UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ELETRÔNICA E MICROELETRÔNICA

QUANTICO - Simulador para Circuitos Analógicos que Operam Segundo a Técnica de Quantização

Dilvan de Abreu Moreira

ORIENTADOR: Dr. Carlos Alberto dos Reis Filho

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Janeiro - 1991

A Mariá, minha mãe; a meus irmãos e meu pai.

... sugiro-lhe firmar-se neste princípio básico: não dê crédito algum ao que ele diz, mas julgue aquilo que produziu! Porque o criador tem essa característica: as produções da sua imaginação se impõem a ele, tão indispensáveis, tão naturais, que não pode considerá-las como imagem do espírito, mas as conhece como realidade evidentes.

Albert Einstein - Como vejo o Mundo.

Agradecimentos:

A meus amigos baianos, em especial a Itamar, Ivan, César e Marcos, pelo apoio e amizade.

A Daniel Roseno (Programador dos bons), pelo seu apoio e incentivo.

Ao pessoal do laboratório, em especial a Marcinha e Neil.

A Fernanda, pelo trabalho de revisão.

Aos prof. Wilmar e Elnatan, pelas sugestões.

A Reis, pela orientação e incentivo.

À CAPES, pelo apoio financeiro.

Enfim, a todas as pessoas que contribuiram para a realização desse trabalho.

Resumo

O QUANTICO é um simulador para circuitos elétricos aplicados no processamento de sinais analógicos, segundo técnica de amostragem de sinais discretos no tempo e amplitude.

Na classe de circuitos em que o QUANTICO se aplica, são utilizados como primitivas os dispositivos: chave analógica, capacitor, somador de tensão de ganho unitário, comparador de tensão, buffer de ganho unitário, divisor de tensão por dois, subtrator de tensão e fonte de tensão. Basicamente, o simulador efetua a operação de atualização de tensões nos capacitores conectados ao circuito, que funcionam como elementos de memória analógica. Com estes dispositivos são implementados circuitos que desempenham funções básicas de condicionamento de sinais analógicos: amplificador de tensão, multiplicador de duas tensões, conversor digital serial / analógico, etc.

Apesar da semelhança com os simuladores para redes de capacitor chaveado, o QUANTICO foi feito para uma classe particular de circuitos. Ele é executável em micros compatíveis com o IBM-PC rodando MS-DOS versão 2.1 (ou posterior). Ele foi escrito na linguagem C para ter fácil portabilidade.

Abstract

QUANTICO is a simulator for electrical circuits applied to the analog signal processing using a sampling technique of signals discret in time and amplitude.

In the class of circuits where QUANTICO is applied, the following devices are used as primitives: analog switch, capacitor, unit-gain voltage adder, voltage comparator, unit-gain buffer, voltage devider by two, voltage subtractor, and voltage source. Basically, the simulator does the operation of voltage updating in the capacitors connected to the circuit, which work like analog memory elements. With these devices circuits are implemented which perform basic functions of analog signals conditioning, such as: voltage amplifier, multiplier of two voltages, digital serial to analog converter, etc.

Despite the similarity with the simulators for switched capacitor networks, QUANTICO was made for a particular class of circuits. It runs in IBM-PC compatible microcomputers running under the MS-DOS version 2.1 (or later). It was written in C language for easy portability.

Índice

Introdução

O homem domina a natureza não pela força, mas pela compreensão. Esse é o motivo pelo qual a ciência evolui onde a mágica falhou, pois não procurou um feitiço para lançar sobre a natureza. Jacob Bronowski.

Este trabalho tem como objetivo o desenvolvimento de um simulador para circuitos elétricos do tipo aplicado no processamento de sinais analógicos segundo técnica de amostragem de sinais discretos no tempo e na amplitude.

A técnica de processamento de sinais para o qual o simulador foi desenvolvido visa o processamento de sinais analógicos na mesma faixa de freqüência e amplitude onde se aplica a técnica de capacitores chaveados. A comparação dessas duas técnicas está fora do escopo desse trabalho; entretanto, é necessário informar que a despeito da semelhança operacional entre estas técnicas de processamento de sinais, os simuladores desenvolvidos para redes de capacitor chaveado (SCN's - Switched Capacitor Network)[1] [2] [3]; tais como SWITCAP

ou CAPZ, não se aplicam à técnica de processamento para o qual o presente simulador foi desenvolvido.

O QUANTICO trabalha basicamente com circuitos compostos por chaves analógicas e capacitores, como as SCN's. Ele calcula os valores de tensão do circuito, em cada ciclo da simulação, percorrendo os segmentos do circuito que estão conectados entre si (*nets*). Se existe alguma fonte de tensão numa *net*, todos os nós que pertencem a ela passam a ter o valor de tensão da fonte. Se só existem capacitores, o valor de tensão da net passa a ser a média ponderada das tensões dos capacitores (as cargas se redistribuem entre os capacitores). Com os valores de tensão dos nós calculados, o QUANTICO efetua a operação de atualização das cargas, e consequentemente das tensões, nos capacitores do circuito.

Depois disso o simulador testa se algum comparador mudou de estado, abrindo ou fechando chaves analógicas que, então, alteram a configuração das *nets* do circuito. Quando isso ocorre, os novos valores de tensão são calculados. O QUANTICO então guarda os valores calculados e conta mais um ciclo de simulação, operando as chaves associadas aos sinais de clock que se modificaram. O QUANTICO prossegue desse forma até que terminem todos os ciclos de clock da simulação.

O QUANTICO trabalha em computadores compatíveis com o IBM-PC que tenham como configuração mínima 512Kb de memória, com sistema operacional MS-DOS versão 2.1 (ou posterior) e uma unidade de disco flexível. Ele foi escrito em C (Turbo C - Borland) para maior portabilidade, podendo também ser transportado para outros ambientes (Mainframes ou Workstations).

1

A METODOLOGIA USADA

A ciência é construída com fatos, como uma casa o é com pedras. Mas um conjunto de fatos é uma ciência tanto quanto uma pilha de pedras é uma casa. Jules Henri Poincaré.

O QUANTICO é dividido em 3 módulos; essa divisão foi feita visando maior aproveitamento do espaço de memória RAM, porque assim as três partes do programa nunca estarão na memória ao mesmo tempo, e maior portabilidade:

O primeiro módulo (GERA) lê o arquivo de descrição do circuito, testa erros de sintaxe e o uso de configurações de circuito não permitidas como, por exemplo, curto-circuitos de chaves, 2 fontes ligadas ao mesmo nó, etc.

O segundo módulo (SIMULA) é o simulador propriamente dito. Ele lê a descrição do circuito gerada pelo GERA, simula o circuito e gera um arquivo com os dados produzidos pela simulação.

O terceiro módulo (OSC) é um pós processador gráfico, inspirado na utilização de um osciloscópio. Ele lê os dados gerados pela simulação (SIMULA) e os apresenta na tela, como se fosse um osciloscópio multicanais, contando com recursos como: expansão de período, ganho independente para cada canal, etc, comumente encontrados em osciloscópios.

Quanto à portabilidade, ela foi facilitada porque o único módulo que tem rotinas gráficas (OSC) fica separado dos outros. As rotinas gráficas são dependentes do hardware usado prejudicando a portabilidade do OSC.

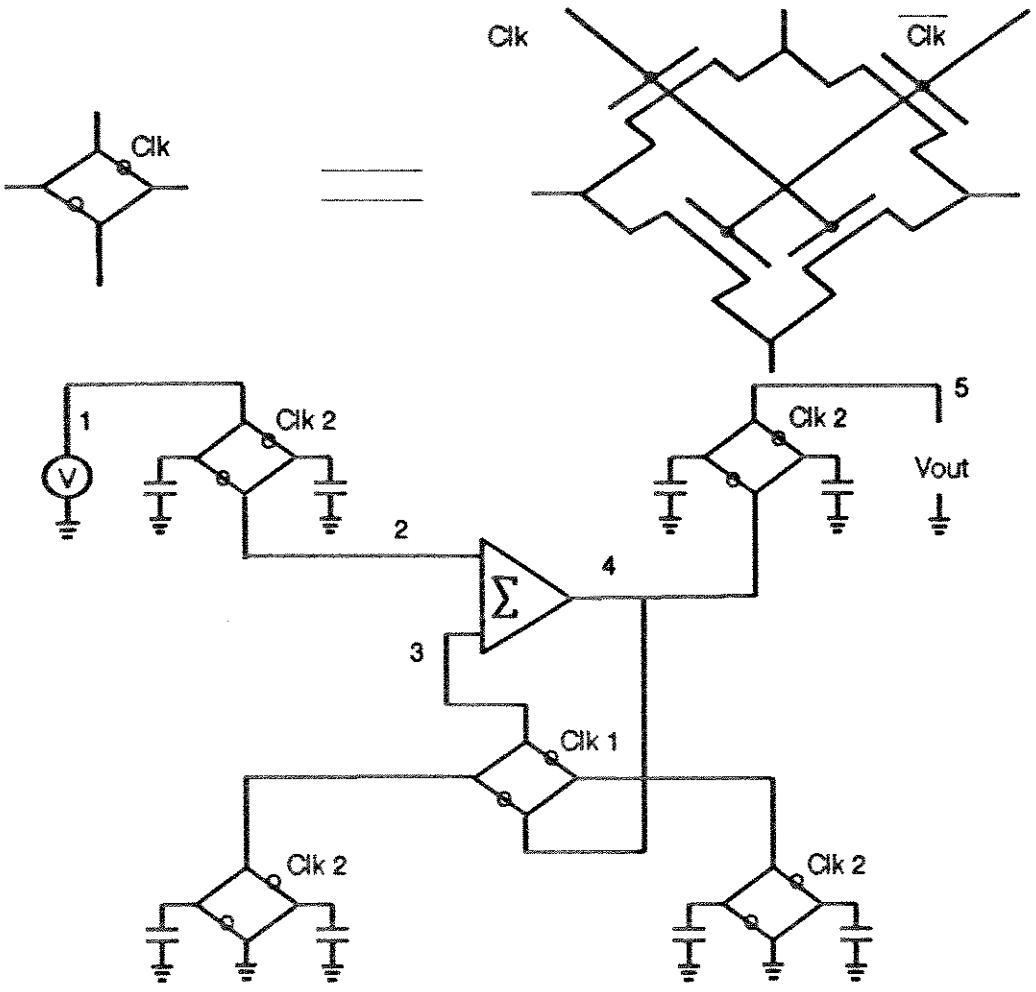
1.1 Descrição da técnica de quantização

O QUANTICO é uma ferramenta de suporte no desenvolvimento de circuitos que operam segundo a técnica que chamamos de *quantização*.

