Simulação em VHDL de Máquinas de Estados Finitas Hierárquicas

António Adrego da Rocha, Valery Sklyarov

Resumo **- Este artigo descreve a simulação em VHDL de máquinas de estados finitas hierárquicas. São referidos os resultados obtidos com a simulação do modelo descrito em [1], é introduzido um novo modelo e discutidas as suas vantagens. Ambos os modelos das máquinas de Mealy e Moore são considerados. O modelo considerado providencia novas facilidades, tais como flexibilidade, extensibilidade e reutilização de um algoritmo descrito por grafos hierárquicos.**

Abstract **- This paper discusses the VHDL simulation of Hierarchical Finite State Machines. It presentes the results obtained with the simulation of the model described in [1], introduces a new model and shows its advantages. Both Moore and Mealy machines are being considered. The considered model provides such new facilities as flexibility, extensibility and reuse of an algorithm described by Hierarchical Graph-Schemes.**

I. MÁQUINAS DE ESTADOS FINITAS HIERÁRQUICAS

Os **grafos hierárquicos** (Hierarchical Graph-Schemes **HGS**) foram inicialmente propostos em [2] e podem ser usados para descrever eficientemente o comportamento de unidades de controlo [1]. Podem ser implementados usando **máquinas de estados finitas hierárquicas** (**HFSM**) com *stack* [1]. A simulação em VHDL desse modelo, para ambas as máquinas de Mealy e Moore, prova que é possível combinar micro e macro instruções no mesmo nodo operacional se for usado o mecanismo de sincronismo sugerido em [1]. É ainda possível eliminar os estados de entrada dos vários HGS na máquina de Moore (estados $a_2,a_3,...,a_{V+2}$ atribuídos aos nodos ***Begin***) quando estes são seguidos por nodos operacionais.

O novo modelo apresentado na Figura 1 substitui o **Registo^h** do modelo anterior (Figura 3 de [1]) pelo **Conversor de Código**, cuja função é a de gerar o código do estado inicial do HGS que vai ser executado. Esta substituição implica uma alteração do comportamento do **Circuito Combinatório**. Nos estados onde é seleccionado um novo HGS (estados que contêm quer uma macro operação, quer uma função lógica) é atribuído o código binário 0 a todas as saídas $CSD_1,\ldots,CSD_R$. Nos restantes estados é atribuído o código binário 0 a todas as saídas $CCD_1,\ldots,CCD_R$. Este modelo tem as seguintes vantagens:

- Não é necessário re-alimentar o **Circuito Combinatório** com as entradas destinadas a identificar o HGS que vai ser executado, pelo que, o número total de entradas diminui.
- Muitas modificações na invocação de uma macro operação (função lógica) podem ser feitas no **Conversor de Código**, não havendo necessidade de modificar o *Kernel* do **Circuito Combinatório**.
- Este modelo providencia novas facilidades, tais como flexibilidade, extensibilidade e reutilização de um algoritmo descrito por grafos hierárquicos.

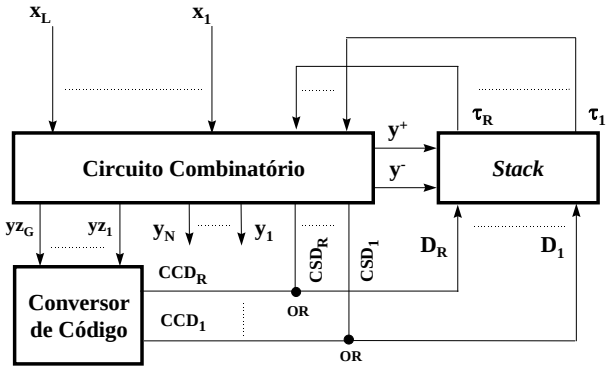

Figura 1 - Estrutura básica da máquina de estados finita hierárquica.

II. MÁQUINA DE ESTADOS HIERÁRQUICA DE MOORE

Para a máquina de estados hierárquica de Moore o **Circuito Combinatório** é decomposto no esquema apresentado na Figura 2.

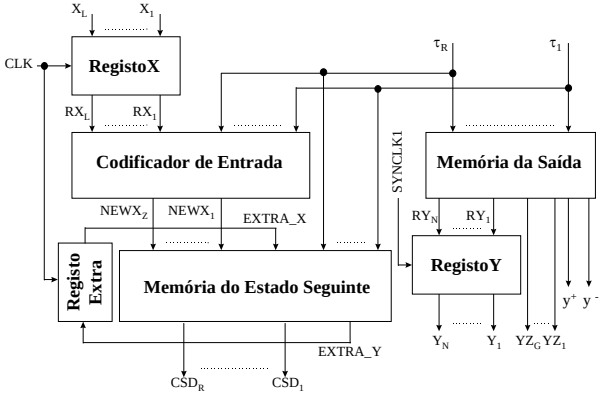

Figura 2 - Decomposição do circuito combinatório da máquina de Moore.

O **RegistoX** é responsável por fixar as variáveis de entrada. Na maioria dos casos, o número de variáveis de entrada que afectam uma transição de estado é muito pequena quando comparado com o número total L. Pelo

que, é feita uma transformação de variáveis de entrada no **Codificador de Entrada**, tal como é proposto em [3] (páginas 147 a 158), de forma a reduzir o número de linhas de endereçamento da **Memória do Estado Seguinte**. Este bloco é responsável pela geração do estado seguinte e de uma saída extra, o valor de retorno de uma função lógica, estabilizada pelo **RegistoExtra**. Este valor é testado em vez de uma qualquer função lógica existente num nodo condicional. Estamos a considerar que um HGS descritivo de uma função lógica deve ser marcado para síntese como uma máquina de Mealy, de forma a reduzir o número de estados usados e também para acelerar a sua execução. A **Memória da Saída** gera todas as restantes saídas (micro operações, código binário das macro operações, incremento e decremento do ponteiro do *stack*). O **RegistoY** é responsável por fixar as variáveis de saída (micro operações).

O mecanismo de sincronização proposto para esta máquina mista Moore/Mealy é apresentada na Figura 3 e baseia-se no mecanismo usado para o primeiro modelo (Figura 7 de [1]). O sinal que era responsável pela sincronização do **Registo^{h}** no modelo anterior, é agora usado para fixar a entrada **yz** do **Conversor de Código**. O valor de retorno da função lógica deve ser fixo durante a transição ascendente do relógio, onde ocorre a transição de estado e onde terminou o HGS que o calculou, de forma a manter este valor estável durante o próximo ciclo de relógio, onde vai ser responsável pela próxima transição de estado.

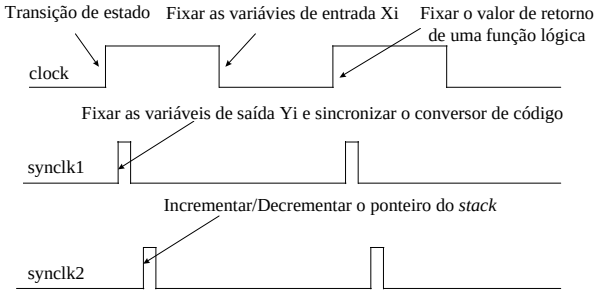

Figura 3 - Mecanismo de sincronização para a máquina mista Moore/Mealy.

O intervalo de tempo entre o relógio **clock** e o sinal de sincronismo **synclk1** deve ser escolhido de forma criteriosa para assegurar uma correcta estabilização das micro operações e do código binário das macro operações (entrada do **Conversor de Código**). Por correcta entenda-se, garantirmos que estamos a estabilizar os valores respeitantes ao estado actual e não ao estado anterior. Este intervalo de tempo deve ser superior à soma dos atrasos envolvidos na geração das saídas, ou seja, o atraso do ***Stack*** mais o atraso da **Memória da Saída**.

