NILO - Uma Linguagem para a Descrição de Hardware no nível de Portas Lógicas

por

Flávio Rech Wagner
Carla M. Dal Sasso Freitas

RP nº 66 MARÇO/87

Nota técnica do projeto "Banco de Dados e Ferramentas para CAD de Sistemas Digitais"

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Osvaldo Aranha, 99
90.210-Porto Alegre-RS-Brasil
Telex (051) 2680 Tel. (0512) 21.8499

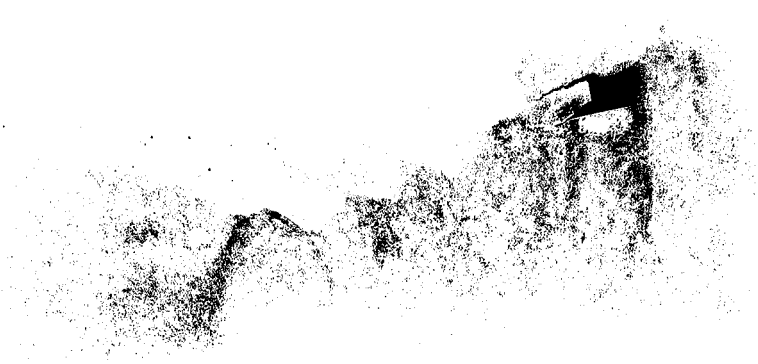
Endereço para Correspondência:
UFRGS/CPGCC/Biblioteca
Caixa Postal 1501
90.001-Porto Alegre-RS-Brasil

Microeletrônica - SBC/II

Linguagens: Descrição; Hardware

NILO

CNPq 3.04.03.00-6



RESUMO

Este relatório apresenta NILO, uma linguagem para descrição de sistemas digitais no nível de portas lógicas. NILO permite também a descrição de portas de transmissão bidirecionais, buffers "tri-state" e resistores "pull-up" e "pull-down". NILO faz parte de AMPLO, um ambiente integrado para o projeto de sistemas digitais.

PALAVRAS-CHAVE: linguagens de descrição de hardware, nível de portas lógicas

ABSTRACT

This report presents NILO, a language for describing digital systems at the gate level. NILO allows also the representation of bidirectional transmission gates, tri-state buffers and pull-up and pull-down resistors. NILO is part of AMPLO, an integrated environment for digital systems design.

KEYWORDS: hardware description languages, gate level

SUMARIO

1. INTRODUÇAO

O sistema AMPLO [WAG86c,WAG87c] é um ambiente integrado para o projeto lógico de sistemas digitais em desenvolvimento na UFRGS. Este ambiente possui ferramentas que permitem a descrição e validação de sistemas digitais em três níveis: sistema, transferência entre registradores (RT) e portas lógicas. Para cada um destes níveis existe uma linguagem de descrição de sistemas, respectivamente LASSO [BOR79], KAPA [WAG87b] e NILO, que é objeto deste relatório.

Sistemas são descritos de forma modular e hierarquizada como redes de agências [WAG84a,b], através da linguagem REDES [WAG87a]. Uma agência pode ser descrita de forma puramente estrutural, como uma composição de outras agências, utilizando-se a linguagem REDES, ou através de um comportamento especificado com uma das linguagens LASSO, KAPA ou NILO. Esta metodologia de projeto é apresentada em [WAG86a,c,WAG87c].

Todas as quatro linguagens suportadas pelo ambiente possuem formas gráfica e textual que são completamente equivalentes entre si [WAG86b]. Isto permite que a descrição de uma agência possa ser feita de forma textual, via editor de textos convencional e compilador, ou de forma gráfica, via um editor especializado para a linguagem. Tanto o compilador como o editor gráfico geram uma mesma estrutura de dados interna, que é independente da forma de entrada, exceto pelas informações geométricas manipuladas pelo editor. Esta representação interna é armazenada em uma base de dados unificada, de onde pode ser recuperada para a construção de modelos simuláveis.

NILO é uma linguagem para a descrição de sistemas digitais no nível de portas lógicas. Além das portas lógicas convencionais, no entanto, NILO possui construções para declarar portas de transmissão bidirecionais, buffers "tri-state" e resistores "pull-up" e "pull-down". A linguagem permite também a declaração de vários atrasos de propagação associados às portas, assim como de intensidades associadas às saídas das portas, o que é necessário na simulação de circuitos MOS. Com estas extensões, NILO se torna uma linguagem aplicável à descrição de sistemas que possam vir a ser implementados em diferentes tecnologias: circuitos discretos ou integrados, lógica TTL, CMOS ou NMOS, etc. O simulador associado à linguagem aceita quaisquer combinações de suas primitivas. Ele será objeto de relatório posterior a ser publicado.

Este relatório apresenta no capítulo 2 a estrutura geral da linguagem, que é idêntica a de todas as demais linguagens implementadas em AMPLO. No capítulo 3 é descrito como se declara a interface das agências. Os capítulos 4 e 5 apresentam respectivamente as primitivas para declaração dos componentes de uma agência e das interconexões entre estes componentes. O capítulo 6, finalmente, apresenta dois pequenos exemplos completos de descrição de agências: um flip-flop em tecnologia TTL e uma célula de memória RAM em tecnologia MOS. O apêndice contém a gramática completa da linguagem.

2. ESTRUTURA GERAL DA LINGUAGEM

A linguagem NILO apresenta a mesma estrutura geral comum a todas as demais linguagens de AMPLO (REDES, LASSO e KAPA). Todas possuem uma mesma sintaxe e semântica para as construções comuns a elas, que são "agency", "level" e "interface".

Descrições de agências podem criar tipos de agências na base de dados. Diferentes ocorrências destes tipos podem então ser utilizadas em descrições puramente estruturais, nas quais novos tipos de agências são criados e declarados como compostos por redes de agências. Descrições estruturais são feitas através da linguagem REDES [WAG87a].

A cada agência podem corresponder diferentes **alternativas** e **versões** de projeto [WAG87a]. Alternativas se distinguem por terem diferentes declarações para a interface da agência. A cada alternativa podem ser associadas várias versões de projeto, sejam elas estruturais, descritas em REDES, ou comportamentais, descritas em LASSO, KAPA ou NILO. Cada tipo de agência criado na base de dados corresponde a uma alternativa de projeto de uma agência. Qualquer versão desta alternativa pode ser utilizada no lugar de uma ocorrência deste tipo, já que todas as versões de uma mesma alternativa têm a mesma interface, o mesmo não valendo para versões de outras alternativas.

