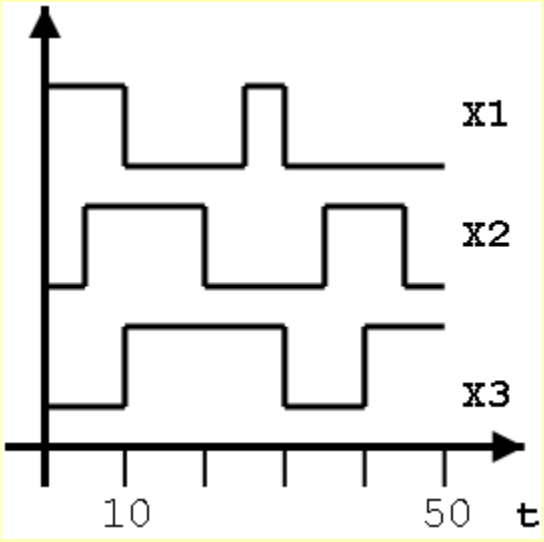

FIG. 3 - Estímulos aplicados ao BLOCOX (*t* em nanosegundos).

No simulador VHDL Simili tem que se pedir explicitamente os resultados da simulação. Isso é feito normalmente especificando um ficheiro de comandos com a opção **-do** do simulador. No presente caso esse ficheiro de comandos foi chamado **blocox.cmd** e consiste de

```
# comando de simulação do blocox
add list x1 x2 x3 z
```

A primeira linha é um comentário e a segunda indica que se pretende ver os resultados da simulação dos sinais nomeados na listagem. Neste caso pretende-se observar todos os sinais.

Eis a invocação do simulador e o respectivo relatório:

```
D:\>vhdle -do blocox.cmd blocox(comportamentalest)
Symphony EDA (R) VHDL Compiler/Simulator Module VhdlE, Version 1.5, Build#17.
Copyright(C) Symphony EDA 1997-2001. All rights reserved.
Reading G:\VHDLSymph\bin\symphony.ini ...
Library 'ieee'        ==> $SYMPHONY/Lib/leee/leee.sym (readonly)
Library 'work'        ==> work.sym
Reading  work.sym\blocox\_comportamentalest.var
      # of Signals      = 4
      # of Components   = 0
      # of Processes    = 5
      # of Drivers      = 4
Design Load/Elaboration Elapsed Time: 00h:00m:00s:248ms
Simulation stopped at: 47 ns
Simulation Elapsed Time: 00h:00m:00s:008ms
Time: 47 ns+0
```

Por defeito, o resultado da simulação é escrito num ficheiro denominado simili.lst. No entanto, este nome pode ser alterado com a opção **-list** do simulador vhdle. O resultado da presente simulação foi o seguinte:

```
                  x1  x3
           ps delta  x2   z
=========================
            0    +0 0 0 0 0
            0    +1 1 0 0 0
         5000    +0 1 1 0 0
         7000    +0 1 1 0 1
        10000    +0 0 1 1 1
        12000    +0 0 1 1 0
        20000    +0 0 0 1 0
        25000    +0 1 0 1 0
        30000    +0 0 0 0 0
        35000    +0 0 1 0 0
        40000    +0 0 1 1 0
        45000    +0 0 0 1 0
```

A unidade de tempo de simulação é o picosegundo, por defeito. Assim os instantes importantes de simulação são assinalados nesta unidade. No instante 0 são feitas 2 simulações (nos instantes 0 e 0+delta...) e este facto tem a ver com o 'motor' utilizado neste tipo de simulação: note que é necessário simular sistemas concorrentes, que sofrem alterações simultâneas em vários locais, como sejam as máquinas de estados (os flip-flops são todos estimulados simultaneamente com um mesmo relógio). Aquele instante 'delta' tem a ver com esta capacidade de simulação concorrente implementada nestes programas, mas abstemo-nos de detalhar mais este assunto.