Devido às alterações introduzidas neste modelo, bem como aos resultados obtidos na simulação do primeiro modelo (onde se verifica que é possível eliminar alguns dos estados de entrada dos vários HGS) é necessário alterar as regras de marcação de um HGS. Para marcar um HGS para síntese como uma máquina de Moore (as funções lógicas devem ser marcado para síntese como máquinas de Mealy) é necessário executar os seguintes passos:

- A etiqueta $\mathbf{a_0}$ é atribuída aos nodos ***Begin*** e ***End*** do HGS principal $\mathbf{\Gamma_1}$;
- A etiqueta $\mathbf{a_1}$ é atribuída a todos os nodos ***End*** dos restantes HGS $\mathbf{\Gamma_2,...,\Gamma_V}$;
- As etiquetas $\mathbf{a_2,a_3,...,a_M}$ são atribuídas aos seguintes nodos e entradas: *1)* aos nodos ***Begin*** dos HGS $\mathbf{\Gamma_2,...,\Gamma_V}$, mas apenas se estes forem seguidos por nodos condicionais que não contêm funções lógicas; *2)* aos nodos rectangulares dos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$; *3)* às entradas de nodos losangulares dos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$ que contêm funções lógicas;
- É proibido repetir a mesma etiqueta nos vários HGS (com excepção de $\mathbf{a_0}$ e $\mathbf{a_1}$) e não é permitido marcar algum nodo ou entrada com mais do que uma etiqueta.

A Figura 4 apresenta um HGS (contendo os HGS $\Gamma_1,...,\Gamma_5$, e duas versões da função lógica Γ_6) marcado para síntese como uma máquina de Moore.

A Tabela 1 é a tabela estrutural ordinária obtida do HGS da Figura 4 e a partir da qual se modela o comportamento de alguns dos componentes em VHDL, nomeadamente o **Conversor de Código**, o **Codificador de Entrada**, a **Memória do Estado Seguinte** e a **Memória da Saída**.

Para melhor se compreender a sua construção é preciso ter em conta que os estados iniciais de cada HGS (Z_1 (a_2), Z_2 (a_7), Z_3 (a_{11}), Z_4 (a_{15}), Z_5 (a_{19}), θ_6^1 (a_{23}), θ_6^2 (a_{24})) são gerados pelo **Conversor de Código**. Pelo que, nos estados em que as macro instruções ou as funções lógicas são invocadas (Z_1 implicitamente em a_0 e Z_2, Z_3, Z_4, Z_5 e θ_6 explicitamente em a_3, a_5, a_6, a_9 e a_{12} respectivamente) é necessário gerar o código binário **yz** identificador de cada HGS (Z_1 (001), Z_2 (010), Z_3 (011), Z_4 (100), Z_5 (101), θ_6^1 (110), θ_6^2 (111)) e incrementar o ponteiro do *stack* (activar $\mathbf{y}^+$). Para Z_1 não é necessário incrementar o ponteiro do *stack*, uma vez que já estamos posicionados na primeira posição do stack e esta se encontra disponível, pelo que, o sinal $\mathbf{y}^+$ não é gerado. Nestes estados, em que o estado seguinte é gerado no **Conversor de Código**, o **Circuito Combinatório** deve gerar um estado seguinte com o código binário com todos os bits a zero (saídas $CSD_1,\ldots,CSD_R$ da Figura 1 todas a zero). Isto é conseguido da seguinte forma. O código decimal zero, código binário com todos os bits a zero, é reservado para o estado $\mathbf{a_0}$ (estado de arranque e estado final do HGS principal Z_1) e o código decimal um, código binário com todos os bits a zero excepto o bit menos significativo, é reservado para o estado $\mathbf{a_1}$ (estado final dos restantes HGS). Quando se invoca uma macro instrução ou função lógica, uma nova posição do *stack* vai ser usada. O valor existente nessa posição tem o código decimal zero (estado $\mathbf{a_0}$) caso essa posição ainda não tenha sido utilizada ou o código decimal um (estado $\mathbf{a_1}$) se essa posição do *stack* já foi anteriormente usada. Em ambos os casos é gerado o estado seguinte $\mathbf{a_0}$.

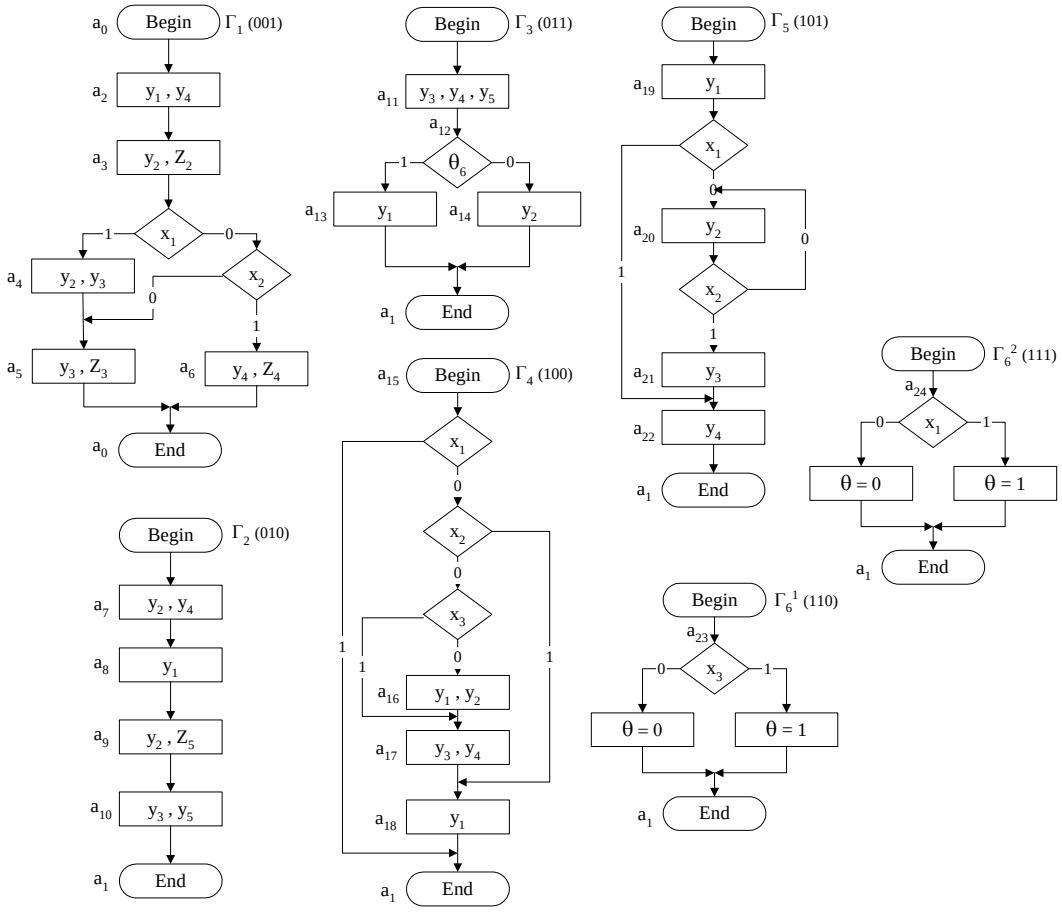

Figura 4 - HGS marcado para síntese como uma máquina de Moore.

Tabela 1

a_m / $Y(a_m)$	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$
a_0 , ($\mathbf{Z_1}$) $\mathbf{yz_1}$	a_0	1	-
a_1 , $\mathbf{y^-}$	a_0	1	-
a_2 , y_1 , y_4	a_3	1	-
a_3 , y_2 , ($\mathbf{Z_2}$) $\mathbf{yz_2}$, $\mathbf{y^+}$	a_4 a_5 a_6	x_1 $\bar{x}_1\bar{x}_2$ $\bar{x}_1 x_2$	- - -
a_4 , y_2 , y_3	a_5	1	-
a_5 , y_3 , ($\mathbf{Z_3}$) $\mathbf{yz_2}$, $\mathbf{yz_1}$, $\mathbf{y^+}$	a_0	1	-
a_6 , y_4 , ($\mathbf{Z_4}$) $\mathbf{yz_3}$, $\mathbf{y^+}$	a_0	1	-
a_7 , y_2 , y_4	a_8	1	-
a_8 , y_1	a_9	1	-
a_9 , y_2 , ($\mathbf{Z_5}$) $\mathbf{yz_3}$, $\mathbf{yz_1}$, $\mathbf{y^+}$	a_{10}	1	-
a_{10} , y_3 , y_5	a_1	1	-
a_{11} , y_3 , y_4 , y_5	a_{12}	1	-
a_{12} , (θ_6) $\mathbf{yz_3}$, $\mathbf{yz_2}$, $\mathbf{y^+}$	a_{13} a_{14}	$\overline{extra\_x}$ $extra\_x$	- -
a_{13} , y_1	a_1	1	-
a_{14} , y_2	a_1	1	-
a_{15}	a_1 a_{18} a_{17} a_{16}	x_1 $\bar{x}_1 x_2$ $\bar{x}_1\bar{x}_2 x_3$ $\bar{x}_1\bar{x}_2\bar{x}_3$	- - - -
a_{16} , y_1 , y_2	a_{17}	1	-
a_{17} , y_3 , y_4	a_{18}	1	-
a_{18} , y_1	a_1	1	-
a_{19} , y_1	a_{22} a_{20}	x_1 $\bar{x}_1$	- -
a_{20} , y_2	a_{21} a_{20}	x_2 $\bar{x}_2$	- -
a_{21} , y_3	a_{22}	1	-
a_{22} , y_4	a_1	1	-
a_{23}	a_1 a_1	x_3 $\bar{x}_3$	extra_y -