Toda descrição de uma agência deve portanto iniciar pela designação da agência, que é identificada pela construção

agency <nome_agência> . <nro_alternativa>. <nro.versão>

Esta descrição cria uma nova versão para a alternativa indicada (ou substitui uma versão anterior de mesmo número). Se esta for a primeira versão para esta alternativa, a alternativa também é criada.

A seguir deve ser declarado o nível no qual a agência está descrita, identificado pela construção

level = NILO.

Seguem-se a declaração da interface da agência, introduzida pela palavra-chave interface, e a declaração do comportamento da agência, introduzida pela palavra-chave behavior. A declaração da interface será vista no capítulo 3, enquanto a declaração do comportamento, isto é, dos componentes primitivos de uma agência (portas e resistores) e de suas interconexões, será vista nos capítulos 4 e 5.

A descrição de uma agência é encerrada pela construção

end ;

3. DECLARAÇÃO DA INTERFACE

A declaração da interface de uma agência segue o modelo abaixo, que é comum a todas as linguagens de AMPLO:

```
interface
  in  <desig_sinal>, ..., <desig_sinal> : <tipo_sinal> ;
      |
      <desig_sinal>, ..., <desig_sinal> : <tipo_sinal> ;
  out <desig_sinal>, ..., <desig_sinal> : <tipo_sinal> ;
      |
      <desig_sinal>, ..., <desig_sinal> : <tipo_sinal> ;
  inout <desig_sinal>, ..., <desig_sinal> : <tipo_sinal> ;
```

No modelo acima, <desig_sinal> é
<nome_sinal> <dimensão_sinal>,
onde <dimensão_sinal> pode ser omitida, se o sinal tiver uma largura de 1 bit, ou pode ser
[n1 : n2],
se o sinal for um vetor com largura de n bits, sendo n1 a designação do bit mais significativo e n2 a do bit menos significativo.

As 3 listas de sinais, iniciadas pelas palavras-chaves in, out e inout, identificam respectivamente os sinais de entrada, saída e bidirecionais da interface, e podem ser declaradas em qualquer ordem. É também possível a declaração de qualquer número de listas iniciadas por cada uma destas palavras-chave, e de qualquer número de sub-listas de sinais com um mesmo <tipo_sinal>.

No caso de NILO, apenas 3 tipos de dados existem para a declaração dos sinais de interface:
terminal,
bus e
clock.

O tipo "clock" só pode ser utilizado em sinais de saída. Sua função é identificar sinais que serão conectados a entradas de agências descritas em KAPA ou LASSO, e que são utilizados para sincronização de ações internas destas agências. Sinais de tipo "clock" podem ser ligados a entradas de tipo "terminal" de outras agências descritas em NILO [WAG87a].

O tipo "bus" pode ser utilizado tanto em sinais de entrada como bidirecionais. Devem ser declarados como bidirecionais e de tipo "bus" todos os sinais de interface que estiverem conectados internamente a saídas de buffers "tri-state" ou a pinos de dados de portas de transmissão bidirecionais. Sinais de entrada podem ser declarados de tipo "bus" quando se sabe que esta entrada será conectada a um sinal de outra agência que é de tipo "bus", embora isto não seja obrigatório, já que um sinal de interface bidirecional (de tipo "bus", portanto) pode ser conectado a um sinal de entrada de tipo "terminal" (ver [WAG87a]).

O tipo "terminal" é usado para identificar todos os sinais não bidirecionais e que não venham a ser usados como "clock" por outras agências.

Sinais no nível de portas lógicas não são normalmente agrupados em vetores. AMPLO, no entanto, é um ambiente integrado de projeto, no qual agências descritas em NILO podem ser conectadas a agências descritas em LASSO ou KAPA. Nestas outras duas linguagens, vetores de n bits são usados intensivamente. Para simplificar portanto a declaração de interconexões entre agências descritas em níveis diferentes, também em NILO é

possível identificar sinais da interface como vetores de n bits. Este tipo de designação não pode no entanto ser utilizado para sinais internos a uma agência descrita em NILO, onde apenas sinais de 1 bit podem ser declarados. Estes sinais internos podem ser conectados a bits dos sinais declarados como vetores na interface, através da construção "equivalence", o que será visto no capítulo 5.

4. DESCRIÇÃO DOS ELEMENTOS

A linguagem NILO permite a descrição de sistemas digitais no nível lógico como sendo uma interconexão de três tipos de elementos: portas lógicas, portas de transmissão bidirecionais e resistores.

4.1 Portas Lógicas

Portas lógicas são declaradas através da construção

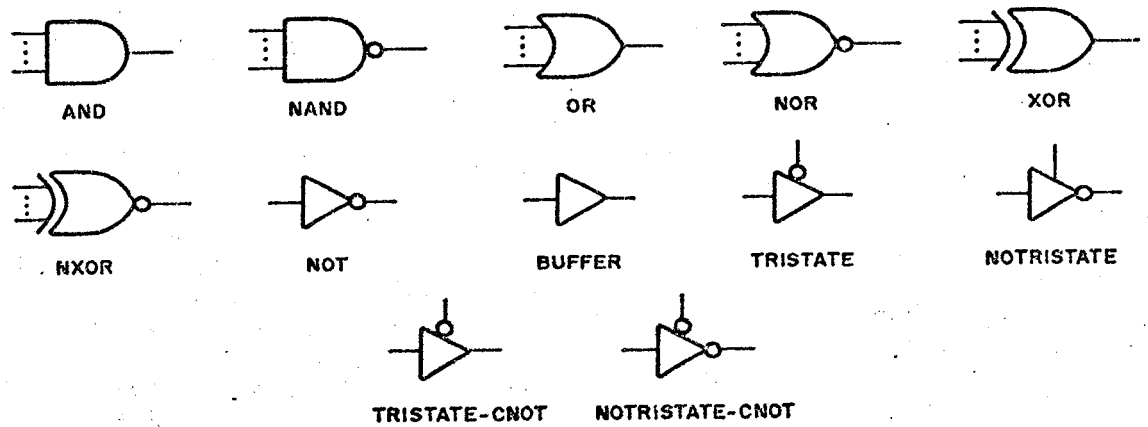
gate <nome-porta> (<atributos>) : <tipo-porta>.