Utilizando componentes como capacitores, chaves analógicas, somadores de ganho unitário, comparadores de tensão, etc, a técnica de quantização permite a realização de amplificadores de tensão, multiplicadores de tensão (2 e 4 quadrantes), conversores Digital serial - Analógico, integradores, diferenciadores, etc.

Fundamentalmente, obtem-se através desta técnica o efeito equivalente à amplificação de tensão, através de ciclos de acumulação no tempo, da amplitude do sinal a ser amplificado. O número de vezes que o sinal inicial é somado a si mesmo define o fator de amplificação.

Ao se acumular um sinal de tensão de referência, até que o resultado da acumulação atinja o nível de um sinal de tensão V_1, está-se quantizando V_1. Concomitantemente, acumulando-se a amplitude de um outro sinal V_2 o mesmo número de vezes que se levou para quantizar V_1, tem-se o produto algébrico $V_1 . V_2$, tendo como elemento unitário a tensão de referência que quantiza V_1.



Cada lado dos losangos representa uma chave, os lados marcados com um círculo operam com sinal baixo. Ao lado de cada losango tem-se o clock que o comanda. A amplificação de tensão é dada por f1 / f2, onde f1 e f2 são as frequências dos clocks CLK1 e CLK2.

Figura 1.1: *Amplificador de tensão*

Através desta técnica, operações matemáticas podem ser implementadas para realizar funções específicas.

A técnica de quantização concorre, no universo de circuitos em que se aplica, principalmente com a técnica de capacitor chaveado (SCN's). Há dois aspectos na técnica de quantização

que a tornam viável, considerando que são vantagens numa comparação com SCN's: primeiro, na técnica de quantização todos os capacitores têm um terminal aterrado: isto implica na utilização de uma tecnologia mais simples, portanto mais barata; segundo, o seu modo de operar é compatível com circuitos digitais, viabilizando a integração de sistemas completos.

1.2 Descrição da técnica de processamento

Os dispositivos utilizados na realização dos circuitos aos quais o QUANTICO se destina são: capacitores ligados ao terra, que têm a função de guardar valores de tensão; chaves analógicas, que mudam as ligações entre os diversos elementos do circuito; geradores de pulsos de clock, para o *timing* das chaves; comparadores de tensão, que podem controlar chaves ; e blocos de apoio, como buffers, somadores de tensão, etc.

Apesar de haver grande semelhança com circuitos usando capacitores chaveados (SCN's), os circuitos simulados pelo QUANTICO permitem o acionamento de chaves fora do *timing* dos sinais de clock. Isso é feito através de comparadores que controlam chaves da rede, o que inviabiliza o uso dos simuladores para SCN's.

A simulação de circuitos não se baseia, como ocorre em SCN's, numa redistribuição de cargas entre os capacitores, mas sim na troca de valores entre eles. O QUANTICO assume que, entre os pulsos do clock, há tempo suficiente para a completa carga e descarga dos capacitores do circuito.

Partindo dessa premissa, torna-se desnecessário considerar as resistências das chaves do circuito (Nas SCN's essas resistências não são consideradas), reduzindo o esforço computacional.

Assim, foi adotado que todas as fontes de tensão dos circuitos são modeladas com impedância de saída zero. As chaves, transistores n-MOS, têm resistência de canal nula, quando fechadas, e infinita, quando abertas. Além dessas premissas, foi adotado que os capacitores não têm correntes de fuga e

os elementos ligados ao circuito têm impedância de entrada infinita.

O cálculo das tensões dos nós fica grandemente simplificado. Basta percorrer segmentos do circuito, que estão conectados entre si (redes), e verificar se existe alguma fonte de tensão. Se existir alguma, o valor de tensão de todos os nós da rede passa a ser o valor da fonte de tensão; se não existir, o valor da tensão da rede passa a ser a média ponderada dos valores de tensão dos capacitores ligados à rede (as cargas se redistribuem entre os capacitores).

Se numa rede não existirem capacitores ou fontes, ela será considerada aberta e a sua tensão será indeterminada; nesse caso, inicialmente, foi adotado o valor zero para a tensão, mas o modelo completo de simulação (explicado mais adiante) permite um cálculo mais correto. Na Fig. 1.1 tem-se um exemplo de circuito e na Fig. 1.3 um exemplo de uma rede.

Uma outra vantagem dessa forma de cálculo é que ela não necessita trabalhar com a resolução de matrizes de equações diferenciais, o que evita os problemas de convergência. Isso evita que os erros de arredondamento, especialmente entre elementos de faixa de valores diferentes, influam de maneira decisiva nos valores finais calculados. Por conseguinte não foi necessário adotar nenhuma estratégia de normalização para os valores dentro do simulador.

Com isso tem-se o modelo básico para os circuitos com os quais o QUANTICO trabalha. Com esse modelo foram montados os protótipos do simulador e testado seu princípio de funcionamento.

1.3 O Efeito de Injeção de Cargas

O fenômeno de injeção de cargas, que ocorre na transição de abertura de uma chave analógica implementada com um transistor MOS, tem sido uma das mais significativas fontes de erro em circuitos do tipo *Sample & Hold*.

Uma revisão da literatura revela que este problema tem sido estudado [4] [5] [6] [7] [8] com enfoque, principalmente, no estabelecimento de um modelo elétrico que melhor represente o fenômeno. Há evidências, porém, de que o assunto ainda deva ser mais explorado, com vistas à proposição de uma técnica de compensação desse efeito, para minimizar os erros causados por ele.

Na implementação do QUANTICO, consideraram-se os modelos de chave analógica propostos por D. MacQuigg [4] e B. J. Sheu e C. Hu [5]. Embora existam algumas diferenças nas duas abordagens, elas tendem à mesma formulação quando se deseja saber a quantidade de carga injetada no capacitor pela comutação do transistor MOS, comandado por um pulso de transição muito rápida, depois de um tempo relativamente longo.

As duas abordagens se baseiam em um mesmo modelo de chave analógica e analisam o circuito mostrado na Fig. 1.2b. Neste modelo se incluem capacitores do gate ao dreno e do gate ao source dos transistores, que representam as capacitâncias de overlaping e de óxido. A capacitância de overlaping aparece nas áreas do gate que estão superpostas ao dreno e ao source. A capacitância de óxido aparece entre

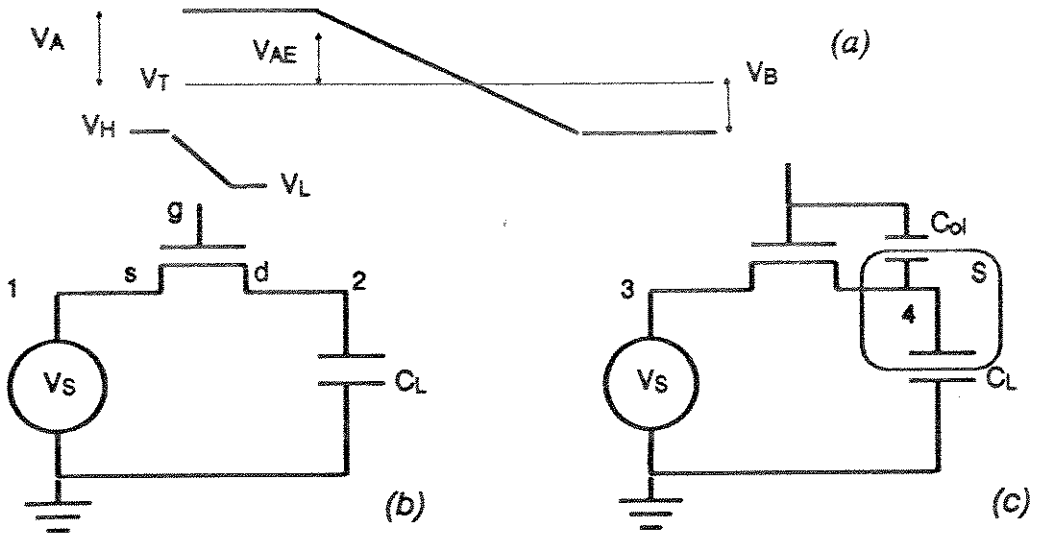


Figura 1.2: *Circuitos básicos dos modelos*

o gate e o canal, quando este está formado. Quando os transistores estão ligados, essas duas capacitâncias se somam; quando eles estão desligados, existe apenas a de overlaping.

Na primeira abordagem [4], a carga residual no capacitor é calculada para o caso em que a fonte de sinal é nula, ou seja:

$$Q(\infty) = Q_{max} = (C_{GD}+C_{ov})V_{AE} + C_{ov}V_B \qquad (1.1)$$

Sendo Q_{max} a carga residual máxima, C_{GD} a capacitância intrínseca entre gate e dreno e C_{ov} a capacitância de overlaping (Fig. 1.2a).

$V_{AE} = V_A = V_H - V_T$, quando o tempo de transição do pulso do gate é muito rápido. (1.2)

$$V_B = V_T - V_L \qquad (1.3)$$

Sendo V_H a tensão máxima, V_L a tensão mínima de clock e V_T a tensão de threshold. A carga injetada é de elétrons.

Na segunda abordagem [5], é descrita a tensão adicional devida à carga residual (v_{dn}) no capacitor C_L como:

$$-v_{dn} = v_{dm} = V_{HT}\frac{(C_{ol}+C_{ox}/2)}{C_L} + (V_S+V_T-V_L)\frac{C_{ol}}{C_L} \qquad (1.4)$$

$$V_{HT} = V_H - V_S - V_T \qquad (1.5)$$

Sendo $C_{ox}/2=C_{GD}$, $C_{ol}=C_{ov}$, e V_S o sinal de excitação.

Da equação 1.4 e 1.5 tem-se:

$$-v_{dn} = (V_H-V_L)\frac{C_{ol}}{C_L} + (V_H-V_S-V_T)\frac{C_{ox}/2}{C_L} \qquad (1.6)$$

Que é igual ao que seria obtido usando (1.1) (1.2) (1.3) considerando $V_S = 0$.