a_{24}	a_1 a_1	x_1 $\overline{x}_1$	extra_y -

Nos estados em que o estado seguinte é gerado pelo **Circuito Combinatório**, o **Conversor de Código** deve gerar um estado com o código binário com todos os bits a zero (saídas $CCD_1,\ldots,CCD_R$ da Figura 1 todas a zero). Para que tal aconteça, não é usada a codificação **000** como código binário identificador de nenhum HGS. E para essa codificação de **yz**, o **Conversor de Código** gera o estado $\mathbf{a_0}$. Para regressar ao HGS anterior é necessário decrementar o ponteiro do *stack* (activar $\mathbf{y}^-$), no estado final de qualquer HGS com excepção de Z_1, ou seja no estado $\mathbf{a_1}$. Para se determinar o estado seguinte num nodo condicional que contem uma função lógica (estado $\mathbf{a_{12}}$), é necessário testar a entrada **extra_x** que representa o valor calculado da função lógica $\boldsymbol{\theta_6}$. Nos estados onde se atribui o valor de retorno da função lógica, estados $\mathbf{a_{23}}$ em θ_6^1 e $\mathbf{a_{24}}$ em θ_6^2, é necessário activar a variável **extra_y** quando se pretende retornar o valor lógico **1**. Informação mais detalhada acerca de tabelas estruturais e sua utilização é descrita em [1], [3] e [4].

Vamos agora focar a nossa atenção na descrição em VHDL da HFSM. Todos os parâmetros usados na descrição dos componentes da HFSM estão especificados num ficheiro de parâmetros. O ficheiro inclui: a dimensão das variáveis da HFSM (número de entradas **L**, número de novas entradas após a transformação de variáveis **Z**, número de micro operações **N**, número de bits do código binário das macro operações **G**, número de bits da codificação de estado **R** e a dimensão do *stack*); a frequência do relógio; a posição relativa de cada sinal de sincronismo; a duração do sinal de sincronismo; atraso de cada componente; tipos de dados utilizado por cada componente; e a função **bitint**. Desta forma é possível simular uma nova HFSM sem necessidade de alterar o código VHDL de cada componente, bastando para o efeito alterar o ficheiro de parâmetros e o conteúdo das *look up tables*.

A HFSM é descrita de forma estrutural usando os seguintes componentes: Gerador do relógio e dos sinais de sincronismo; componente que faz o **or lógico** das saídas CSD e CCD (ver Figura 1); ***Stack***; **Conversor de Código** e **Circuito Combinatório**. Com excepção deste último que também é descrito de forma estrutural, todos os restantes componentes (incluindo os componentes do próprio **Circuito Combinatório**) são descritos de forma comportamental. Apresentamos de seguida uma breve descrição do comportamento dos componentes mais importantes e o respectivo código em VHDL.

O Gerador do relógio e dos sinais de sincronismo tem uma entrada de activação **enable** e gera o relógio **clk** e os sinais de sincronismo **synclk1** e **synclk2**, de acordo com o mecanismo de sincronização apresentado na Figura 3.

O componente **Or Lógico entre CCD e CSD** simula uma porta *or* lógica de R bits com um atraso DELORBLK.

Código VHDL do **Gerador do relógio e dos sinais de sincronismo**

```
entity CLKSYNCMOORE2 is
   Port (  ENABLE  : In  BIT;
           CLK     : Out  BIT;
           SYNCLK1: Out  BIT := '0';
           SYNCLK2: Out  BIT := '0'  );
end CLKSYNCMOORE2;

architecture BEHAVIORAL of CLKSYNCMOORE2 is
   signal periodic : bit := '0';
begin
   P1:process
   begin
      if ( ENABLE = '1' ) then
          periodic <= not periodic;
      end if;
      wait for CLKTIME;
   end process P1;

   P2:process ( periodic )
   begin
      CLK <= periodic;
      if ( periodic = '1' and periodic'EVENT ) then
          SYNCLK1<= '1' after DELSYNC1 ,
                    '0' after DELSYNC1+SYNCTIME;
          SYNCLK2<= '1' after DELSYNC2 ,
                    '0' after DELSYNC2+SYNCTIME;
      end if;
   end process P2;
end BEHAVIORAL;
```

Código VHDL do **Or Lógico entre CCD e CSD**

```
entity ORCCCSMOORE2 is
   Port (  CCD   : In  BIT_VECTOR (R downto 1);
           CSD   : In  BIT_VECTOR (R downto 1);
           ORD   : Out  BIT_VECTOR (R downto 1)  );
end ORCCCSMOORE2;

architecture BEHAVIORAL of ORCCCSMOORE2 is
begin
   ORD<=CCD or CSD after DELORBLK;
end BEHAVIORAL;
```

O ***Stack*** tem dois sinais de entrada de sincronização. O **clk** sincroniza a transição de estado. Na transição ascendente do relógio o estado apresentado à entrada do *stack* é armazenado na posição actual do *stack* e é colocado na saída **f** com o atraso DELSTACK. O **synclk** sincroniza o incremento ou decremento do ponteiro do *stack*. Na sua transição ascendente o ponteiro do *stack* é incrementado se o sinal $\mathbf{y}^+$ está activo, ou o ponteiro do *stack* é decrementado se o sinal $\mathbf{y}^-$ está activo. Se o ponteiro do *stack* for alterado, o valor armazenado na nova posição do *stack* é colocado na saída **f** com o atraso DELSTACK.

Código VHDL do ***Stack***

```
entity STACKMOORE2 is
  Port ( CLK      : In BIT;
         SYNCLK   : In BIT;
         DSTACK   : In BIT;
         ISTACK   : In BIT;
         D        : In BIT_VECTOR (R downto 1);
         F        : Out BIT_VECTOR (R downto 1) );
end STACKMOORE2;

architecture BEHAVIORAL of STACKMOORE2 is
  signal STACK : STACK_TYPE;
begin
  process ( CLK , SYNCLK )
    variable stpointer : NATURAL := 1;
  begin
    if ( CLK='1' and CLK'EVENT ) then
        STACK(stpointer)<=D;
        F<=D after DELSTACK;
    elsif ( SYNCLK='1' and SYNCLK'EVENT ) then
        if ( ISTACK='1' ) then
            stpointer := stpointer+1;
            F<=STACK(stpointer) after DELSTACK;
        elsif ( DSTACK='1' ) then
            stpointer := stpointer-1;
            F<=STACK(stpointer) after DELSTACK;
        end if;
    end if;
  end process;
end BEHAVIORAL;
```

O **Conversor de Código** tem um sinal de entrada de sincronização que fixa a entrada **yz**, e é modelado como uma *look up table* com **G** linhas de endereço e com palavras de **R** bits. Na transição ascendente de **synclk** o valor endereçado por **yz** é colocado na saída **d** com o atraso DELCCONV.