Uma mesma construção gate pode servir a várias declarações de portas lógicas, de mesmo tipo ou de tipos diversos, como se pode verificar nos exemplos do capítulo 6. A sintaxe exata da construção se encontra no Apêndice.

O <tipo-porta> é uma das palavras-chave da tabela abaixo:

and	nand	or	nor
xor	nxor	not	buffer
tristate		notristate	
tristate-cnot		notristate-cnot	

A cada um destes tipos elementares está associado um símbolo gráfico especial, como mostrado abaixo. As portas de tipo not e buffer têm sempre uma única entrada, enquanto as portas de tipo "tristate" têm uma entrada de dados e uma entrada de controle. As portas dos demais tipos podem ter qualquer número de entradas, a partir de 2. Todos os tipos de portas têm uma única saída.



Cada porta lógica tem três tipos de atributos: a designação dos pinos de entrada e saída, a declaração dos atrasos de propagação ("delays") associados à porta, e a declaração da intensidade do sinal de saída da porta. A designação dos pinos deve necessariamente ser declarada antes dos demais atributos, que por sua vez podem ser colocados em qualquer ordem.

Os pinos de entrada e saída podem ser designados de forma implícita ou explícita. Na forma implícita, deve ser adotada a construção

in = n ;

onde n é o número de pinos de entrada, que serão identificados pelos números de 1 até n. O pino de saída será implicitamente designado pelo número n+1. No caso das portas de tipo "tristate", que possuem um pino de controle, este será identificado pelo número n+2.

Na forma explícita de designação dos pinos de entrada e saída, é associado um nome a cada pino, utilizando-se a construção

in = <nome-pino>, ..., <nome-pino> ;
out = <nome-pino> ;
contr = <nome-pino> ;

sendo que o pino de controle só existe nas portas de tipo "tristate".

A designação dos pinos de entrada e saída é necessária para as construções node e equivalence, a serem vistas no capítulo 5, que permitem a especificação das conexões entre elementos. Um pino, nestas construções, é designado por

<nome-porta> . <nome-pino>,

caso ele tenha sido designado explicitamente, ou por

<nome-porta> . <numero-pino>,

caso ele tenha sido designado implicitamente.

A cada porta devem ser necessariamente associados dois atrasos de propagação, nesta ordem:

- Tp01, que é o tempo de transição do valor lógico 0 para o valor lógico 1, e
- Tp10, que é o tempo de transição de 1 para 0.

No caso das portas de tipo "tristate", dois atrasos devem ser atribuídos além dos dois já mencionados:

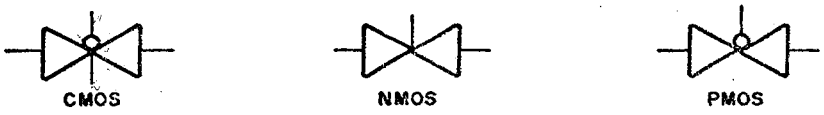
- Tpaz, que é o tempo de propagação de um dos estados ativos (0 ou 1) para o estado de alta impedância (z), e
- Tpza, que é o tempo de propagação do estado de alta impedância para um dos estados ativos.

As construções disponíveis em NILO para a declaração dos delays serão vistas no capítulo 4.4.

O último atributo a ser associado a uma porta é a intensidade do sinal de saída. Este atributo é no entanto obrigatório apenas quando a agência contém também portas de transmissão bidirecionais ou portas lógicas tipo "open-collector" ou similar. Neste caso, as intensidades dos sinais de saída dos elementos contidos na agência decidem o sentido de propagação de transições de valores lógicos através das portas bidirecionais, conforme será descrito em relatório posterior sobre o algoritmo de simulação. As formas de declaração da intensidade do sinal de saída de uma porta lógica serão vistas no capítulo 4.5.

4.2 Portas de Transmissão Bidirecionais

NILO permite a declaração de três tipos de portas de transmissão bidirecionais (também chamadas transistores de passagem): portas CMOS, NMOS e PMOS, às quais são associados os símbolos gráficos abaixo.



Portas CMOS possuem dois pinos de controle, um dos quais com lógica negada. Portas NMOS e PMOS possuem um único pino de controle, sendo este pino negado no caso das portas PMOS. Todos os três tipos de portas possuem dois pinos de dados, que são simétricos entre si.

Portas de transmissão são declaradas através da construção

transmission-gate <nome-porta> (<atributos>) :
<tipo-porta> ;

As palavras-chave tr-gate ou transistor podem ser usadas opcionalmente no lugar de transmission-gate.

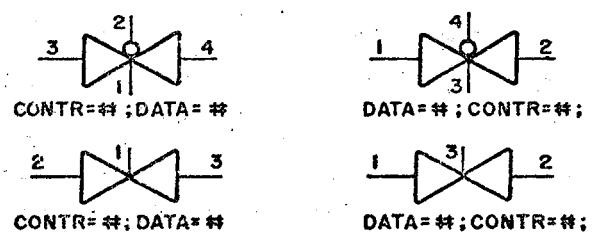
O <tipo-porta> é uma das palavras-chave cmos, nmos ou pmos.

Portas de transmissão possuem dois tipos de atributos: designação dos pinos e atrasos de propagação. Não 'e necessário atribuir-se intensidade aos sinais de dados de portas de transmissão.

Assim como para as portas lógicas, os pinos de portas de transmissão podem ser designados implicita ou explicitamente. A designação implícita é feita através da construção

contr = # ; data = # ;

No caso das portas CMOS, esta designação atribuirá o número 1 ao pino de controle com lógica não-negada, 2 ao pino de controle com lógica negada, e 3 e 4 aos pinos de dados. No caso de portas NMOSe PMOS, o pino de controle terá o número 1 e os pinos de dados os nmeros 2 e 3. A ordem das palavras-chave contr e data pode ser invertida, o que redundará em outra numeração para os pinos de controle e dados. A figura abaixo resume as duas formas de declaração para as portas CMOS e NMOS (ou PMOS) e as numerações de pinos resultantes.