1.3.1 Método usado no QUANTICO

No QUANTICO foi usado um método parecido com o descrito anteriormente. Usando o princípio de conservação de carga numa superfície fechada, apresentado em [1] e [2], calcula-se a carga que fica retida na superfície S associada ao nó 2, onde está ligado o capacitor C_L (Fig. 1.2b):

$$Q = V_S C_L + (V_S - V_H)C_{ol} + (V_S + V_T - V_H)C_{ox}/2 \qquad (1.7)$$

Usando o mesmo princípio, calcula-se a tensão em C_L, depois que o transistor abre (Fig. 1.2c):

$$Q = V_{CL}C_L + (V_{CL} - V_L)C_{ol} \qquad (1.8)$$

$$V_{CL} = \frac{Q + V_L C_{ol}}{C_L + C_{ol}} \qquad (1.9)$$

Como a carga se conserva:

$$V_{CL} = V_S - \frac{(V_H - V_L)C_{ol} - (V_H - V_S - V_T)C_{ox}/2}{C_L + C_{ol}} \qquad (1.10)$$

$$-v_{dn} = v_{dm} = \frac{(V_H - V_L)C_{ol} - (V_H - V_S - V_T)C_{ox}/2}{C_L + C_{ol}} \qquad (1.11)$$

A equação 1.11 se diferencia da equação 1.6 pelo termo C_{ol} do denominador. No método adotado no QUANTICO considerou-se que a carga injetada se distribui entre C_L e C_{ol}. A diferença entre as duas metodologias não deve ser muito grande, já que $C_L >> C_{ol}$. Contudo, considerou-se a equação 1.11 mais correta que a 1.6.

1.3.2 Cálculo usado no algoritmo

O cálculo das tensões nas redes que contêm fontes de tensão não se modifica. Como todos os transistores passam a ter capacitores ligados, não existem mais nós abertos. Isto foi

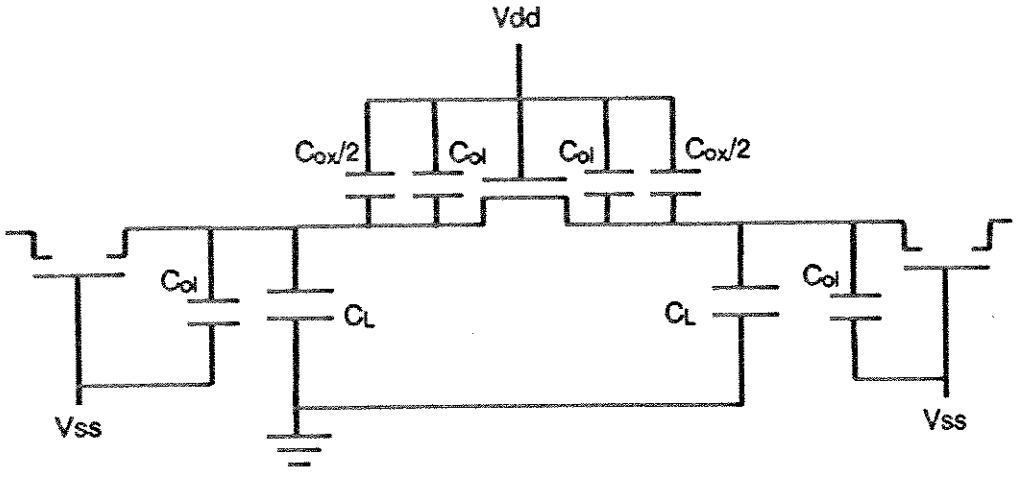


Figura 1.3: *Circuito de uma rede no QUANTICO*

discutido no item 1.2. A tensão das redes que não têm fontes, como a da Fig. 1.3, passa a ser calculada por:

$$V = \frac{Q + \sum C_1 V_{dd} + \sum C_2 V_{ss}}{\sum C_L + \sum C_1 + \sum C_2} \qquad (1.12)$$

Onde C_1 representa os capacitores ligados a Vdd (Parasitas dos transistores ligados), C_2 representa os capacitores ligados a Vss (Parasitas dos transistores desligados) e C_L representa os capacitores ligados aos nós da rede. A carga Q é a soma das contribuições de cada capacitor (incluindo os parasitas), que foram calculadas e armazenadas no ciclo anterior.

Esse método trata os capacitores igualmente (C_{ol} e C_{ox}). A carga armazenada em cada capacitor é dada pelo valor da tensão sobre ele vezes o valor da capacitância. Os C_{ox} contudo não têm essa carga, apesar de se comportarem como se tivessem, enquanto existem no circuito. Assim o programa retira uma parcela da carga armazenada nesses capacitores quando o transistor abre (e eles deixam de existir); esse resíduo de carga é de $V_T C_{ox}$, que é o valor subtraído na equação 1.11 da carga do capacitor.

Isso modela o fenômeno da abertura da chave analógica (transistor MOS), contudo o fechamento não é abordado em

nenhum dos trabalhos (que se preocupavam com o fenômeno para o caso de circuitos do tipo Sample & Hold). Decidiu-se, então, por uma abordagem simétrica, ou seja, o que acontece na abertura deve acontecer ao inverso no fechamento. Assim o resíduo de carga é devolvido à carga dos C_{ox}.

Devido à simplicidade do modelo usado no QUANTICO, foi possível incluir esse efeito na versão completa, sem um grande prejuízo da perfomance. O maior custo é o cálculo das cargas dos capacitores parasitas dos transistores, que não existiam no modelo simplificado e que deve ser feito em cada ciclo do clock de simulação.

1.4 Algoritmo do simulador

O tipo de representação dos dados determinou em grande parte o algoritmo utilizado. Ele foi escolhido entre duas técnicas:

Equacionamento do circuito, utilizando matrizes que representam os estados do circuito durante a simulação. Nesta técnica o algoritmo converte o circuito para uma representação matemática matricial e então tenta resolver o problema matemático em si. Ao final da simulação, os dados representam o comportamento do circuito (Seguindo o modelo matemático simulado).

Essa representação funciona bem, quando se dispõe de uma representação (modelo) matematicamente bem determinada do circuito e este não é excessivamente complexo. Ele é usado em simuladores tipo SPICE [9], que resolve matrizes de equações diferenciais. As maiores desvantagens desse tipo de abordagem são:

- As quantidades de memória e tempo de CPU necessárias para a simulação de grandes circuitos, pois o uso desses dois recursos não crescem de maneira linear com o tamanho do circuito.
- A impossibilidade de se manter uma descrição do circuito durante a simulação para testar condições proibidas (Tensão de ruptura do capacitor, tempos de escrita em memórias, etc).

O segundo tipo de representação considerado é o que usa estruturas de dados para manter uma descrição do circuito e para armazenar os estados durante a simulação. Esse é o caso típico de diversos simuladores lógicos [11] [12]. A maior vantagem dessa estrutura é que ela trabalha diretamente com uma representação do próprio circuito, guardando informações sobre a topologia do circuito, bem como sobre seus elementos individuais. Com isso pode-se reter informações úteis para o controle da simulação, diminuindo o perigo de gerar conclusões absurdas. Outra vantagem é a possibilidade de testar situações potencialmente perigosas para o projeto, como, por exemplo, uma violação de *set-up time* de um registrador dinâmico ou da tensão de isolação de um capacitor.

Esse tipo de representação usa uma descrição do circuito e não uma matriz associada; o espaço de memória cresce linearmente com o tamanho do circuito. Ao se usarem modelos mais simplificados para os componentes do circuito e para as relações entre eles, obtém-se uma redução do tempo de simulação.

O segundo tipo de representação foi escolhido para o QUANTICO, por se achar que os dispositivos envolvidos eram suficientemente simples, e por serem usados modelos resumidos para os elementos do circuito e para as relações entre eles.

1.5 Protótipo do simulador

O modelo predominante para o desenvolvimento de programas aplicativos atualmente é a abordagem do refinamento em fases. Nessa abordagem, toda a funcionalidade do sistema é especificada no primeiro passo do desenvolvimento. Fases de implementação subsequentes adicionam detalhes predefinidos ao projeto.

Um novo método é a criação de protótipos, que é o processo de desenvolver uma versão, em menor escala, de um sistema para ser usado na construção de um sistema em escala maior [13].

No projeto do QUANTICO essa técnica foi utilizada para dar maior rapidez ao desenvolvimento do projeto. Os dois primeiros módulos, o GERA e o SIMULA, tiveram um protótipo único em Turbo Prolog, que é uma boa linguagem para protótipos. Esse protótipo usava o modelo simplificado, onde não era simulado o efeito da injeção de cargas pelos transistores. O protótipo serviu para validar a técnica de simulação e o algoritmo usados. O módulo OSC teve um protótipo em Turbo Pascal, devido às suas exigências gráficas.

2

O PROGRAMA LEITOR

God made the integers; all else is the work of man. Leopold Kronecker.

O reconhecimento de uma linguagem de comandos complicada é normalmente dividido em duas fases: análise léxica e reconhecimento. Objetos de nível baixo, como números, operadores e palavras especiais são reconhecidos durante a fase de análise léxica, ao passo que objetos de alto nível, como instruções em uma linguagem de programação, são reconhecidos durante a fase de reconhecimento [14].

O programa GERA tem, além da função de reconhecer a linguagem de comandos do simulador, a função de gerar, a partir desse reconhecimento, um arquivo descritivo que possa ser usado pelo simulador (SIMULA).

Ele é formado por 3 partes : Um analisador léxico e um reconhecedor, que implementam o reconhecimento da linguagem e um compilador, que gera o arquivo de saída. Sua estrutura de dados é basicamente constituída de listas apontadas.

2.1 O Analisador Léxico

A entrada para a fase de análise léxica é um fluxo de caracteres vindo do arquivo de entrada. Isso é feito através de uma solicitação do reconhecedor. O programa de análise léxica verifica a entrada de acordo com um conjunto de regras. Quando ele reconhece um objeto (tipo léxico) , ele o envia ao reconhecedor com uma indicação do tipo. O propósito da fase da análise léxica é melhorar muitas irregularidades da entrada, como o espaçamento e o comprimento dos itens, para que se produza uma saída que reconheça e separe os itens encontrados (independentemente da sua arrumação).

Existem cinco tipos léxicos que o analisador do GERA identifica:

- **Comentários** são todas as letras minúsculas e linhas iniciadas por *. Eles são identificados e ignorados pelo analisador, não sendo assim passados para o reconhecedor.
- **Números inteiros** são cadeias de números, além do símbolo "-".
- **Números reais** são cadeias de números e podem conter também os símbolos "+", "-", "E".
- **Palavras** são cadeias de caracteres alfa-numéricos.
- **Símbolos** são caracteres especiais como EOF (Fim de arquivo), ENTER, que é o marcador de fim de linha, "(", ")", etc.

Seria difícil separar os números inteiros dos reais (visto que um número inteiro é real). Por isso todos os números são lidos na categoria de palavras e depois são testados por uma outra rotina da parte léxica, a depender se naquela posição era esperado um número inteiro ou real.

2.2 O Reconhecedor

A fase de análise do reconhecedor é responsável pela compreensão das propriedades de alto nível da entrada. O

reconhecedor é responsável pela verificação de que uma seqüência representa uma expressão válida e pela criação de um conjunto apropriado de ações.