Código VHDL do **Conversor de Código**

```
entity CODECONVMOORE2 is
  Port ( SYNCLK  : In BIT;
         YZ      : In BIT_VECTOR (G downto 1);
         D       : Out BIT_VECTOR (R downto 1) );
end CODECONVMOORE2;

architecture BEHAVIORAL of CODECONVMOORE2 is
  signal CCRAM : CODECONV_TYPE :=

       ("00000","00010","00111","01011",
        "01111","10011","10111","11000");
begin
  process ( SYNCLK )
  begin
    if ( SYNCLK ='1' and SYNCLK'EVENT ) then
       D<=CCRAM ( bitint(YZ) ) after DELCCONV;
    end if;
  end process;
end BEHAVIORAL;
```

O **Circuito Combinatório** contem três registos **RegistoX**, **RegistoY** e **RegistoExtra**. O seu funcionamento é semelhante e o código VHDL difere apenas na dimensão dos sinais de entrada e de saída, e na transição utilizada (ascendente ou descendente) do sinal de sincronização (ver Figura 2 e Figura 3). A título de exemplo apresenta-se o código do **RegistoY**. Na transição ascendente de **synclk** a entrada **regy** é colocada na saída **y** com o atraso DELREGIY.

Código VHDL do **RegistoY**

```
entity REGYMOORE2 is
  Port ( SYNCLK  : In BIT;
         REGY    : In BIT_VECTOR (N downto 1);
         Y       : Out BIT_VECTOR (N downto 1) );
end REGYMOORE2;

architecture BEHAVIORAL of REGYMOORE2 is
begin
   process ( SYNCLK )
   begin
     if ( SYNCLK ='1' and SYNCLK'EVENT ) then
        Y<=REGY after DELREGIY;
     end if;
  end process;
end BEHAVIORAL;
```

O **Codificador de Entrada** é um circuito puramente combinatório e como tal, sempre que as entradas mudam a saída é recalculada. Ou seja, sempre que exista uma alteração nas variáveis de entrada **regx** ou no estado da máquina **state**, é colocado um novo valor na saída **newx** com o atraso DELINPEN. Esta parte do código do componente depende da tabela de transição de estados e tem de ser reprogramado para cada nova HFSM. O HGS em análise tem apenas três variáveis de entrada e existe um estado ($\mathbf{a_{15}}$) em que o estado seguinte depende de todas elas, pelo que, é impossível diminuir o número de variáveis de entrada pelo método proposto em [3]. Consequentemente, neste exemplo o **Codificador de Entrada** limita-se a colocar a entrada na saída com um atraso nulo (DELINPEN=0), ou seja, é como se não existisse no circuito.

Código VHDL do **Codificador de Entrada**

```
entity INPENCMOORE2 is
  Port ( REGX    : In BIT_VECTOR (L downto 1);
         STATE   : In BIT_VECTOR (R downto 1);
         NEWX    : Out BIT_VECTOR (Z downto 1) );
end INPENCMOORE2;

architecture BEHAVIORAL of INPENCMOORE2 is
begin
  process ( REGX , STATE )
  begin
     NEWX<=REGX after DELINPEN;
  end process;
end BEHAVIORAL;
```

A **Memória da Saída** é modulada como uma *look up table*, com **R** linhas de endereço e com palavras de **N+G+2** bits. Quando o estado actual **state** muda são gerados novos valores para as saídas (micro operações **y**, código binário das macro operações **yz**, incremento e decremento do ponteiro do *stack* **ist** e **dst** respectivamente) com o atraso DELOUTPM.

Código VHDL da **Memória da Saída**

```
entity  OUTPUTMOORE2 is
   Port (  STATE     : In  BIT_VECTOR (R downto 1);
           Y         : Out  BIT_VECTOR (N downto 1);
           YZ        : Out  BIT_VECTOR (G downto 1);
           IST       : Out  BIT;
           DST       : Out  BIT  );
end OUTPUTMOORE2;

architecture  BEHAVIORAL  of  OUTPUTMOORE2  is
    signal  OUTPUTRAM : OUTPUTMEM_TYPE :=

      ( "0000000100","0000000001","0100100000","0001001010",
        "0011000000","0010001110","0100010010","0101000000",
        "0000100000","0001010110","1010000000","1110000000",
        "0000011010","0000100000","0001000000","0000000000",
        "0001100000","0110000000","0000100000","0000100000",
        "0001000000","0010000000","0100000000","0000000000",
        "0000000000","0000000000","0000000000","0000000000",
        "0000000000","0000000000","0000000000","0000000000" );
begin
   process ( STATE )
       variable index : NATURAL;
   begin
      index := bitint ( STATE );
      Y<=OUTPUTRAM (index) (N+G+2 downto G+3)
                    after DELOUTPM;
      YZ<=OUTPUTRAM (index) (G+2 downto 3)
                    after DELOUTPM;
      IST<=OUTPUTRAM (index) (2)
                    after DELOUTPM;
      DST<=OUTPUTRAM (index) (1)
                    after DELOUTPM;
   end process;
end  BEHAVIORAL;
```

A **Memória do Estado Seguinte** é modulada como uma *look up table*, com **R+Z+1** linhas de endereço e com palavras de **R+1** bits. O estado seguinte depende do estado actual **astate** e das variáveis de entrada, ou seja, da saída **newx** do **Codificador de Entrada** e da entrada extra **extrax** que representa o valor de retorno da função lógica anteriormente calculada. Sempre que uma destas linhas de endereço muda é recalculado o novo endereço da memória e o seu conteúdo, estado seguinte **nstate** e a saída extra **extray** que representa o valor calculado para a função lógica, é colocado à saída com o atraso DELNEXST.

Para a simulação em VHDL utilizou-se o *software* da Synopsys em ambiente UNIX. Para melhor se identificar o estado actual da máquina foi utilizada uma codificação de estados binária e a visualização de todos os sinais vectoriais, com excepção do vector das variáveis de entrada, é feita em decimal.

Código VHDL da **Memória do Estado Seguinte**

```
entity NXSTATEMOORE2 is
   Port ( ASTATE  : In  BIT_VECTOR (R downto 1);
          NEWX    : In  BIT_VECTOR (Z downto 1);
          EXTRAX  : In  BIT;
          NSTATE  : Out BIT_VECTOR (R downto 1);
          EXTRAY  : Out BIT );
end NXSTATEMOORE2;

architecture BEHAVIORAL of NXSTATEMOORE2 is
   signal NEXTSTATERAM : NSTATEMEM_TYPE :=
      ("000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000110","000110","000110","000110","000110","000110","000110","000110",
       "000110","000110","000110","000110","000110","000110","000110","000110",
       "001010","001010","001000","001000","001100","001100","001000","001000",
       "001010","001010","001000","001000","001100","001100","001000","001000",
       "001010","001010","001010","001010","001010","001010","001010","001010",
       "001010","001010","001010","001010","001010","001010","001010","001010",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "010000","010000","010000","010000","010000","010000","010000","010000",
       "010000","010000","010000","010000","010000","010000","010000","010000",
       "010010","010010","010010","010010","010010","010010","010010","010010",
       "010010","010010","010010","010010","010010","010010","010010","010010",
       "010100","010100","010100","010100","010100","010100","010100","010100",
       "010100","010100","010100","010100","010100","010100","010100","010100",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "011000","011000","011000","011000","011000","011000","011000","011000",
       "011000","011000","011000","011000","011000","011000","011000","011000",
       "011100","011010","011100","011010","011100","011010","011100","011010",
       "011100","011010","011100","011010","011100","011010","011100","011010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "100000","100000","000010","000010","100100","100100","000010","000010",
       "100010","000010","000010","000010","100100","100100","000010","000010",
       "100010","100010","100010","100010","100010","100010","100010","100010",
       "100010","100010","100010","100010","100010","100010","100010","100010",
       "100100","100100","100100","100100","100100","100100","100100","100100",
       "100100","100100","100100","100100","100100","100100","100100","100100",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "101000","101000","101100","101100","101000","101000","101100","101100",
       "101000","101000","101100","101100","101000","101000","101100","101100",
       "101000","101000","101000","101000","101010","101010","101010","101010",
       "101000","101000","101000","101000","101010","101010","101010","101010",
       "101100","101100","101100","101100","101100","101100","101100","101100",
       "101100","101100","101100","101100","101100","101100","101100","101100",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000010","000010","000010","000010","000010","000010","000010","000010",
       "000011","000011","000011","000011","000011","000011","000011","000011",
       "000010","000010","000011","000011","000010","000010","000011","000011",
       "000010","000010","000011","000011","000010","000010","000011","000011",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000",
       "000000","000000","000000","000000","000000","000000","000000","000000");
begin
   process (NEWX , EXTRAX , ASTATE)
      variable  address : BIT_VECTOR (R+Z+1 downto 1);
      variable index : NATURAL;
   begin
      address := ASTATE&NEWX&EXTRAX;
      index := bitint ( address );
      NSTATE<=NEXTSTATERAM (index) (R+1 downto 2)
                     after DELNEXST;
      EXTRAY<=NEXTSTATERAM (index) (1)
                     after DELNEXST;
   end process;
end BEHAVIORAL;
```