Na forma explícita de designação dos pinos, um nome é associado a cada pino de controle e de dados, utilizando-se a construção

contr = <nome-pino-não-negado>, <nome-pino-negado> ;
data = <nome-pino>, <nome-pino> ;

no caso de portas CMOS, e

contr = <nome-pino> ;
data = <nome-pino>, <nome-pino> ;

no caso de portas NMOS.

Dois atrasos de propagação devem ser associados a cada porta de transmissão, nesta ordem:

- Tp01 (em qualquer um dos pinos de dados);
- Tp10 (em qualquer um dos pinos de dados).

4.3 Resistores

NILO permite a declaração implícita de portas lógicas de tipo "open-collector", "open-emitter" ou "open-drain", o que se obtém através da atribuição adequada de valores de intensidade aos sinais de saída das portas. Às saídas destes tipos de portas estão sempre conectados resistores "pull-up" ou "pull-down", que podem ser declarados através da construção

resistor <nome-resistor> (<designação-pinos>) ;

A designação de pinos, tal como no caso das portas lógicas e de transmissão, pode ser feita de maneira implícita ou explícita. A designação implícita é feita por

, gnd

no caso de resistores "pull-down", e por

vcc, #

no caso de resistores "pull-up". Em ambos os casos, o pino não conectado a "vcc" ou "ground" é designado implicitamente pelo número 1.

A designação explícita é feita por

<nome-pino>, gnd

no caso de resistores "pull-down", e por

vcc, <nome-pino>

no caso de resistores "pull-up".

4.4 Delays

Os atrasos de propagação, que precisam ser necessariamente associados às portas lógicas e de transmissão devido ao algoritmo de simulação adotado por NILO, podem ser declarados de três maneiras diferentes.

No primeiro modo, os valores dos atrasos (dois ou mais valores, de acordo com o tipo de porta, conforme foi visto nas seções anteriores) são declarados dentro da lista de atributos que se segue ao nome da porta, como no exemplo abaixo:

gate G1 (... ; delay = 8, 10) : nand ;

No segundo modo, a lista de atributos contém apenas um identificador de uma lista de valores. Os valores associados a este identificador são então declarados numa construção separada

da declaração da porta, como no exemplo abaixo:

```
gate G1 ( ... ; delay = D1 ) : nand ;
delay D1 = 8, 10 ;
```

Este mesmo identificador D1 pode ser utilizado então na declaração de diversas portas. Esta construção separada para atribuição de valores a um identificador de delay pode aparecer em qualquer ordem em relação às declarações **gate** e **transmission-gate**.

No terceiro modo, são associados valores "default" de delay a todas as portas de mesmo tipo. Nenhuma referência é feita aos atrasos dentro das declarações das portas. Os valores "default" são declarados por

```
delay <tipo-porta> = <lista-de-valores> ;
```

onde <tipo-porta> é qualquer um dos tipos de portas lógicas e de transmissão. No exemplo anteriormente utilizado, teríamos

```
delay nand = 8, 10 ;
```

Os três modos de declaração de delays podem ser utilizados simultaneamente numa mesma descrição. Neste caso, um valor "default" de delay atribuído a um certo tipo de porta pode ser sobrepujado por um valor diferente atribuído a uma certa porta deste tipo através de um dos outros modos de declaração. No exemplo

```
delay nor = 7, 9 ;
gate GG ( ... ; delay = 10, 12 ) : nor ;
```

as portas de tipo "nor" têm valores "default" de delay de 7 e 9 unidades de tempo. A porta GG, que é deste mesmo tipo, tem, no entanto, valores de atraso de 10 e 12 unidades de tempo.

Em todos os três modos de declaração de delays as palavras-chave del e d podem ser usadas em lugar de delay.

4.5 Declaração da intensidade dos sinais

O simulador NILO trabalha com um modelo de 5 estados lógicos básicos para os sinais: 0, L ("low"), Z (alta impedância), H ("high") e 1. Estes estados lógicos correspondem a 3 valores de intensidade de sinal:

forte (estados 0 e 1),
fraca (estados L e H) e
alta impedância (estado Z).

A intensidade de um sinal de saída de uma porta lógica é uma característica intrínseca, e portanto estática, desta porta. Portas de tipo "open-collector" e "open-emitter" apresentam, além disto, uma intensidade diferente para cada nível lógico na saída. Portas "open-collector", por exemplo, têm intensidade forte quando a saída está em nível baixo, e intensidade "alta impedância" quando a saída está em nível alto.

Em função destas características, a linguagem NILO permite a declaração de dois valores de intensidade para os sinais de saída das portas lógicas. Esta declaração pode ser feita de um de dois modos.

No primeiro modo, pode-se declarar a intensidade na lista de atributos de cada porta:

```
gate <nome-porta> ( ... ; intensity = <vb> , <va> ) :
                                                <tipo-porta> ;
```

onde

<vb> é a intensidade para o nível lógico "baixo", e
<va> é a intensidade para o nível lógico "alto".

Os valores <vb> e <va> podem ser designados numericamente, através de um valor inteiro entre 0 e 2, ou simbolicamente, através de uma das palavras-chave

strong (correspondente à intensidade 2),
weak (correspondente à intensidade 1) e
highimp (correspondente à intensidade 0).

No segundo modo, a lista de atributos da porta indica um identificador para a intensidade do sinal de saída da porta. Os valores associados a este identificador são atribuídos através de uma construção adicional. Exemplo:

```
gate G1 ( ... ; intensity = TTL_OC ) : nand ;
gate G2 ( ... ; intensity = TRI ) : tristate ;
intensity TTL_OC = strong, highimp ;
          TRI = strong, strong ;
```

Em todas estas situações, as palavras-chave int e i podem ser utilizadas no lugar de intensity.

5. DECLARAÇÃO DAS INTERCONEXÕES

No capítulo anterior vimos como declarar isoladamente os componentes de uma agência e seus atributos. Neste capítulo, veremos como declarar as interconexões entre componentes, através da construção node, e entre componentes e sinais de interface da agência, através da construção equivalence.