O reconhecedor do GERA reconhece as estruturas da linguagem de descrição usada e cria uma estrutura de registros apontados (listas) para descrevê-las. Ao fazer isso, o reconhecedor tem que interpretar os comandos postos pelo usuário no arquivo de entrada. Durante esta ação, ele detecta os erros de sintaxe cometidos. É muito importante que, ao detectar esses erros, o reconhecedor forneça mensagens de erro precisas, para que o usuário possa corrigi-los devidamente.

Existem rotinas reconhecedoras bastante sofisticadas, que podem reconhecer diversos erros ao mesmo tempo, ou reconhecer os erros e indicar possíveis correções. Na implementação do GERA foi adotada uma estratégia mais simples, já que a simulação é o objetivo principal. O GERA só pode reconhecer um erro por vez e, assim que um erro é encontrado, o programa mostra a mensagem de erro correspondente e pára a listagem no local onde este ocorreu.

Como se trata apenas de uma linguagem de descrição, e não de uma linguagem de programação, considerou-se que isso era suficiente.

2.2.1 Algoritmo

O algoritmo usado para análise de reconhecimento é clássico [15] [14] e baseia-se em seguir o conjunto de regras que regem a sintaxe da linguagem de comandos. Se um comando se adapta às regras, então alguma ação apropriada é tomada; se o comando não se ajusta às regras, tem-se um erro. O conjunto dessas regras constitui o grafo sintático da linguagem. O grafo que a linguagem do GERA segue é definido no Anexo 2.

2.2.2 Estrutura de Dados

Depois de efetuar a leitura e verificar a sintaxe, o reconhecedor gera 3 estruturas de lista apontada: A primeira, para guardar

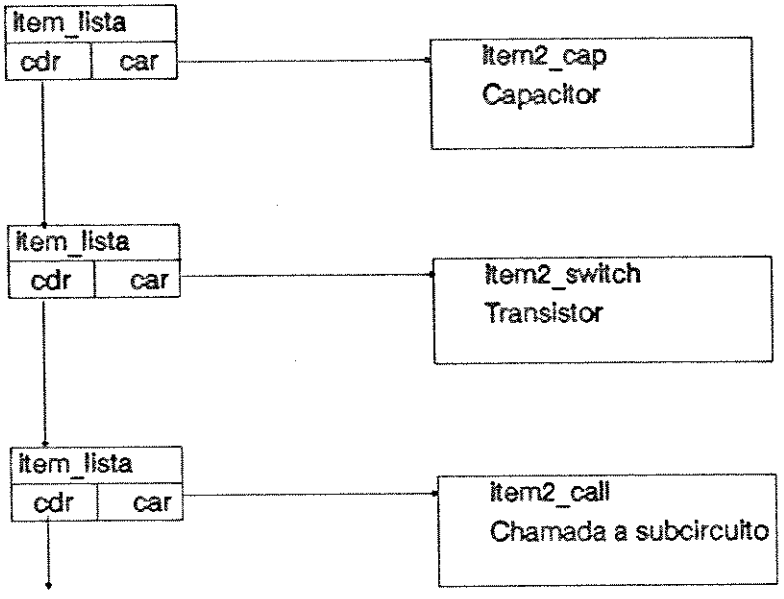


Figura 2.1: *Estrutura da descrição do circuito*

a descrição geral sobre o circuito, a segunda, para guardar as informações sobre cada sub-circuito chamado na descrição geral e a terceira, para guardar informações sobre os modelos usados.

A lista com a descrição geral do programa (Fig. 2.1) é agrupada através do registro (porção de memória) item_lista, que aponta (tem o endereço) para um outro tipo de registro (campo car), que contém informações sobre o que foi lido. O registro item_lista guarda também o tipo de dado para o qual ele aponta e o endereço do próximo item_lista (campo cdr).

Os tipos de registros apontados por item_lista e a informação que cada um guarda, são mostrados na tabela a seguir:

item2_cap	Capacitores
item2_switch	Transistores
item2_clock	Clocks
item2_devices	Buffers, Somadores, Subtratores e Divisores por 2

item2_cmp	Comparadores
item2_fonte_dc	Fontes DC
item2_senoide	Fontes de tensão senoidal
item2_sffm	Fonte de tensão senoidal com modulação FM
item2_pulso	Fonte de tensão pulsada
item2_palavra	Fonte de tensão pulsada, com definição da duração de cada pulso
item2_call	Chamada a um sub-circuito

Os subcircuitos chamados na descrição do circuito são agrupados em listas (Fig. 2.2).

As listas, que são apontadas pelos registros do tipo sub-circuito, têm a mesma estrutura da descrição geral do circuito, podendo, inclusive, ter novas chamadas a outros sub-circuitos.

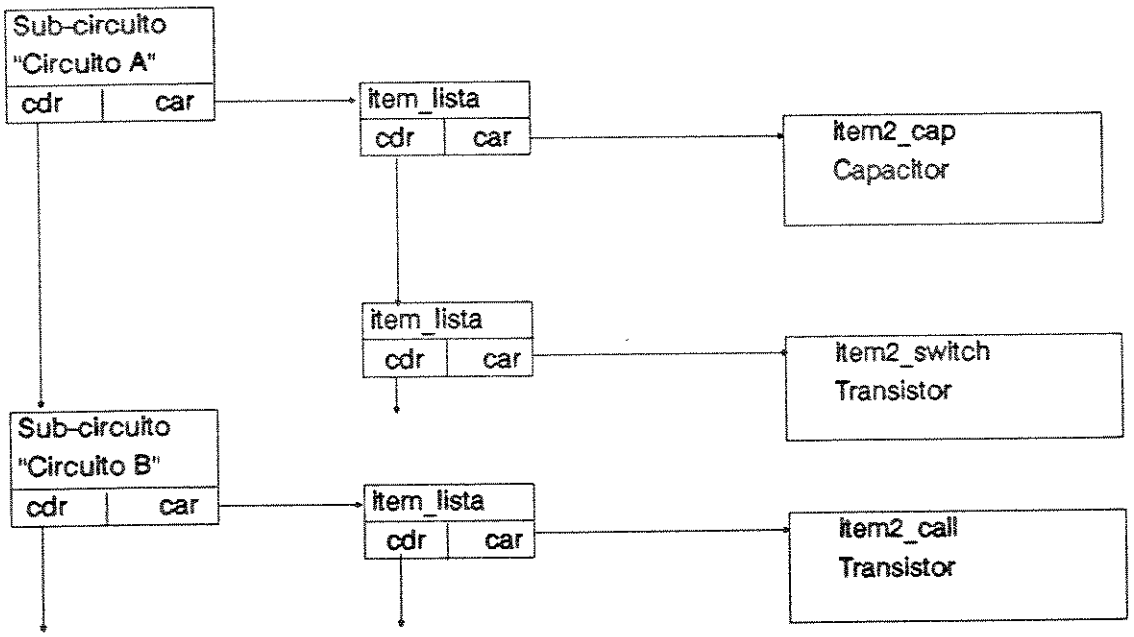


Figura 2.2: *Estrutura de dados dos sub-circuitos*

Os modelos dos componentes são separados por tipo, existindo uma lista para cada tipo. São listas de registros apontados sem a necessidade do item_lista.

Existem também algumas estruturas auxiliares que são lidas pelo reconhecedor e passadas ao simulador pelo registro Trans. Elas têm informações a respeito de parâmetros opcionais de simulação e informações sobre o timing (instrução .TRAN).

2.3 A Compilação

Uma vez que o reconhecedor tenha gerado as 3 listas, o compilador gerará a estrutura final dos dados (Fig. 2.3). Nessa fase ocorre a expansão dos subcircuitos em extensões do circuito principal, a troca dos nomes dos nós por números e a verificação da ocorrência de ligações proibidas (ex: uma fonte DC num gate de um transistor).

O algoritmo gera uma lista de dados definitiva, semelhante à lista de descrição geral do circuito, sem a chamada dos subcircuitos. Para gerar essa lista, o algoritmo varre a lista da

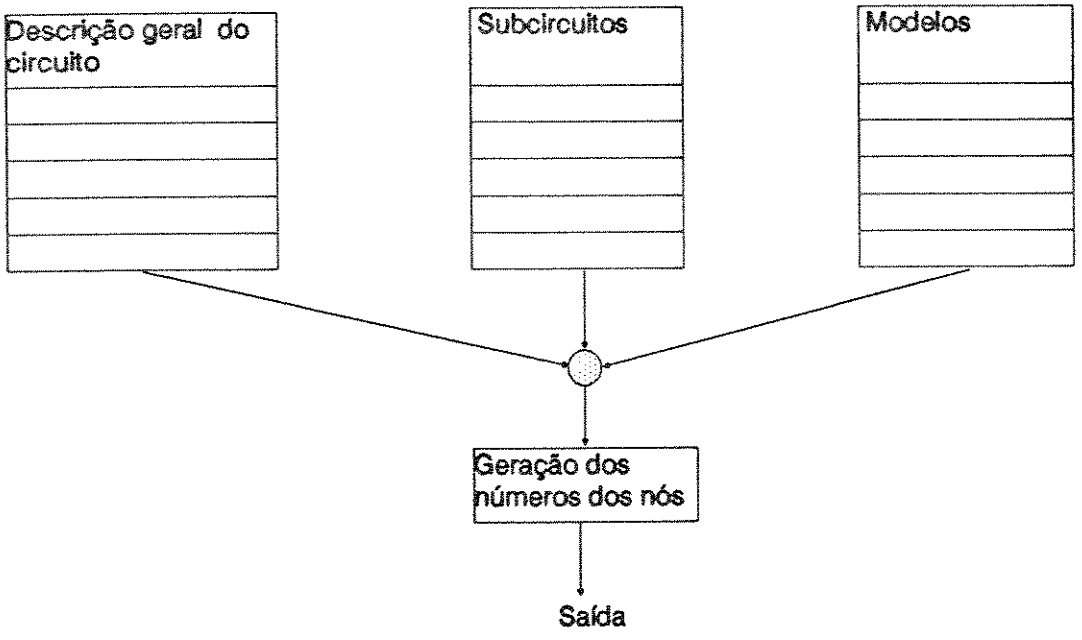


Figura 2.3: *Algoritmo do compilador*

descrição geral do circuito e, para cada item, é criado um novo item com o número definitivo do nó onde este estará ligado (esse número é gerado de forma sequencial) e com os parâmetros indicados pelo seu modelo (se existir um).

Ao encontrar uma chamada de subcircuito, o algoritmo procura a sua descrição na lista dos subcircuito através do seu nome. Ao encontrá-lo, copia o seu conteúdo na lista definitiva, da mesma forma como foi feito na descrição geral do circuito, dando novos números para os nós internos do sub-circuito a cada vez que ele é chamado. Se dentro da definição de um sub-circuito um outro sub-circuito for chamado, a mesma operação é repetida de maneira recursiva.