De modo que, ao estado $\mathbf{a_n}$ corresponde a codificação decimal **n**. A Figura 5, que a seguir se descreve, apresenta a forma de onda obtida durante a simulação da máquina de Moore. A macro operação Z_1 começa a ser executada e o estado actual da máquina é armazenado na primeira posição do *stack*. Assim temos a seguinte sequência de estados, a_0 com saída 0, a_2 com saída 9 (y_1y_4), a_3 com saída 2 (y_2). Neste estado é invocada a execução da macro operação Z_2 e o estado actual da máquina é armazenado na segunda posição do *stack*. E temos os estados a_7 com saída 10 (y_2y_4), a_8 com saída 1 (y_1), a_9 com saída 2 (y_2). No estado a_9 é invocada a macro operação Z_5 e o estado actual da máquina é armazenado na terceira posição do *stack*. Devido ao valor do vector de entrada (010) temos a seguinte sequência de estados, a_{19} com saída 1 (y_1), a_{20} com saída 2 (y_2), a_{21} com saída 4 (y_3), a_{22} com saída 8 (y_4) e o estado final a_1, que provoca o regresso ao estado a_9 de Z_2. Segue-se o estado a_{10} com saída 20 (y_3y_5) e no estado final a_1 dá-se o regresso ao estado a_3 de Z_1. De novo em Z_1 e devido ao novo valor do vector de entrada (011) temos o estado a_4 com saída 6 (y_2y_3) e o estado a_5 com saída 4 (y_3). Neste estado é invocada a macro operação Z_3 e o estado actual da máquina é armazenado na segunda posição do *stack*. O estado inicial de Z_3 é estado a_{11} com saída 28 ($y_3y_4y_5$). O estado seguinte é o estado a_{12} com saída 0 onde é invocada a função lógica θ_6 (primeira versão dependente de x_3). A função lógica utiliza a terceira posição do *stack* para armazenar o estado a_{23} e como x_3 é igual a 1 a função retorna o valor 1 (ver sinal **extrax**). De volta ao estado a_{12} de Z_3 e com **extrax** igual a 1 temos o estado seguinte a_{13} com saída 1 (y_1). Z_3 termina e dá-se o regresso ao estado a_5 de Z_1, que termina a sua execução quando atinge o estado a_0.

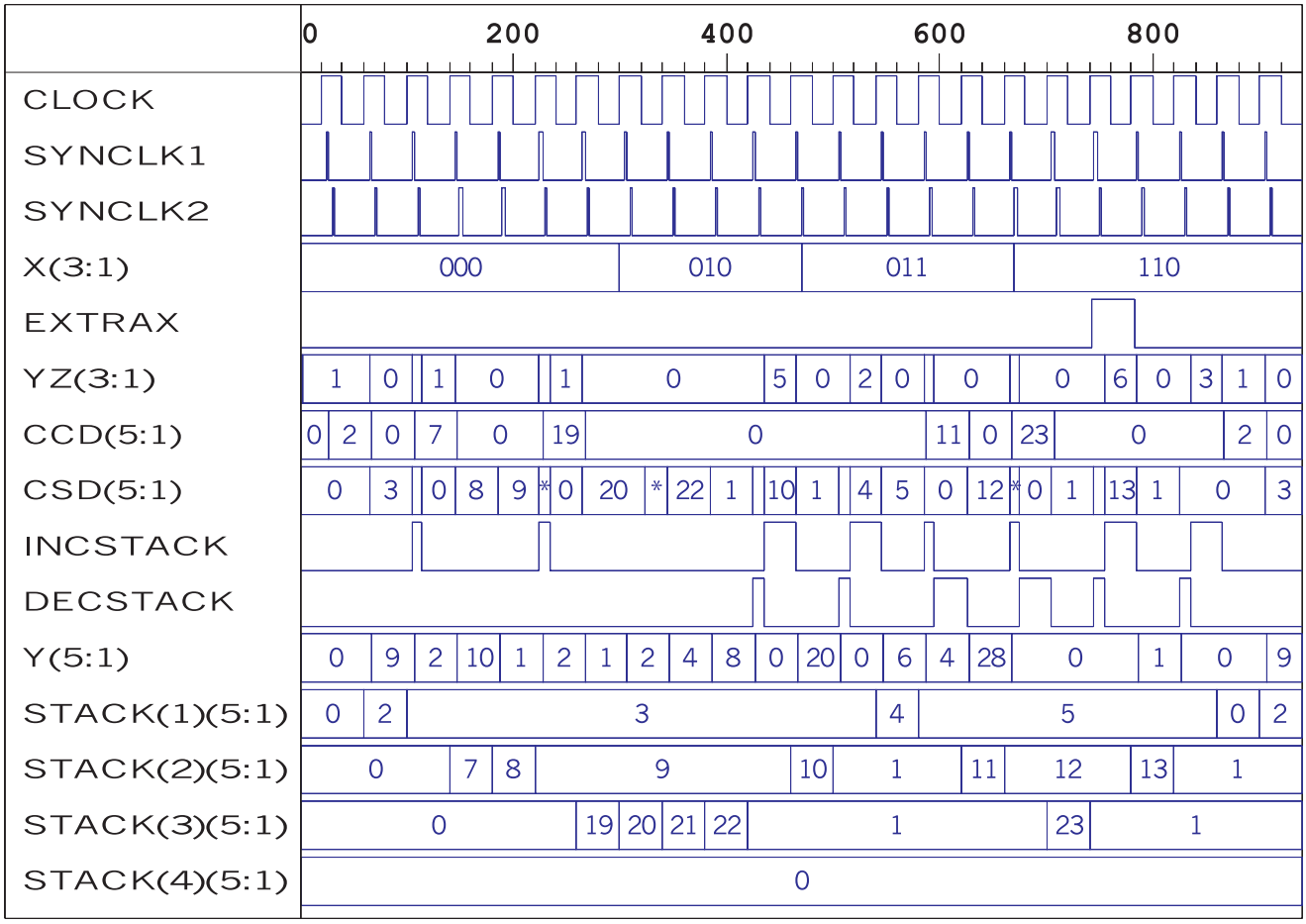

Figura 5 - Forma de onda gerada durante a simulação em VHDL da máquina de Moore.

III. MÁQUINA DE ESTADOS HIERÁRQUICA DE MEALY

Na máquina de estados hierárquica de Mealy as saídas são dependentes das entradas, pelo que, é necessário decompor o **Circuito Combinatório** de forma diferente da máquina de estados hierárquica de Moore. Como se pode ver na Figura 6, o estado seguinte e todas as saídas são geradas no bloco **Memória do Estado Seguinte e da Saída**.

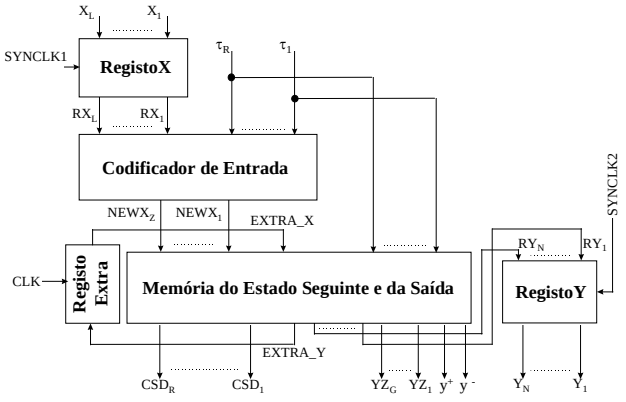

Figura 6 - Decomposição do circuito combinatório da máquina de Mealy.