5.1 Nodos

Nodos são os pontos de interconexão entre pinos de componentes (portas lógicas e de transmissão e resistores). Nodos podem receber um nome, que pode posteriormente ser utilizado na declaração de interconexões com a interface da agência. A declaração de um nodo é feita através da construção

```
node <nome-nodo> = <id-pino> , ... , <id-pino> ;
```

ou

```
node <id-pino> , ... , <id-pino> ;
```

A identificação dos pinos conectados em um nodo segue o formato apresentado no capítulo anterior, isto é,

<nome-elemento> . <nome-pino>
caso os pinos do elemento tenham sido designados explicitamente, ou

<nome-elemento> . <numero-pino>
caso os pinos tenham sido designados implicitamente.

Nenhuma ordem específica precisa ser seguida na declaração dos pinos conectados em um nodo, não importando se eles são pinos de entrada ou de saída.

5.2 Equivalências

Equivalências são definições de identidade entre nodos e sinais de interface, e são declaradas pela construção

equivalence <sinal-interface> = <nome-nodo> ;

No caso de um pino de elemento conectado diretamente a um sinal de interface, a declaração do nodo é desnecessária, podendo-se utilizar diretamente a construção

equivalence <sinal-interface> = <id-pino> ;

Numa terceira forma da declaração de equivalência, é possível omitir-se a declaração de um nodo, fazendo-o implicitamente na própria definição da equivalência, caso em que o nodo não recebe nenhum nome:

equivalence <sinal-interface> = <id-pino> , ..., <id-pino> ;

Em todos estes casos, um sinal de interface pode ser identificado por uma de duas formas:

<nome-sinal>, ou

<nome-sinal> [n].

Este segundo caso é reservado àqueles sinais de interface declarados como vetores (ver capítulo 3), na forma

<nome-sinal> [n1 : n2].

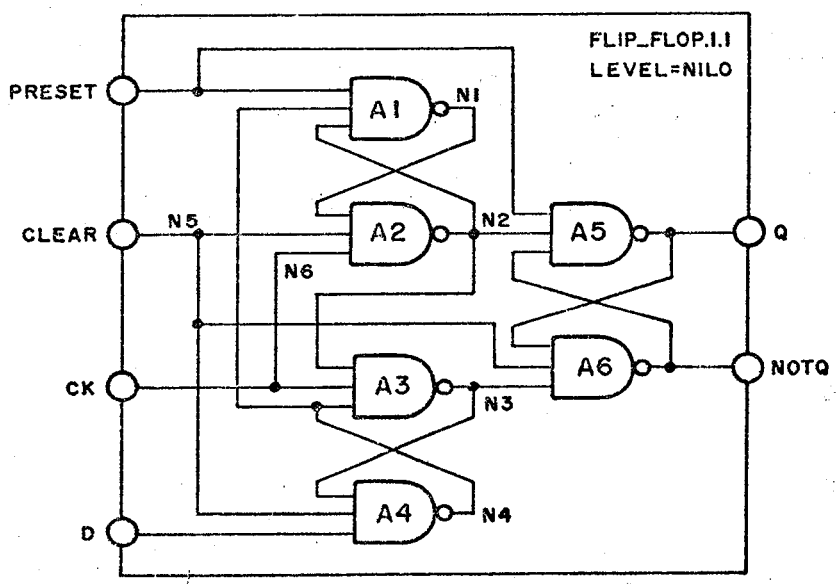
Neste caso, cada um dos bits do vetor da interface deve ser conectado a pinos ou nodos através de uma declaração de equivalência, que deve identificar por [n] o bit particular dentro do vetor.

6. EXEMPLOS COMPLETOS

6.1 Flip-flop tipo D

O exemplo abaixo mostra a descrição completa de uma agência que modela em NILO um flip-flop do tipo D, com entradas de PRESET e CLEAR e saídas negada e não-negada, implementado com lógica TTL.

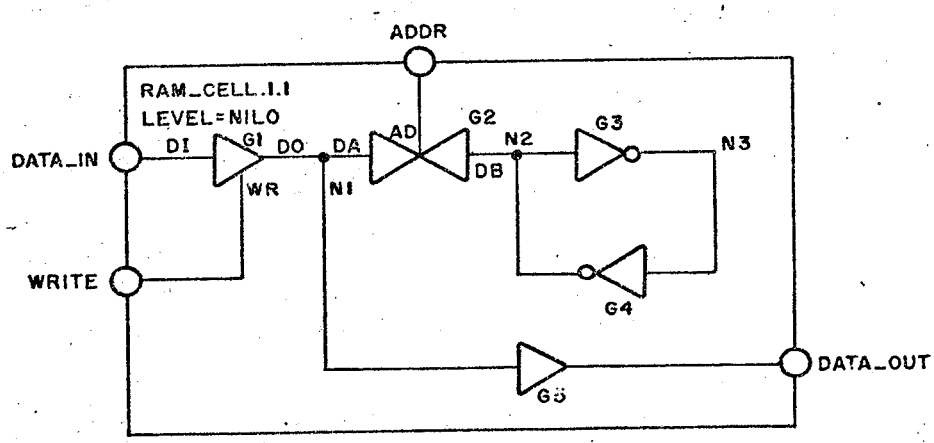
```
agency  FLIP_FLOP.1.1
level = NILO
interface
     in  PRESET, CLEAR, D, CK : terminal;
     out  Q, NOTQ : terminal;
behavior
     gate  A1 ( in = 3; del = D1 ),
           A2 ( in = 3; del = D1 ),
           A3 ( in = 3; del = D1 ),
           A4 ( in = 3; del = D1 ),
           A5 ( in = 3; del = D2 ),
           A6 ( in = 3; del = D2 ) : nand ;
     delay  D1 = 10, 15 ;
            D2 = 8, 13 ;
     node  N1 = A1.4, A2.1 ;
           N2 = A2.4, A5.2, A1.3, A3.1 ;
           N3 = A3.4, A6.3, A4.1 ;
           N4 = A4.4, A3.3, A1.2 ;
           N5 = A2.2, A6.2, A4.2 ;
           N6 = A2.3, A3.2 ;
     equivalence  PRESET = A1.1, A5.1 ;
                  CLEAR = N5 ;
                  CK = N6 ;
                  D = A4.3 ;
                  Q = A5.4, A6.1 ;
                  NOTQ = A6.4, A5.3 ;
end;
```