Depois de gerada, essa lista é gravada num arquivo em disco de extensão .TMP, separando-se cada tipo de item em listas (não apontadas). No início do arquivo existem informações, como o número de cada tipo de item e as estruturas auxiliares já mencionadas. Esse arquivo será lido pelo programa SIMULA.

3

O SIMULADOR

A natureza não é embaraçada pelas dificuldades da análise. Augustin Fresnel.

O simulador (SIMULA) é o módulo de programa que desempenha a função central, simular o circuito lido pelo GERA. Seu algoritmo usa estruturas de dados para manter uma descrição do circuito e para armazenar os estados durante a simulação de maneira análoga à usada em simuladores lógicos.

O SIMULA trabalha basicamente de maneira à executar uma ação apenas quando essa for necessária. Desta forma, qualquer valor é calculado apenas quando seu uso é necessário e só é descartando quando se torna inválido. Isso visa utilizar o tempo de CPU apenas quando houver mudanças nos estados dentro do circuito. A desvantagem dessa abordagem são as estruturas auxiliares de dados que, geralmente, são necessárias para garantir a consistência dos dados. Numa abordagem tradicional, poder-se-ia calcular todos os estados do circuito em cada ciclo do relógio; assim, num ciclo, todos os valores sempre seriam válidos. Concluiu-se, contudo, que as estruturas

auxiliares requeridas não carregariam demasiadamente o programa.

3.1 A estrutura de dados do SIMULA

Ao contrário do programa GERA, o SIMULA utiliza vetores, e não listas, como sua estrutura básica de dados. Esses vetores são definidos e alocados (alocação dinâmica) durante a leitura

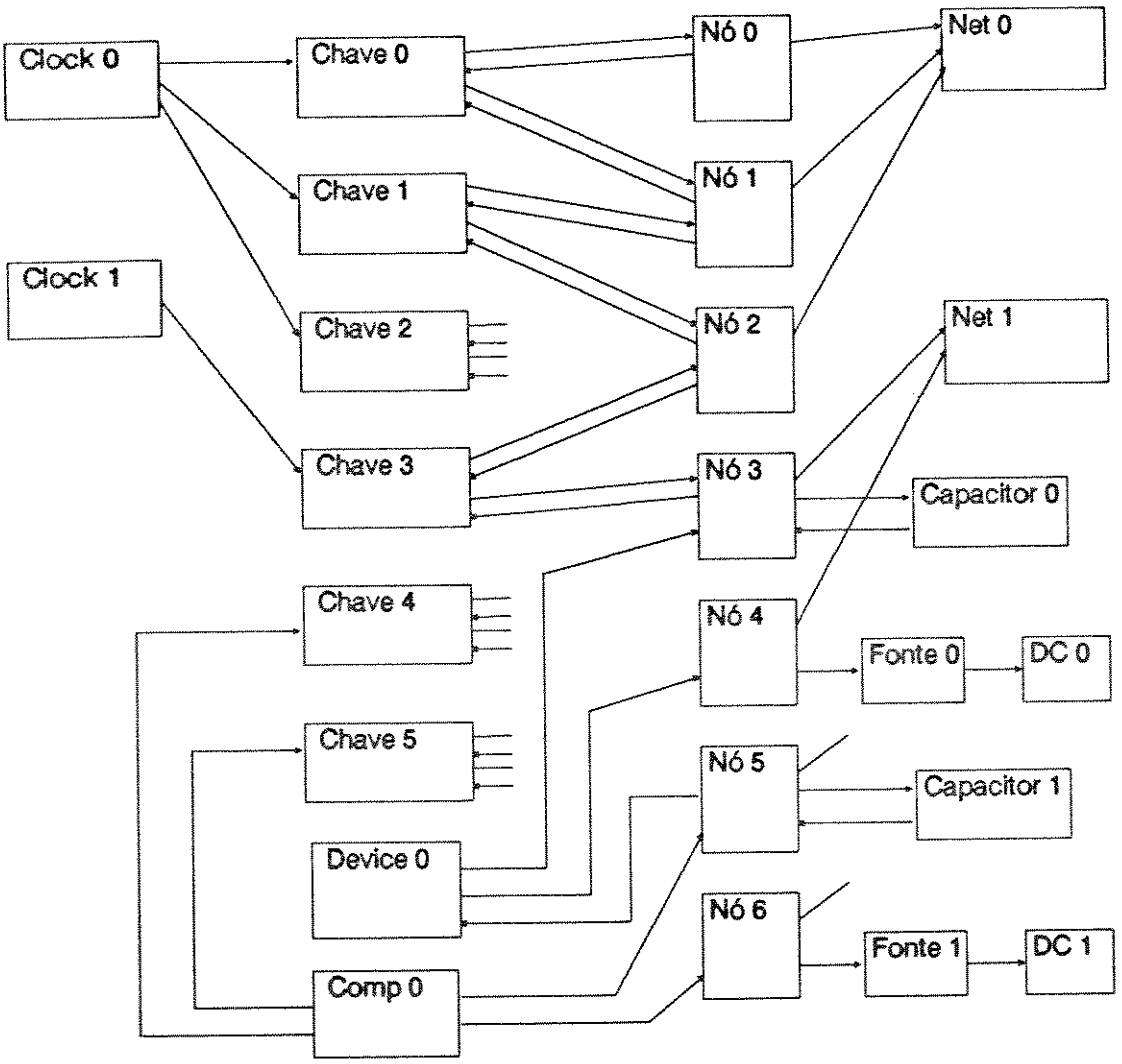


Figura 3.1: *Estrutura do banco de dados*

dos dados gerados pelo programa GERA. Existe um vetor para cada tipo de registro de dados necessário.

A estrutura de dados forma um banco de dados relacional sobre o circuito. As relações entre os diversos elementos do circuito são representadas por ponteiros, que ligam os elementos segundo a topologia do circuito (Fig. 3.1). Esses ponteiros não utilizam os endereços reais dos registros para os quais apontam, e sim a ordem desses registros dentro dos vetores aos quais eles pertencem. Assim o nó 5 (Registro 5 do vetor de nós) pode apontar para o capacitor 1 (Registro 1 do vetor de capacitores) mas, se o programa necessita do endereço real, terá de somar 4 * (tamanho de um registro de capacitor) ao endereço base do vetor de capacitores.

Em casos onde um registro apontar para um número variável de outros registros (Clocks ou comparadores apontando diversas chaves), é novamente usada a estrutura de listas apontadas. No SIMULA todos os ponteiros dessas listas são indiretos. Um registro de tipo clock, por exemplo, aponta para um registro do vetor geral, que é um vetor de apontadores. Esse registro aponta para uma das chaves, que faz parte do conjunto de

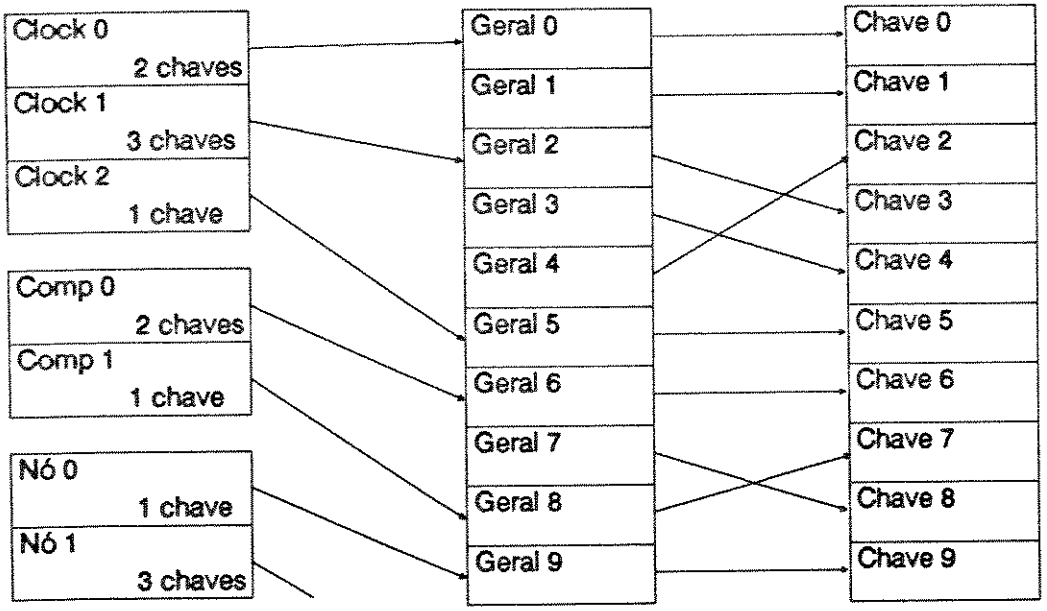


Figura 3.2: *Estrutura das listas apontadas*

chaves comandadas por esse clock, e os subsequentes, para as chaves restantes. O número de chaves apontadas é guardado no registro de clock (Fig. 3.2).

Toda essa estrutura é montada pelo programa GERA para ser usada pelo SIMULA.

Os tipos de dados utilizados pelo SIMULA são mostrados na tabela a seguir. Esses tipos de dados são partilhados pelo programa GERA, por isso sua definição fica em um arquivo separado, para uso na compilação dos 2 programas. No anexo 3 podem ser encontradas as definições completas de cada tipo.

Item_no	Registro dos nós
item_cap	Capacitores
item_switch	Transistores
item_clock	Clocks
item_devices	Buffers, Somadores, Subtratores e Divisores por 2
item_cmp	Comparadores
item_fonte	Fonte de tensão (qualquer)
item_fonte_dc	Fontes DC
item_senoide	Fontes de tensão senoidal
item_sffm	Fonte de tensão senoidal com modulação FM
item_pulso	Fonte de tensão pulsada
item_palavra	Fonte de tensão pulsada, com definição da duração de cada pulso
item_tran	Registro dos parâmetros de simulação

3.1.1 O Uso do Endereçamento Indireto

Esse tipo de endereçamento indireto para os ponteiros consome um certo tempo de CPU para o cálculo do endereço efetivo (sendo uma operação com números inteiros, esse tempo é

pequeno). Essa abordagem tem duas vantagens: economia de espaço, pois um endereço real ocupa 4 bytes (Cada byte tem 8 bits no PC) e um indireto ocupa 2 bytes, e liberdade de tipo, pois todos os ponteiros são de um único tipo (unsigned int). O fato do endereçamento ocupar 2 bytes limita a quantidade de qualquer item a 65536. O programa determina o tipo de registro, para o qual um ponteiro aponta, levando em conta o tipo do endereço base somado a ele. Na linguagem C isso representa uma grande simplificação na passagem de parâmetros entre rotinas e na declaração de tipos de dados.