Por outro lado temos que alterar a ordem de sincronização dos eventos da HFSM. O mecanismo de sincronização proposto para a máquina de Mealy é apresentado na Figura 7 e baseia-se no da Figura 3 com as seguintes alterações. O primeiro sinal de sincronismo é usado para fixar as variáveis de entrada. O segundo sinal

de sincronismo é responsável por fixar as variáveis de saída e sincronizar o **Conversor de Código**. O incremento ou decremento do ponteiro do *stack* é feito na transição descendente do relógio.

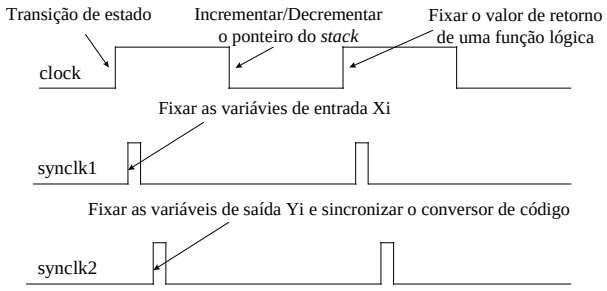

Figura 7 - Mecanismo de sincronização para a máquina de Mealy.

O intervalo de tempo entre o primeiro e o segundo sinal de sincronismo deve ser escolhido de forma criteriosa para assegurar uma correcta estabilização das micro operações e do código binário das macro operações (entrada do **Conversor de Código**). Por correcta entenda-se, garantirmos que estamos a estabilizar os valores respeitantes às variáveis de entrada actuais acabadas de fixar e não às variáveis de entrada anteriores. Este intervalo de tempo deve ser superior à soma dos atrasos envolvidos na geração das saídas, ou seja, o atraso do **RegistoX** mais o atraso do **Codificador de Entrada**, mais o atraso da **Memória do Estado Seguinte e da Saída**.

Devido às alterações introduzidas neste modelo é necessário alterar as regras de marcação de um HGS. Para marcar um HGS para síntese como uma máquina de Mealy é necessário executar os seguintes passos:

- A etiqueta $\mathbf{a_0}$ é usada no HGS principal $\mathbf{\Gamma_1}$ e é atribuída à entrada do nodo a que chega a seta proveniente do nodo ***Begin*** e ao nodo ***End***;
- A etiqueta $\mathbf{a_1}$ é atribuída a todos os nodos ***End*** dos restantes HGS $\mathbf{\Gamma_2,...,\Gamma_V}$;
- As etiquetas $\mathbf{a_2,a_3,...,a_M}$ são atribuídas às seguintes entradas: *1)* aquelas a que chegam setas provenientes dos nodos ***Begin*** nos HGS $\mathbf{\Gamma_2,...,\Gamma_V}$; *2)* às que são entradas de nodos rectangulares que contêm macro operações nos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$; *3)* aquelas a que chegam setas provenientes das saídas de nodos rectangulares nos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$; *4)* às que são entradas de nodos losangulares que contêm funções lógicas nos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$; *5)* aquelas a que chegam setas provenientes dos nodos losangulares que contêm funções lógicas nos HGS $\mathbf{\Gamma_1,...,\Gamma_V}$;
- É proibido repetir a mesma etiqueta nos vários HGS (com excepção de $\mathbf{a_0}$ e $\mathbf{a_1}$) e não é permitido marcar algum nodo ou entrada com mais do que uma etiqueta.

A Figura 8 apresenta um HGS (contendo os HGS $\Gamma_1,...,\Gamma_5$, e duas versões da função lógica Γ_6) marcado para síntese como uma máquina de Mealy. Verifica-se que o número de estados obtido com a marcação para síntese como uma máquina de Mealy não é significativamente inferior ao número obtido com a marcação para síntese como uma máquina de Moore (Figura 4). Para melhor se compreender o porquê da existência dos estados $\mathbf{a_{12}}$ e $\mathbf{a_{13}}$, regra de marcação número 5, é preciso ter em conta que só depois de executado o HGS da função lógica $\mathbf{\theta_6}$ é que se pode testar a entrada **extra_x**, que representa o valor calculado para a função lógica $\mathbf{\theta_6}$, e decidir qual o percurso seguinte. E em função desse percurso activar a micro operação $\mathbf{y_1}$ ou a micro operação $\mathbf{y_2}$. Sem o recurso a esses estados intermédios teríamos que activar uma micro operação ($\mathbf{y_1}$ ou $\mathbf{y_2}$) no estado $\mathbf{a_{11}}$, que depende do resultado da função lógica $\mathbf{\theta_6}$, mas sem ainda a ter executado.

A Tabela 2 é a tabela estrutural ordinária obtida do HGS da Figura 8 e a partir da qual se modela o comportamento de alguns dos componentes em VHDL, nomeadamente o **Conversor de Código**, o **Codificador de Entrada** e a **Memória do Estado Seguinte e da Saída**.

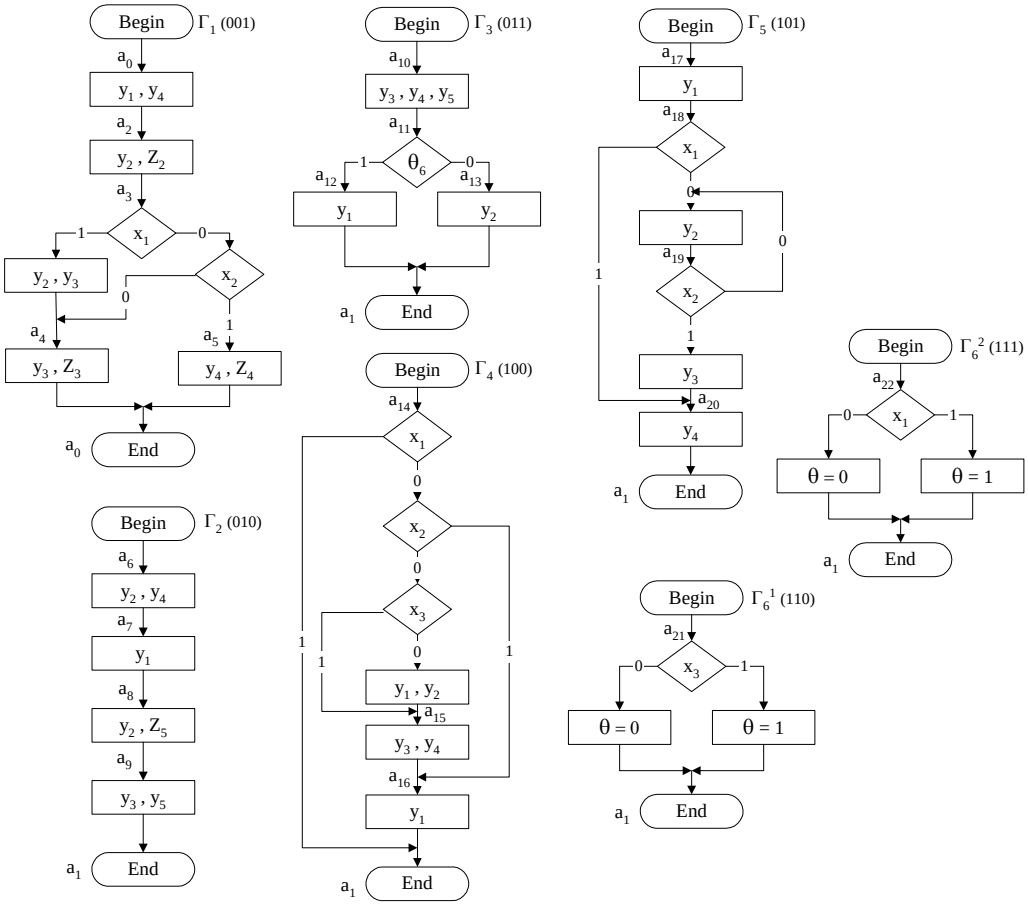

Figura 8 - HGS marcado para síntese como uma máquina de Mealy.