6.2 Célula de memória MOS

O segundo exemplo mostra a descrição completa de uma agência que modela uma célula de memória RAM de 1 bit, implementada em lógica MOS.

```
agency RAM_CELL.1.1
level = NILO
interface
     in  ADDR, DATA_IN, WRITE : terminal;
     out  DATA_OUT : terminal ;
behavior
     gate
       G1 ( in = DI; contr = WR; out = DO; int = TRI ) : tristate;
          G3 ( in = #; del = D1, int = TTL ),
          G4 ( in = #; del = D1; int = TTL ) : not;
          G5 ( in = #; del = D2; int = TTL ) : buffer;
     tr-gate G2 ( contr = AD; data = DA, DB ) : nmos;
     delay D1 = 6, 10;
           D2 = 5, 9;
           tristate = 6, 8, 10, 10;
           nmos = 6, 2;
     intensity TTL = strong, strong;
               TRI = strong, strong;
     node N1 = G1.DO, G2.DA, G5.1;
          N2 = G2.DB, G3.1, G4.2;
          N3 = G3.2, G4.1;
     equivalence DATA_IN = G1.DI; WRITE = G1.WR;
                 ADDR = G2.AD; DATA_OUT = G5.2;
end;
```



Agradecemos a Leandro F. Rey e Ney L. V. Calazans, pela valiosa contribuição na definição dos elementos e seus atributos.

REFERÊNCIAS BIBLIOGRAFICAS

[BOR79] BORRIONE,D. e J.F.GRABOWIECKI. "Informal Introduction to LASSO: A Language for Asynchronous Systems Specification and Simulation". In: P.A.Samet (ed), EURO-IFIP Z9. North-Holland Publ. Co., Amsterdam, 1979. pp 419-426

[WAG84a] WAGNER,F.R. "Modelamento de Processos Digitais com Redes de Instâncias". Porto Alegre, CPGCC-UFRGS, Mar. 1984. 20p. (Relatório Técnico 006)

[WAG84b] WAGNER,F.R. "Modelamento e Simulação de Processos Digitais Usando Redes de Instâncias". In: Seminário Integrado de Software e Hardware, XI. Viçosa, MG, 21-28 julho 1984. Anais, UFV, 1984. pp 309-318

[WAG86a] WAGNER,F.R. "A Multi-level Digital Systems Simulator based on Nets of Agencies:. In: JSST Conference on Recent Advances in Simulation of Complex Systems. Toquio, Japão, 15-17 julho 1986. Proceedings, 1986. pp 125-130

[WAG86b] WAGNER,F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "Equivalência de Descrições Textuais e Gráficas de Sistemas Digitais num Ambiente de CAD". In: Seminário Integrado de Software e Hardware, XIII. Recife, PE, 19-25 julho 1986. Anais, UFPe, 1986. pp 486-493

[WAG86c] WAGNER,F.R. et al. "Ambiente Integrado para Projeto de Sistemas Digitais Auxiliado por Computador". In: Congresso Nacional de Informática, XIX. Rio de Janeiro, RJ, 18-25 agosto 1986. Anais, Vol. 2, SUCESU, 1986. pp 111-116

[WAG87a] WAGNER,F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "Linguagens de Descrição de Hardware para Suporte à Integração do Processo de Projeto em AMPLO". Porto Alegre, CPGCC - UFRGS, Mar. 1987. 17p (Relatório Técnico 065)

[WAG87b] WAGNER,F.R. "KAPA - Uma Linguagem para a Descrição de Hardware no Nível de Transferência Entre Registradores". Porto Alegre, CPGCC - UFRGS, 1987. (Relatório Técnico, a ser publicado)

[WAG87c] WAGNER,F.R., C.M.D.S.-FREITAS e L.G.GOLENDZINER. "A Digital Systems Design Methodology based on Nets of Agencies". Aceito para publicação em: 8th International Symposium on Computer Hardware Description Languages and their Applications. Amsterdam, Holanda, 27-29 abril 1987

APENDICE. GRAMATICA NILO

Nota: A primeira versão desta gramática foi especificada pelo aluno Carlos Carravetta, como trabalho de aula da disciplina Compiladores I da UFRGS, sob orientação da Profa. Ana Price.