3.2 Algoritmo

O algoritmo do SIMULA pode ser dividido em 2 partes:

- Na primeira tem-se o loop de simulação propriamente dito.
- Na segunda tem-se o cálculo do valor de tensão de um nó.

Essa divisão dá uma estrutura mais leve ao simulador, pois as duas tarefas são claramente separadas.

3.2.1 Loop de Simulação

No programa principal é lido o arquivo de extensão .TMP, onde estão todos os vetores, arranjados de forma sequencial, e dados sobre a simulação (Número de pontos , intervalo de tempo, etc). O arquivo de dados é aberto e inicializado com as informações sobre o número de pontos e o nome dos nós que vão ser escritos. Todos os clocks são inicializados com o último estado da lista de estados de cada clock, as chaves são operadas pela primeira vez e a rotina *roda_comp()* é chamada para determinar o estado inicial. Para cada ciclo de clock é recalculado o estado do circuito e são gravados os nós de interesse no arquivo de saída. Durante a simulação, o andamento do programa é mostrado na tela através de um relógio.

O pseudo código a seguir representa as ações do loop de simulação:

```
Início
     Inicializa o banco de dados lendo o arquivo *.TMP gerado
     pelo GERA.
     Inicializa as variáveis auxiliares.
     Abre e inicializa o arquivo *.DAT para os dados de saída.
     Inicializa todas as chaves do circuito no estado inicial
     de operação.
     Chama roda_comp(), para gerar os valores iniciais do
     circuito.
     Atualiza o arquivo de dados.
     Para todos os pontos da simulação faz :
          Para todos os clocks faz:
               Atualiza o contador de tempo.
               Se mudou o estado do clock, faz:
                    Atualiza o estado.
                    Opera as chaves relacionadas com esse clock
                    e marca as nets modificadas.
               Fim.
          Fim.
          Chama roda_comp(), para estabelecer os novos valores
          dos nós.
          Atualiza o arquivo de dados.
     Fim.
     Fecha o arquivo de dados .DAT.
Fim.
```

Roda_comp() Na rotina *roda_comp()* calcula-se o valor de tensão das nets que não foram modificadas, as cargas nos capacitores dos circuito são atualizadas, calculando-se as tensões nos nós onde eles estão ligados (incluindo as capacitâncias parasitas), e são operadas as chaves dos comparadores que mudaram de estado. Esse ciclo é repetido enquanto ocorrerem mudanças de estado nos comparadores (ou seja, até que o circuito fique estável), ou até que o número máximo de ciclos pré determinado se repita.

As nets mudam à medida que a operação das chaves muda as interconexões do circuito. Se uma net não se modificou de um ciclo de clock para outro, o seu valor de tensão será igual ao da fonte de tensão ligada a ela. Caso não haja nenhuma fonte ligada, o valor da net não muda. Como mostrado na estrutura do banco de dados, cada nó tem um ponteiro para a net a que ele pertence. Quando a operação de uma chave modifica uma net, esta é marcada como inválida e o valor de tensão associado aos nós que pertenciam a ela têm que ser recalculados.

```
Rotina roda_comp():
     Faz:
          Estabelece os valores de tensão para todas as nets
          ativas:
               Para todas as nets ativas calcula-se o seu valor
               de tensão.
          Para todos os transistores faz:
               Chama pegue_valor(), que retorna as tensões dos
               nós do source e dreno dos transistores.
               Estabelece a carga retida nas capaciâncias dos
               transistores.
          Fim.
          Para todos os capacitores faz:
               Chama pegue_valor(), que retorna as tensões dos
               nós onde estão ligados os capacitores.
               Estabelece a carga retidas nos capacitores.
          Fim.
          Testa quais comparadores mudaram de estado.
          Se algum comparador mudou de estado faz:
               Para todos os comparadores que mudaram de estado
               operar as chaves associadas a eles.
          Fim.
     Enquanto houver mudanças dos comparadores volta.
Fim.
```

3.2.2 Cálculo das Tensões

Para calcular o valor de tensão dos nós, o loop de simulação chama a rotina *pegue_valor*.

Pegue_valor Essa rotina testa se a net a que pertence o nó ainda é válida. Em caso positivo ela volta com o valor armazenado, caso contrário, a rotina requisita uma net livre e chama a rotina *valor_net*, para montar a nova net e calcular o valor de tensão associado a ela. Isso permite que o valor de tensão associado a uma net seja calculado apenas uma vez. A grande maioria dos acessos a essa rotina se dá com nets ainda válidas, o que reduz enormemente seu tempo de execução.

```
pegue_valor()
     Se a net a que pertence o nó ainda é válida faz:
          Retorna o valor de tensão da net.
     Pega o número de uma net livre (que não tem nós). Se não
     houver nenhuma, uma nova é criada.
     Chama valor_net() para montar a nova net e volta com o
     valor de tensão da fonte ligada à net (se houver) ou às
```

```
        suas capacitâncias.
        Se a net não tem uma fonte ligada a ela faz:
             Calcula o divisor capacitivo e acha a tensão da net.
        Grava o tipo e a tensão da net.
        Testa se a tensão extrapolou os valores permitidos.
        Retorna o valor.
Fim.
```

Valor_net A rotina *valor_net* monta uma net para determinado nó, percorrendo todos os nós ligados, no momento, a esse nó. Ela usa um algoritmo de busca em grafos com *loops* [16]. A rotina recebe como entrada um registro do tipo net e um nó. Ela marca esse nó como já analisado e testa se há alguma fonte ligada ao nó. Em caso positivo, chama a rotina *valor_no*, para determinar a tensão associada ao nó; em caso negativo, soma as capacitâncias ligadas ao nó. Depois varre as chaves que estão conectadas ao nó, à procura das que estão comutadas. Ao encontrar uma chave comutada ligada a um novo nó ainda não marcado, a rotina se auto chama, passando a atual net e o novo nó como parâmetros. A recursão termina quando a rotina não encontra mais chaves comutadas ou nós não marcados.

Depois de varridos todos os nós ligados ao nó atual, são retornados os tipos das sub-nets associadas a eles. Elas são classificadas pelo tipo das fontes ligadas a elas em: nets com fonte independente, nets com fonte dependente, nets apenas com capacitores e nets abertas. A depender do tipo das sub-nets que retornam e do tipo do nó atual, a rotina junta os valores em um só, retornando esse valor e testando condições proibidas, como duas fontes ligadas na mesma net, etc.

O valor de tensão associado à net será o valor de tensão de qualquer fonte que esteja ligada a ela ou, caso não haja nenhuma fonte, o cálculo da distribuição das cargas entre os capacitores.

O pseudo-código da rotina valor_net é:

```
valor_net()
     Testa o tipo de nó para o qual a rotina foi chamada:
          Se tipo=Capacitor faz:
               Soma a capacitância do nó ao total da net.
               Soma a carga do nó ao total da net.
          Fim.
          Se tipo = Fonte de tensão faz:
```

```
                Tensão da net igual ao valor da fonte calculado
                pela rotina valor_no().
            Fim.
        Fim.
        Para todas as chaves ligadas ao nó atual faz:
            Acha a que nó a chave está conectada e suas
            capacitâncias parasitas, através da rotina conexão.
            Se a chave está ligada e o novo nó ainda não foi
            computado na net, faz:
                Chama a rotina valor_net() recursivamente para o
                novo nó.
                Testa o tipo do nó atual com o da sub-net:
                    Se um deles é aberto, não é processado.
                    Se os dois são capacitores, soma as suas
                    capacitâncias.
                    Se um deles é uma fonte, o seu valor é
                    adotado para a net.
                    Se os dois são fontes, é gerado um erro e o
                    programa pára.
                Fim.
            Fim.
        Fim.
Fim.
```

Valor_no A rotina *valor_no* calcula o valor de tensão associado a um nó. Ela recebe como entrada o número do nó. No registro desse nó existe um ponteiro para a fonte que está ligada a ele. Se esta fonte for independente, a rotina calcula seu valor de tensão naquele momento e retorna. Se esta fonte for dependente, a rotina chama *pegue_valor* para fornecer os valores de tensão das entradas da fonte (Somador, Buffer, etc), calcula a tensão de saída e retorna esse valor.

Como essa rotina chama *pegue_valor*, que pode ter iniciado a seqüência que acabou por acionar essa rotina, poder-se-ia ter a impressão que haveria um loop dentro do programa. Isso não ocorre porque os nós que pertencem à net sob análise são marcados e, se a rotina *valor_net* encontra um nó já marcado com o número de outra net, é gerada uma mensagem de erro, avisando que houve um loop através de uma fonte dependente (que muito provavelmente oscilaria num caso real) e o programa pára.

O pseudo-código desta rotina segue:

```
valor_no()
     Testa o tipo de fonte que está ligado ao nó chamado na
     rotina:
     Se tipo= Fonte independente faz:
          Acha tipo da fonte (DC, Senóide, SFFM, Pulso ou PWL).
          Calcula o valor de tensão atual.
     Fim.
     Se tipo= Fonte dependente faz:
          Acha o tipo da fonte (Somador, Subtrator, Divisor ou
          Buffer)
          Chama a rotina pegue_valor() para determinar o valor
          de tensão nos terminais de entrada.
          Calcula a tensão de saída.
     Fim.
Fim.
```

4

A INTERFACE GRÁFICA

The purpose of computing is insight, not numbers.
Richard Hamming.

Visualização é um termo usado na indústria, desde a publicação, em 1987, do relatório da National Science Foundation *Visualization in Scientific Computing* [17]. Ele representa muito mais que gráficos de computador e processamento de imagens, sendo uma forma de comunicação que transcende as fronteiras da tecnologia.

O dilúvio de dados gerado por programas que simulam processos físicos tornou impossível para os usuários, quantitativamente, examinar mais que uma pequena fração de uma dada solução. Isto significa que é impossível investigar a natureza qualitativa global da solução numérica. Com o advento dos displays de alta resolução, os usuários podem converter grandes massas de dados em imagens coloridas [18].

Pensando nisso, os simuladores modernos oferecem interfaces gráficas para a apresentação dos resultados da simulação (Ex: PROBE do PSpice). A interface gráfica do QUANTICO é o

módulo OSC. Ele se comporta como se fosse um osciloscópio de 8 canais. Esta forma de visualização facilita a utilização do programa, considerando que os usuários têm esperiência com o uso de osciloscópios.

4.1 Estrutura do programa

O OSC lê o número de pontos, o intervalo de tempo entre eles e o número de nós existentes. Com esses dados, ele testa se há espaço suficiente na memória, multiplicando o número de pontos pelo números de nós e pelo tamanho de um número real (32 bits), comparando esse valor com a memória disponível. Havendo memória o OSC aloca a quantidade necessária e lê o nome de cada nó. Em seguida, lê os valores para cada nó medido, até terminar o número de pontos.