Tabela 2

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$
a_0	a_0	1	y_1 , y_4 , ($\mathbf{Z_1}$) $\mathbf{yz_1}$
a_1	a_0	1	$\mathbf{y^-}$
a_2	a_3	1	y_2 , ($\mathbf{Z_2}$) $\mathbf{yz_2}$, $\mathbf{y^+}$
a_3	a_4 a_4 a_5	x_1 $\bar{x}_1\bar{x}_2$ $\bar{x}_1 x_2$	y_2 , y_3 - -
a_4	a_0	1	y_3 , ($\mathbf{Z_3}$) $\mathbf{yz_2}$, $\mathbf{yz_1}$, $\mathbf{y^+}$
a_5	a_0	1	y_4 , ($\mathbf{Z_4}$) $\mathbf{yz_3}$, $\mathbf{y^+}$
a_6	a_7	1	y_2 , y_4
a_7	a_8	1	y_1
a_8	a_9	1	y_2 , ($\mathbf{Z_5}$) $\mathbf{yz_3}$, $\mathbf{yz_1}$, $\mathbf{y^+}$
a_9	a_1	1	y_3 , y_5
a_{10}	a_{11}	1	y_3 , y_4 , y_5
a_{11}	a_{12} a_{13}	$\overline{extra\_x}$ $extra\_x$	(θ_6) $\mathbf{yz_3}$, $\mathbf{yz_2}$, $\mathbf{y^+}$ (θ_6) $\mathbf{yz_3}$, $\mathbf{yz_2}$, $\mathbf{y^+}$

a_{12}	a_1	1	y_1
a_{13}	a_1	1	y_2
a_{14}	a_1 a_{16} a_{15} a_{15}	x_1 $\bar{x}_1 x_2$ $\bar{x}_1\bar{x}_2 x_3$ $\bar{x}_1\bar{x}_2\bar{x}_3$	- - - y_1 , y_2
a_{15}	a_{16}	1	y_3 , y_4
a_{16}	a_1	1	y_1
a_{17}	a_{18}	1	y_1
a_{18}	a_{20} a_{19}	x_1 $\bar{x}_1$	- y_2
a_{19}	a_{20} a_{19}	x_2 $\bar{x}_2$	y_3 y_2
a_{20}	a_1	1	y_4
a_{21}	a_1 a_1	x_3 $\bar{x}_3$	extra_y -
a_{22}	a_1 a_1	x_1 $\bar{x}_1$	extra_y -

A HFSM de Mealy é descrita em VHDL da mesma forma que a de Moore, mas a descrição estrutural do **Circuito Combinatório** tem menos um componente. Os sinais de sincronização dos componentes foram alterados em função da mudança da ordem de sincronização dos eventos. Assim sendo, o **RegistoX** que em Moore era sincronizado na transição descendente do relógio **clk**, em Mealy é sincronizado na transição ascendente do primeiro impulso de sincronismo **synclk1**. E o ***Stack*** passa a ter apenas um sinal de entrada de sincronização **clk**, mas sendo usadas ambas as transições ascendente e descendente do sinal. Apresenta-se apenas o código VHDL do ***Stack***.

Código VHDL do ***Stack***

```
entity  STACKMEALY2 is
   Port (  CLK       : In  BIT;
           DSTACK    : In  BIT;
           ISTACK    : In  BIT;
           D         : In  BIT_VECTOR  (R downto 1);
           F         : Out  BIT_VECTOR  (R downto 1)  );
end  STACKMEALY2;

architecture  BEHAVIORAL  of  STACKMEALY2  is
   signal  STACK : STACK_TYPE;
begin
   process ( CLK )
      variable stpointer : NATURAL := 1;
   begin
      if  ( CLK='1'  and  CLK'EVENT )  then
           STACK(stpointer)<=D;
           F<=D after DELSTACK;
      elsif   ( CLK='0'  and  CLK'EVENT )  then
           if  ( ISTACK='1' )  then
                stpointer := stpointer+1;
                F<=STACK(stpointer) after DELSTACK;
           elsif  ( DSTACK='1' )  then
                stpointer := stpointer-1;
                F<=STACK(stpointer) after DELSTACK;
           end if;
      end if;
   end process;
end  BEHAVIORAL;
```

A **Memória do Estado Seguinte e da Saída** é modulado como uma *look up table*, com **R+Z+1** linhas de endereço e com palavras de **R+N+G+3** bits. Em função do estado actual **astate** e das variáveis de entrada, ou seja, da saída **newx** do **Codificador de Entrada** e da entrada extra **extrax** que representa o valor de retorno da função lógica anteriormente calculada, são gerados novos valores para as saídas (estado seguinte **nstate**, saída extra **extray**, micro operações **y**, código binário das macro operações **yz**, incremento e decremento do ponteiro do *stack* **ist** e **dst** respectivamente) com o atraso DELNEXST. O código em VHDL não contem a *look up table*, devido à sua dimensão. A forma de onda obtida durante a simulação da máquina de Mealy é muito semelhante à da máquina de Moore, pelo que, não é apresentada.

Código VHDL da **Memória do Estado Seguinte e da Saída**

```
entity  NXSTOUTMEALY2 is
   Port (  ASTATE   : In  BIT_VECTOR (R downto 1);
           NEWX     : In  BIT_VECTOR (Z downto 1);
           EXTRAX   : In  BIT;
           NSTATE   : Out BIT_VECTOR (R downto 1);
           EXTRAY   : Out BIT;
           Y        : Out BIT_VECTOR (N downto 1);
           YZ       : Out BIT_VECTOR (G downto 1);
           IST      : Out  BIT;
           DST      : Out  BIT  );
end  NXSTOUTMEALY2;

architecture  BEHAVIORAL  of  NXSTOUTMEALY2  is
    signal  NEXTSTATERAM : NSTATEMEM_TYPE ;

begin
   process (NEWX , EXTRAX , ASTATE)
      variable  address : BIT_VECTOR (R+Z+1 downto 1);
      variable index : NATURAL;
   begin
      address := ASTATE&NEWX&EXTRAX;
      index := bitint ( address );
      NSTATE<=NEXTSTATEOUTRAM (index)
      (R+N+G+3 downto N+G+4)  after DELNEXST;
      EXTRAY<=NEXTSTATEOUTRAM (index) (N+G+3)
                    after DELNEXST;
      Y<=NEXTSTATEOUTRAM (index)
      (N+G+2 downto G+3)  after DELNEXST;
      YZ<=NEXTSTATEOUTRAM (index) (G+2 downto 3)
                    after DELNEXST;
      IST<=NEXTSTATEOUTRAM (index) (2)
                    after DELNEXST;
      DST<=NEXTSTATEOUTRAM (index) (1)
                    after DELNEXST;
   end process;
end  BEHAVIORAL;
```

III. NOVAS FACILIDADES

Se o **Conversor de Código**, e os componentes responsáveis por gerarem o estado seguinte e as variáveis de saída forem implementados com memórias do tipo RAM, então este modelo pode providenciar novas facilidades, tais como flexibilidade, extensibilidade e reutilização de um algoritmo descrito por grafos hierárquicos.

Por **flexibilidade** entenda-se a possibilidade de modificar o comportamento de um HGS rapidamente e com o mínimo de esforço. Suponhamos que um nodo operacional contem a macro operação Z_5. Vamos considerar que queremos alterar a invocação da macro operação Z_5 pela macro operação Z_4. Para o conseguir temos que reprogramar apenas o **Conversor de Código**, substituindo o estado endereçado pelo código binário de Z_5, ou seja o estado inicial da macro operação Z_5, pelo estado inicial da macro operação Z_4. Este procedimento permite-nos providenciar um nodo condicional flexível. Podemos desenvolver várias versões de uma função

lógica e depois alterar, se necessário, a versão a utilizar num HGS apenas pela simples alteração do estado inicial armazenado no **Conversor de Código**.