Convenções

< >	simbolos não-terminais
!	alternativas
* *	delimitadores de comentários
negrito	simbolos terminais

```
<descricao-agencia> ::= agency <designacao-agencia> <nivel>
                        <interface> <comportamento> end ;

* identificação da agência e do nivel de descrição *

<designacao-agencia> ::= <designacao-tipo-agencia> .
                         <numero-versao>
<designacao-tipo-agencia> ::= <nome-tipo-agencia> .
                              <numero-alternativa>
<nome-tipo-agencia> ::= <identificador>
<numero-alternativa> ::= <inteiro>
<numero-versao> ::= <inteiro>
<nivel> ::= level = NILO

* declaração da interface *

<interface> ::= interface <declaracao-interface>
<declaracao-interface> ::= <direcao> <lista-declar-sinais> !
          <direcao> <lista-declar-sinais> <declaracao-interface>
<direcao> ::= in ! out ! inout
<lista-declar-sinais> ::= <declaracao-sinais> !
                          <declaracao-sinais> <lista-declar-sinais>
<declaracao-sinais> ::= <lista-sinais> : <nome-tipo-sinal> ;
<lista-sinais> ::= <sinal> ! <sinal> , <lista-sinais>
<sinal> ::= <identificador> !
            <identificador> [ <inteiro> : <inteiro> ]
<nome-tipo-sinal> ::= terminal ! clock ! bus

* declaração do comportamento da agência *

<comportamento> ::= <lista-declar-componentes>
                    <lista-declar-conexoes>
<lista-declar-componentes> ::= <declar-componentes> !
                  <declar-componentes> <lista-declar-componentes>
<declar-componentes> ::= <descricao-gates> !
                         <descricao-transistores> !
                         <descricao-resistores> !
                         <descricao-delays-default> !
                         <descricao-intensidades>
```

```
* declaração de portas lógicas *

<descricao-gates> ::= gate <lista-declar-gates>
<lista-declar-gates> ::= <declaracao-gates> |
                         <declaracao-gates> <lista-declar-gates>
<declaracao-gates> ::=
      <lista-gates-simples> : <tipo-gate-simples> ; |
      <lista-gates-tristate> : <tipo-gate-tristate> ; |
      <lista-buffers> : <tipo-buffer> ;


* declaração de portas lógicas simples *

<lista-gates-simples> ::= <gate-simples> |
                          <gate-simples> , <lista-gates-simples>
<gate-simples> ::= <nome-gate> ( <lista-atributos-simples> )
<nome-gate> ::= <identificador>
<lista-atributos-simples> ::= <lista-explic-implic> |
     <lista-explic-implic> ; <lista-delay-intensidade>
<lista-explic-implic> ::= <lista-explicita> | <lista-implicita>
<lista-explicita> ::= in = <nome-pino> , <lista-pinos> ;
                      out = <nome-pino>
<lista-pinos> ::= <nome-pino> |
                  <nome-pino> , <lista-pinos>
<nome-pino> ::= <identificador>
<lista-implicita> ::= in = <numero-pinos-entrada>
<numero-pinos-entrada> ::= <inteiro>


<lista-delay-intensidade> ::=
    <decl-delays> | <decl-intensidade-gate> |
    <decl-intensidade-gate> ; <decl-delays> |
    <decl-delays> ; <decl-intensidade-gate> |

<decl-intensidade-gate> ::= <id-intensidade> = <valor-intensidade>
<id-intensidade> ::= intensity | int | i
<valor-intensidade> ::= <intensidade> , <intensidade> |
                        <nome-intensidade>
<intensidade> ::= <simbolo-intensidade> | 0 | 1 | 2
<simbolo-intensidade> ::= strong | weak | highimp
<nome-intensidade> ::= <identificador>

<decl-delays> ::= <id-delay> = <valor-delay>
<id-delay> ::= delay | del | d
<valor-delay> ::= <lista-delays> | <nome-delay>
<lista-delays> ::= <delay> | <delay> , <lista-delays>
<delay> ::= <inteiro>
<nome-delay> ::= <identificador>

<tipo-gate-simples> ::= and | nand | or | nor | xor | nxor


* declaração de portas lógicas "tri-state" *

<lista-gates-tristate> ::= <gate-tristate> |
```

```
                                <gate-tristate> , <lista-gates-tristate>
<gate-tristate> ::= <nome-gate> ( <lista-atributos-tristate> )
<lista-atributos-tristate> ::= <lista-explic-implic-tristate> |
   <lista-explic-implic-tristate> ; <lista-delay-intensidade>
<lista-explic-implic-tristate> ::= <lista-explic-tristate> |
                                   <lista-implic-tristate>
<lista-explic-tristate> ::= in = <nome-pino> ;
                            out = <nome-pino> ;
                            contr = <nome-pino>
<lista-implic-tristate> ::= in = #

<tipo-gate-tristate> ::= tristate | notristate |
                         tristate-cnot | notristate-cnot


* declaração de outras portas com uma entrada *

<lista-buffers> ::= <buffer> | <buffer> , <lista-buffers>
<buffer> ::= <nome-gate> ( <lista-atributos-buffer> )
<lista-atributos-buffer> ::= <lista-explic-implic-buffer> |
      <lista-explic-implic-buffer> ; <lista-delay-intensidade>
<lista-explic-implic-buffer> ::= <lista-explic-buffer> |
                                 <lista-implic-buffer>
<lista-explic-buffer> ::= in = <nome-pino> ;
                          out = <nome-pino>
<lista-implic-buffer> ::= in = #
<tipo-buffer> ::= buffer | not


* declaração de portas de transmissão ( transistores ) *

<descricao-transistores> ::= <id-transistor>
                             <lista-declar-transistores>
<id-transistor> ::= transmission-gate | tr-gate | transistor
<lista-declar-transistores> ::= <declar-transistores> |
          <declar-transistores> <lista-declar-transistores>
<declar-transistores> ::= <lista-trans-cmos> : cmos ; |
                          <lista-trans-npmos> : <npmos> ;
<npmos> ::= nmos | pmos


* declaração de transistores CMOS *

<lista-trans-cmos> ::= <transistor-cmos> |
                       <transistor-cmos> , <lista-trans-cmos>
<transistor-cmos>  ::= <nome-transistor>
                       ( <lista-atributos-cmos> )
<nome-transistor> ::= <identificador>
<lista-atributos-cmos> ::= <declar-pinos-cmos> |
                           <declar-pinos-cmos> <decl-delays>
<declar-pinos-cmos> ::=
     <declar-pinos-controle> ; <declar-pinos-dados> ; |
     <declar-pinos-dados> ; <declar-pinos-controle> ;
<declar-pinos-controle> ::=
     contr = <nome-pino> , <nome-pino> |
```

```
     contr = #
<declar-pinos-dados> ::=
     data = <nome-pino> , <nome-pino> |
     data = #


* declaração de transistores NMOS e PMOS *

<lista-trans-npmos> ::= <transistor-npmos> |
                        <transistor-npmos> , <lista-trans-npmos>
<transistor-npmos> ::= <nome-transistor>
                       ( <lista-atributos-npmos> )
<lista-atributos-npmos> ::= <declar-pinos-npmos> <decl-delays>
<declar-pinos-npmos> ::=
     <declar-pino-controle> ; <declar-pinos-dados> ; |
     <declar-pinos-dados> ; <declar-pino-controle> ;
<declar-pino-controle> ::= contr = <nome-pino> |
                           contr = #


* declaração de resistores *

<descricao-resistores> ::= resistor <lista-declar-resistores> ;
<lista-declar-resistores> ::= <declar-resistor> |
                      <declar-resistor> , <lista-declar-resistores>
<declar-resistor> ::= <nome-resistor> <designacao-pinos-resistor>
<nome-resistor> ::= <identificador>
<designacao-pinos-resistor> ::=
          ( <id-pino-resistor> , gnd ) |
          ( vcc , <id-pino-resistor> )
<id-pino-resistor> ::= <nome-pino-saida> | #


* declaração de delays identificados por valores default *

<descricao-delays-default> ::= <id-delay> <lista-decl-delays-def>
<lista-decl-delays-def> ::= <decl-delay-default> |
                     <decl-delay-default> <lista-decl-delays-def>
<decl-delay-default> ::=
     <nome-delay> = <lista-delays> ; |
     <tipo-gate> = <lista-delays> ;
<tipo-gate> ::= <tipo-gate-simples> | <tipo-gate-tristate> |
                <tipo-buffer> | cmos | nmos | pmos


*  declaração  de  intensidades  com  valor  atribuído  por
identificador *


<descricao-intensidades> ::= <id-intensidade>
                             <lista-decl-intensidades>
<lista-decl-intensidades> ::= <decl-intensidade> |
                      <decl-intensidade> <lista-decl-intensidades>
<decl-intensidade> ::=
     <nome-intensidade> = <intensidade> , <intensidade> ;
```

```
* declaração das interconexões entre componentes e destes com a
interface *

<lista-declar-conexoes> ::= <lista-conexoes-internas>
                            <lista-conexoes-interface>

<lista-conexoes-internas> ::= <conexoes-internas> |
                        <conexoes-internas> <lista-conexoes-internas>
<conexoes-internas> ::= node <lista-declar-nodos>
<lista-declar-nodos> ::= <declar-nodo> |
                         <declar-nodo> <lista-declar-nodos>
<declar-nodo> ::= <nome-nodo> = <lista-pinos> ; |
                  <lista-pinos> ;
<nome-nodo> ::= <identificador>
<lista-pinos> ::= <pino> | <pino> , <lista-pinos>
<pino> ::= <nome-elemento> . <identif-pino>
<nome-elemento> ::=
      <nome-gate> | <nome-transistor> | <nome-resistor>
<identif-pino> ::= <nome-pino> | <numero-pino>
<numero-pino> ::= <inteiro>


<lista-conexoes-interface> ::= <conexoes-interface> |
                <conexoes-interface> <lista-conexoes-interface>
<conexoes-interface> ::= equivalence <lista-equivalencias>
<lista-equivalencias> ::= <equivalencia> |
                          <equivalencia> <lista-equivalencias>
<equivalencia> ::= <bit-sinal> = <lista-pinos> ; |
                   <bit-sinal> = <nome-nodo> ;
<bit-sinal> ::= <identificador> |
                <identificador> [ <inteiro> ]


* definições básicas *

<inteiro> ::= <digito> | <digito> <inteiro>
<digito> ::= 0 | 1 | 2 | ... | 9

<identificador> ::= <letra> | <letra> <texto>
<texto> ::= <simbolo> | <simbolo> <texto>
<simbolo> ::= <letra> | <digito> | _
<letra> ::= a | b | c | ... | z |
            A | B | C | ... | Z
```