Dentro da filosofia *User Friendly*, adotada para o OSC, tem-se o uso extensivo de *pop-up* menus. O programa apresenta na tela básica uma janela de apresentação (que seria a tela do osciloscópio) e os comandos disponíveis (Fig. 4.2). Quando é acionada uma das opções, um menu é aberto sobre uma porção da tela (Fig. 4.3). Nesse menu encontram-se informações adicionais sobre a opção e seus comandos particulares.

Os algoritmos dos menus foram copiados do programa EDCHIP [19] [20], um programa para projetos de máscaras de circuitos integrados, sendo que o protótipo utilizou a própria biblioteca do EDCHIP. Os algoritmos foram reescritos de Pascal para C, para fazer o gerenciamento dos menus. Para isso, basta escolher a rotina apropriada, que depende do tipo de menu que se deseja, e enviar a ela as mensagens dos menus. As rotinas retornam a opção escolhida pelo usuário ou o valor digitado por ele (a depender do tipo do menu).

A estrutura de dados utilizada pelo OSC pode ser vista na Fig. 4.1. A matriz dos pontos tem que ser dinâmica, pois é impossível saber a priori, as dimensões da matriz de dados. Poder-se-ia usar uma matriz de tamanho *default*, mas isso desperdiçaria muita memória. Para se criar uma estrutura de matriz dinâmica de dimensão dois, foi necessário criar um vetor

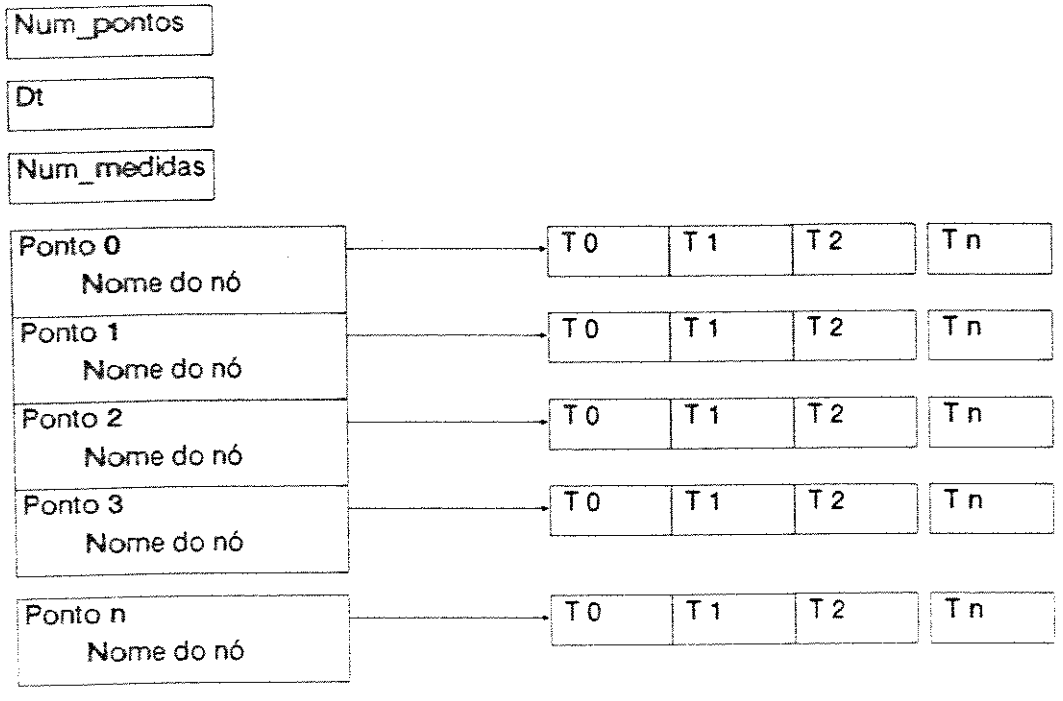


Figura 4.1: *Estrutura de dados*

de ponteiros que apontam para vetores de números reais. Assim, a primeira dimensão da matriz indica uma posição no vetor de ponteiros e a segunda, uma posição dentro do vetor de números.

O funcionamento do corpo do programa é simples. Existe uma rotina que escreve na tela do osciloscópio (*draw_tela*), usando os parâmetros que estiverem definidos no momento. O programa principal recebe os comandos do usuário, através das opções mostradas na tela e dos menus, modifica os parâmetros de controle (período, ganho dos canais, etc) de acordo com esses comandos e chama a rotina *draw_tela* para recompor a imagem do osciloscópio (caso os parâmetros modificados afetem essa imagem).

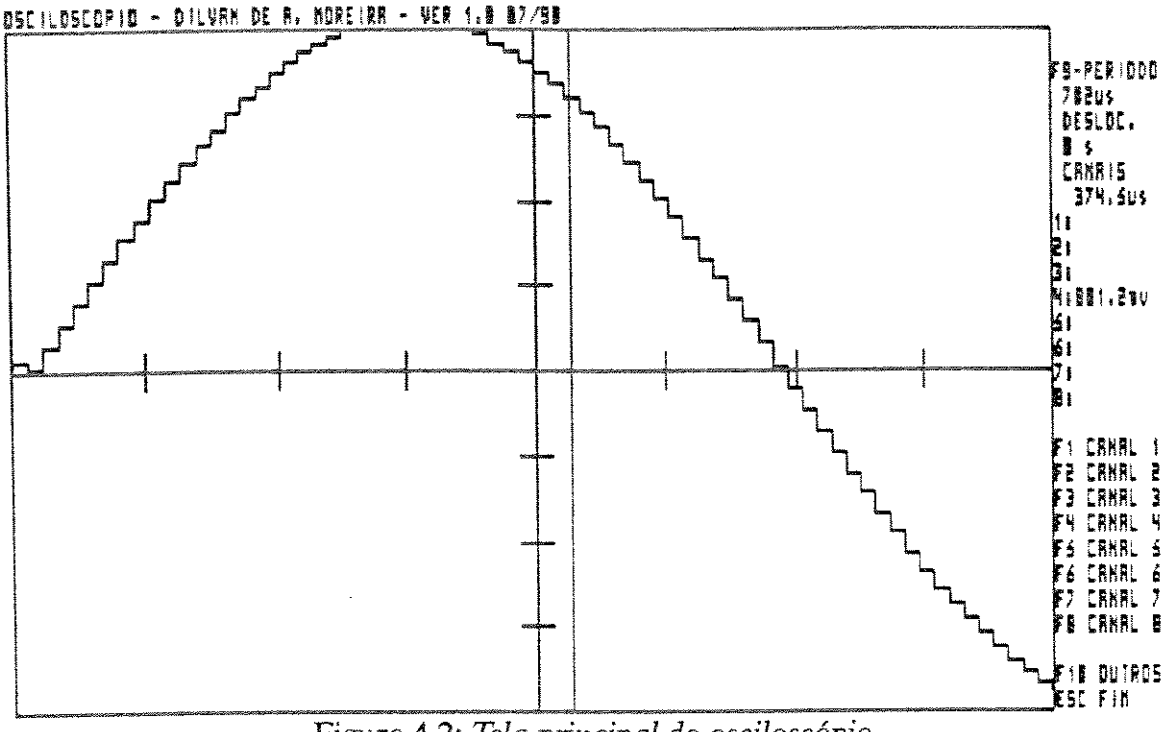


Figura 4.2: *Tela principal do osciloscópio*

4.2 Descrição do funcionamento

Após a leitura dos dados, o programa mostra o nome de todos os nós disponíveis e, ao se pressionar uma tecla qualquer, entra no modo de operação mostrado na Fig. 4.2. No alto da tela (Fig. 4.2) tem-se a versão corrente do programa e, no lado direito, o menu com os comandos possíveis. Para acessá-los basta pressionar a tecla de função indicada. No centro tem-se o que seria a tela do osciloscópio. O traço que pode ser visto próximo ao meio dela é o cursor de tela. Ele marca o ponto a que correspondem as leituras de tensão mostradas no lado direito da tela. Esse cursor pode ser deslocado com as teclas de direção, movendo-se entre os valores medidos, em saltos. A amplitude desses saltos pode ser regulada. As amplitudes máximas da tela do osciloscópio correspondem às leituras de -1 e 1 volts, para canais com ganho 1.

Todos os valores mostrados na tela usam caracteres para indicar constantes multiplicativas. Isso visa facilitar a leitura. As constantes são:

p = 1E-12 n = 1E-9 u = 1E-6 m = 1E-3
K = 1E3 M = 1E6 G = 1E9 T = 1E12

Os comandos observados no lado direito têm as seguintes opções:

F9 - Periodo Abre um menu onde pode ser digitado um novo tempo para o período. Ela seria equivalente ao botão de frequência do osciloscópio.

Desloc Mosta o tempo que equivale à origem da onda na tela (lado esquerdo). Esse valor pode ser modificado pelas teclas de direção esquerda e direita, se a *flag* cursor = desloc. Isso equivale ao botão de ajuste de posição horizontal do osciloscópio.

CANAIS Mostra a posição do cursor de tela no tempo. O cursor pode ser ser movido pelas teclas de direção esquerda e direita se a *flag* cursor = tela. Isso equivale ao uso do cursor nos osciloscópios digitais.

F1 CANAL 1 a F8 CANAL 8 Essas opções permitem modificar os atributos de cada canal. Ao se acionar uma das teclas de F1 a F8 aparece o menu da Fig. 4.3. No alto desse menu aparece o número do canal, em seguida vêm o nó ao qual esse canal está associado, o ganho que esse canal aplica ao sinal de entrada, o nível DC do sinal, a *flag* DESLIGADO - LIGADO e a opção de saída do menu. AS 3 primeiras opções seriam equivalentes num osciloscópio, ao ponto do circuito onde está ligada a ponta de prova, ao ganho vertical e ao botão de posicionamento vertical, respectivamente.