Vamos agora supor que queremos alterar completamente a funcionalidade de um nodo operacional. Por exemplo queremos alterar a invocação da macro operação Z_5 pela macro operação Z_4, e substituir a micro operação y_2 pela micro operação y_5 no estado a_9 do HGS Γ_2 da Figura 4. Esta modificação não é possível executar reprogramando apenas o **Conversor de Código**. Para obter o efeito desejado temos que reprogramar a **Memória da Saída**, alterando o vector de saída armazenado na posição de memória endereçada pelo estado a_9 (código binário 01001). Ou seja, o décimo vector da *look up table* da descrição em VHDL da **Memória da Saída** que é igual a "0001010110" deve ser substituído por "1000010010" para se obter o resultado pretendido.

Por **extensibilidade** entenda-se descrever um HGS e depois com a introdução de novos nodos operacionais, ou a eliminação de nodos operacionais já existentes, estender o seu comportamento de maneira a melhorar ou adaptar a sua funcionalidade. Suponhamos que pretendemos introduzir um novo nodo operacional, contendo a macro operação Z_4 e as micro operações y_1 e y_2 no HGS Γ_1 da Figura 4 ,entre os estados a_2 e a_3 como mostra a Figura 9.

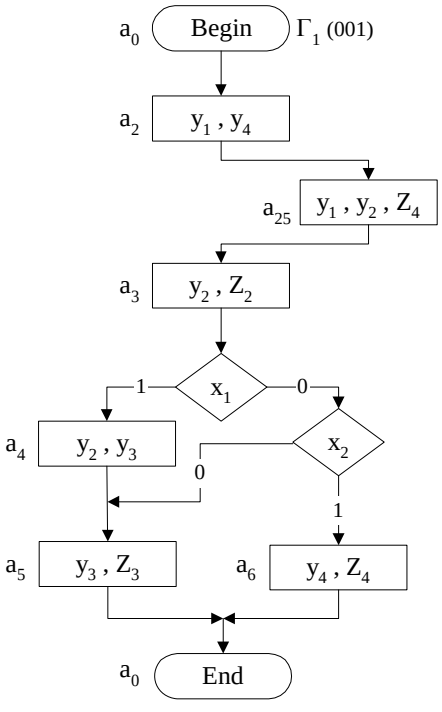

Figura 9 - Introdução de um nodo operacional na macro operação Z_1.

Em primeiro lugar temos que marcar este novo nodo operacional com um estado que esteja disponível, neste caso a_{25}. Depois é necessário reprogramar a **Memória do Estado Seguinte**. É preciso substituir a transição do estado a_2 para o estado a_3, pela transição do estado a_2 para o estado a_{25}, e acrescentar a transição do estado a_{25} para o estado a_3. Finalmente é preciso reprogramar a **Memória da Saída** alterando o conteúdo da posição de memória endereçada pelo estado a_{25} (código binário 11001). Ou seja, o vigésimo sexto vector da *look up table* que é igual a "0000000000" deve ser alterado para "0001110010".

Vamos considerar outro exemplo em que queremos mudar o nodo operacional marcado com o estado a_6 para a posição entre os nodos condicionais x_1 e x_2, como mostra a Figura 10.

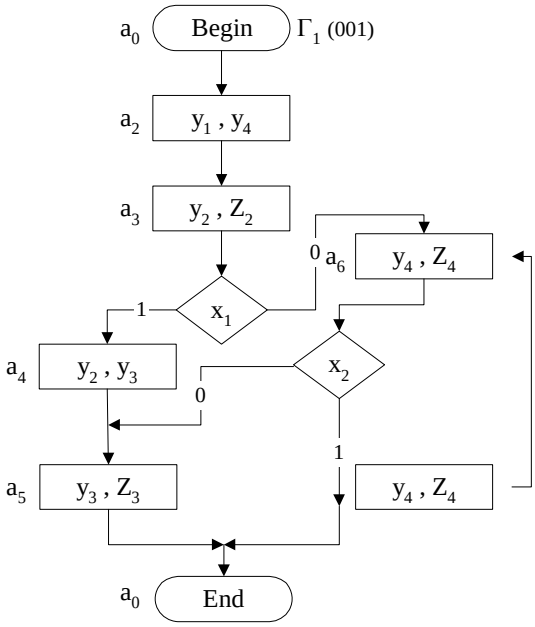

Figura 10 - Mudança de posição de um nodo operacional na macro operação Z_1.

Neste exemplo, apenas temos que reprogramar a **Memória do Estado Seguinte**, o que implica reavaliar as transições de estado para os estados a_3 e a_6. Como se pode ver na Figura 11, as transições do estado a_3 para o estado a_5 passam a ser transições do estado a_3 para o estado a_6, e metade das transições do estado a_6 para o estado a_0, passam a ser transições do estado a_6 para o estado a_5. Fazendo estas modificações nos respectivos vectores da *look up table* da descrição em VHDL da **Memória do Estado Seguinte** obtem-se o resultado pretendido.

a_3	a_4 a_5 a_6	x_1 $\overline{x}_1\overline{x}_2$ $\overline{x}_1 x_2$
a_6	a_0	1

→

a_3	a_4 a_6	x_1 $\overline{x}_1$
a_6	a_0 a_5	x_2 $\overline{x}_2$

Figura 11 - Reavaliação das transições de estado.

Por **reutilização** queremos dizer que podemos criar componentes reutilizáveis, macro operações ou funções lógicas separadas que poderão ser utilizadas em diferentes unidades de controlo que esperamos desenvolver no futuro. Isto pode ser feito investindo no desenho e construção de uma libraria de componentes que facilitarão o desenvolvimento de produtos semelhantes. Esta

abordagem é semelhante à utilizada na programação orientada a objectos. Em [5] (páginas 60 a 105) é apresentado um exemplo de desenvolvimento de componentes reutilizáveis para máquinas de estados finitas.

IV. CONCLUSÕES

Os **grafos hierárquicos** (**HGS**) fornecem uma representação multinível natural de algoritmos de controlo que podem ser vistos em vários níveis de abstracção. Eles fornecem uma boa separação da *interface* da unidade de controlo da sua *implementação*. É muito importante que suportem *hierarquia*.

O modelo proposto, denominado **máquinas de estados finitas hierárquicas** (**HFSM**), suporta uma decomposição *top-down* baseada na ordenação hierárquica de operações. Cada operação particular pode ser descrita por um HGS, que em geral **encapsula** variáveis de entrada e de saída (dados) e operações complexas (macro operações e funções lógicas) que podem ser vistas como **funções de controlo** (compare-se com o encapsulamento na programação orientada a objectos). Finalmente o encapsulamento permite separar a definição de uma instrução da sua implementação.

O modelo considerado providencia novas facilidades, tais como **flexibilidade**, **extensibilidade** e **reutilização** de um algoritmo descrito por grafos hierárquicos.

Para um HGS contendo muitas macro operações e funções lógicas, por vezes obtemos mais estados com a marcação para síntese como uma máquina de Mealy do que com a marcação para síntese como uma máquina de Moore. O que aliado a um mecanismo de sincronização mais sensível, torna o modelo da máquina de Mealy menos atractivo que o modelo da máquina de Moore.

Mesmo para um HGS com poucas variáveis de entrada, a dimensão da *look up table* da **Memória do Estado Seguinte** (no modelo da máquina de Moore) ou da **Memória do Estado Seguinte e da Saída** (no modelo da máquina de Mealy) é grande, pelo que, há a necessidade de desenvolver um novo modelo de implementação do **Circuito Combinatório** que permita reduzir o número de linhas de endereçamento da *look up table* ao mínimo, ou seja, a **R+1** bits.

REFERÊNCIAS

[1] Valery Sklyarov, António Adrego da Rocha. Síntese de Unidades de Controlo Descritas por Grafos dum Esquema Hierárquicos. Electrónica e Telecomunicações, 1996, N 6, pp 577-588.

[2] Valery Sklyarov. Hierarchical Graph-Schemes. Latvian Academy of Science, Automatics and Computers, Riga, 1984, N 2, pp 82-87.

[3] Samary Baranov. Logic Synthesis for Control Automata. Kluwer Academic Publishers, 1994.

[4] Valery Sklyarov. Applying Finite State Machine Theory and Object-Oriented Programming to the Logical Synthesis of Control Devices. Electrónica e Telecomunicações, 1996, N 6, pp 515-529.

[5] Robert C. Martin. Designing Object-Oriented C++ Applications Using the Booch Method. Prentice Hall, 1995.