Relatórios de Pesquisa

RP-66 WAGNER, F. R. & DAL SASSO-FREITAS, C. M., "NILO - uma linguagem para a descrição de hardware no nível de portas lógicas" , março 87.

RP-65 WAGNER, F. R. , DAL SASSO-FREITAS, C. M. & GOLENDZINER, L. G. , "Linguagens de descrição de hardware para suporte à integração do processo de projeto em AMPLO" , março 87.

RP-64 ROCHA COSTA, A. C. , "Prolog como linguagem de implementação de sistemas: oito programas ilustrativos" , fev 87.

RP-63 POLANCZYK, C. A. , CLAUDIO, D. M. & LOPEZ, J. D. , "Software elementar para intervalos" , fev 87.

RP-62 WESTPHALL, C. B. , "Sinópse da especificação MAP", dez 86.

RP-61 ROCHA COSTA, A. C. , "Sobre os fundamentos da inteligência artificial" , nov 86.

RP-060 WALTER, C., "A method for the specification of manufacturing systems and their controllers" , out 86.

RP-059 PALAZZO OLIVEIRA, J.P. & RUIZ, D.D.A., "Formulários Eletrônicos: definição e manipulação automática da interface de usuário" , set 86.

RP-058 TOSCANI, L.V. & SZWARCFITER, J.L., "Algorítmos aproximativos" , set 86.

RP-057 JANSCH-PORTO, I. & COURTOIS, B., "Design and assembling of SDC checkers based on analytical fault hypotesis" , set 86.

RP-056 TAZZA, M., "Fundamentals of a Net Theory Based Performance Evaluation Model" , ago 86.

RP-055 FREITAS, C.M.D.S & OLIVEIRA, F.M, "Editor gráfico para sistemas digitais descritos no nivel lógico" , ago 86.

RP-054 ROCHA COSTA, A.C., "Para uma revisão epistemológica da inteligência artificial" , jul 86.

RP-053 TAZZA, M., "Quantitative analysis of a resource allocation problem: a net theory based proposal" , jul 86.

RP-052 LONGHI, M.T., "Representação de objetos sólidos por octrees" , julho 86.

RP-051 NASCIMENTO, F.R. do, Kit MCP-68000: Descrição do projeto de hardware., julho 86.

RP-050 MURR, A.L.D. & CASTILHO, J.M.V. de, Tradução de sentenças em lógica para sentenças em forma clausal: uma implementação em PROLOG.", julho 86.

RP-049: TOSCANI L.V. & VELOSO P.A.S., Especificação formal e análise da complexidade da programação dinâmica., mai 86.

RP-048: HURTADO, J.O.H.; GOMES, R.F. & SEELING, M., Documentação da concepção e testes de um circuito integrado NMOS. jun/86.

RP-047: PALAZZO OLIVEIRA, J., Electronic forms: a local area microcomputer system - Project description. (também disponivel em português.), mai/86.

RP-046: TOSCANI, L.V. Guia de estudo da complexidade de algorítmos de procura. nov/85.

RP-045: NASCIMENTO, F.R. & OLIVEIRA, R.S., Programa monitor para um microcomputador educacional implementado com o MC68000. dez/85.

RP-044: WAGNER, F.R., Simulação de sistemas modelados como redes de agencias. nov/85.

RP-043: FREITAS, C.M.D.S. & OLIVEIRA, F.M., Editor gráfico para sistemas modelados como redes de agencias. nov/85.

RP-042: TAZZA, M., Sistemas-C/E como ferramenta de modelagem. out/85.

RP-041: WAGNER, F. R.; FREITAS, C. M. D. S. & GOLENDZINER, L. G., O processo de projeto de sistemas digitais num ambiente integrado de CAD. out/85.

RP-040: TAZZA, M., Performance evaluation using a net theory based model. set/85.

RP-039: WAGNER, F. R., On the properties of event oriented logic simulation according to significant timing models. set/85.

RP-038: WAGNER, F. R., Algorítmos de simulação de hardware no nivel RT. set/85.

RP-037: COSTA, A. C. R., Processando linguagens naturais em PROLOG - Parte 1: Formalismo gramatical básico. jul/85.