Para se modificar qualquer dos parâmetros, ou sair do menu, basta posicionar o cursor (Barra com texto invertido) sobre a opção, usando as teclas de direção, e teclar um ENTER. No caso do ganho, do nível DC e do nó ao qual o canal está

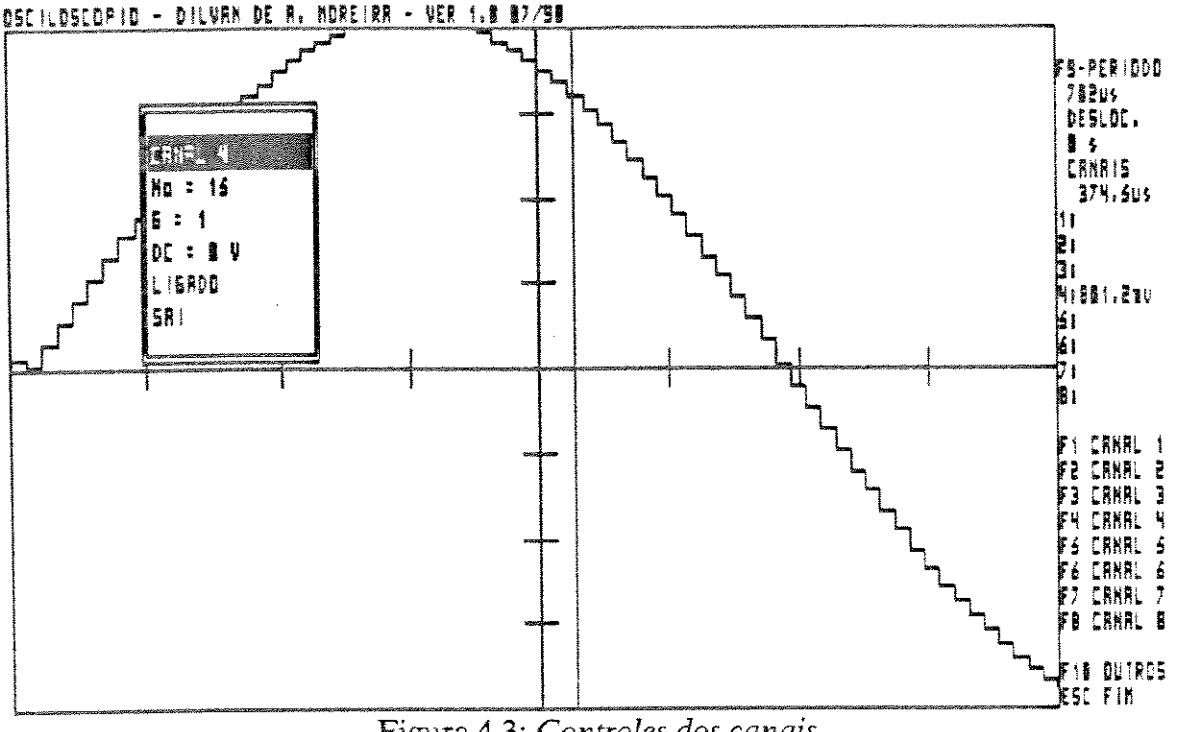


Figura 4.3: *Controles dos canais*

associado, será pedido um novo valor. Para ligar ou desligar o canal e para sair, basta um ENTER sobre a opção.

Se algo foi modificado, a tela será apagada e refeita, de acordo com os novos parâmetros.

F10 OUTROS Abre um menu auxiliar Fig. 4.4. Nesse menu existem 4 opções:

CURSOR = Ao ser selecionada, essa opção muda o status do cursor. Quando ela está em TELA as teclas de direção (<- e ->) movem o cursor de tela. Quando ela está em DESLOC, essas teclas mudam o valor do deslocamento de tela (que equivale ao posicionamento horizontal do osciloscópio), permitindo o deslocamento da tela pelo sinal.

PASSO CURSOR Essa opção modifica o passo do cursor, ou seja o número de saltos que ele dá ao se pressionar uma das setas. Em 1 o cursor pulará de uma amostragem

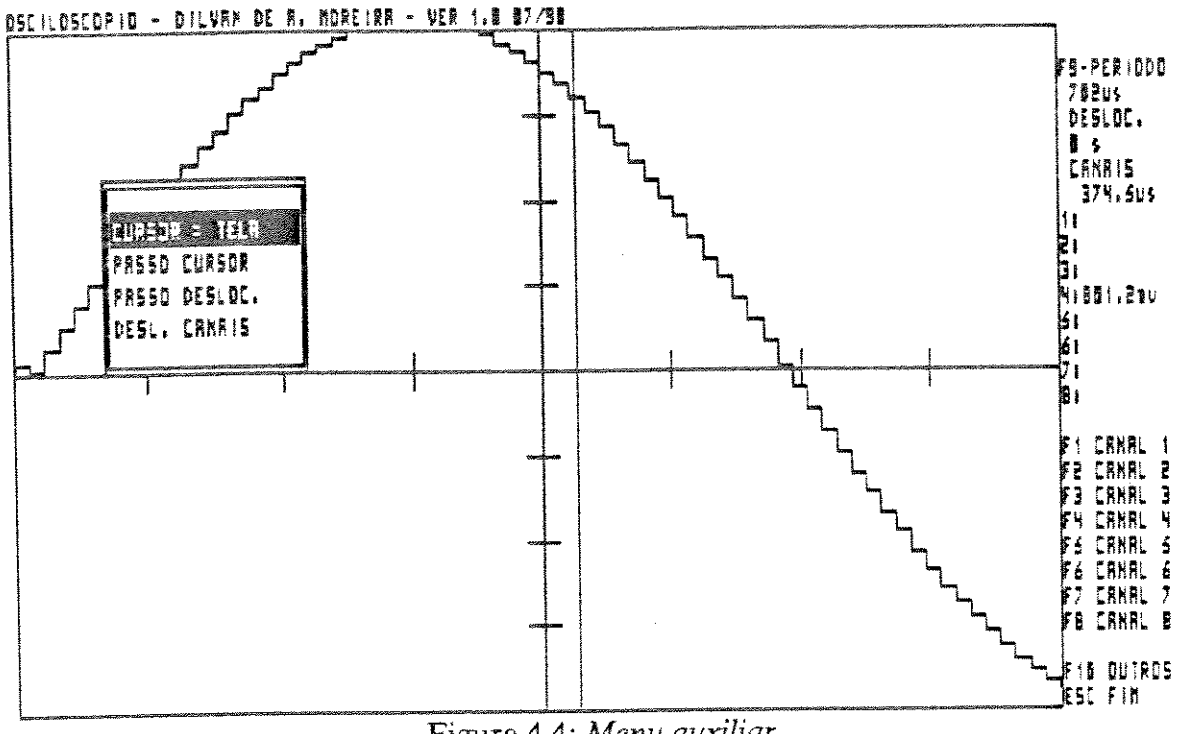


Figura 4.4: *Menu auxiliar*

para a imediatamente seguinte; em 2, pulará uma amostra, e assim por diante.

PASSO TELA Essa opção permite modificar o passo da tela , ou seja, o número de pulos que a tela se moverá, ao se pressionar uma das setas. De maneira análoga à opção anterior, a tela mover-se-á o espaço de uma amostra, para 1 passo; moverá duas amostras, para 2 passos, e assim por diante.

DESL. CANAIS Desliga todos os canais.

ESC A tecla escape finaliza o programa e volta para o nível do sistema operacional.

5

RESULTADOS EXPERIMENTAIS

> *[Rule of Credibility] The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time. T. Cargill - Bell Labs.*

Para avaliar o funcionamento do QUANTICO alguns circuitos foram simulados. Utilizamos como referência o programa SWITCAP nos casos dos circuitos em que este se aplica. As listagens desses circuitos estão no Anexo 4.

5.1 Simulação de um amplificador de tensão

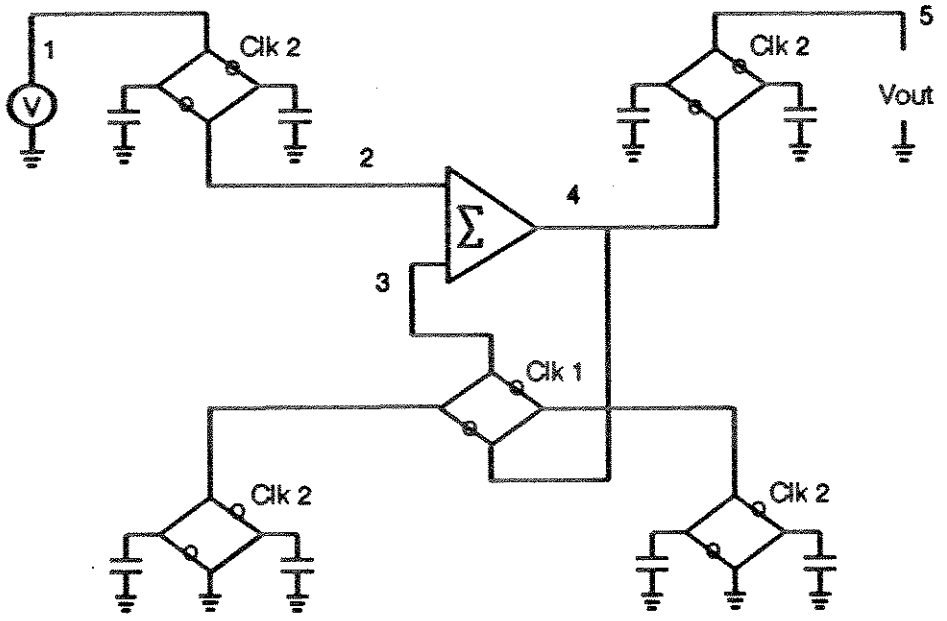
Um desses exemplos é o circuito mostrado na Fig. 5.1 que realiza a função de um amplificador de tensão. Os resultados de ambas simulações são mostrados na Fig. 5.2:

A forma de onda no traço superior da tela corresponde ao sinal de entrada do amplificador (tensão nó 1). Na parte inferior são mostradas as formas de onda que correspondem à saída

do amplificador. O traço de linha pontilhada, produzido pelo SWITCAP, coincide com o traço de linha cheia produzido pelo QUANTICO. Pode-se notar, entretanto, a não coincidência dos sinais nos primeiros ciclos de clock. Essa diferença pode ser explicada por dois motivos:

Primeiro, porque o QUANTICO mostra no primeiro ciclo o valor de tensão correspondente ao instante 0-, ou seja entre -dt e 0, enquanto que o SWITCAP mostra o sinal a partir do instante 0+. Esse ciclo a mais nas formas de onda do QUANTICO é compensado por um atraso correspondente a um ciclo de relógio na forma de onda de entrada do SWITCAP (Listagem no Anexo 4).

Segundo, porque o QUANTICO mostra o retardo de meio período do clk2 para que o sinal de entrada alcançe a saída. Enquanto isso não acontece, é mostrado na saída o valor zero, que é



Cada lado dos losangos representa uma chave, os lados marcados com um círculo operam com sinal baixo. Ao lado de cada losango tem-se o clock que o comanda. A amplificação de tensão é dada por f1 / f2, onde f1 e f2 são as frequências dos relógios CLK1 e CLK2.

Figura 5.1: *Amplificador de tensão*

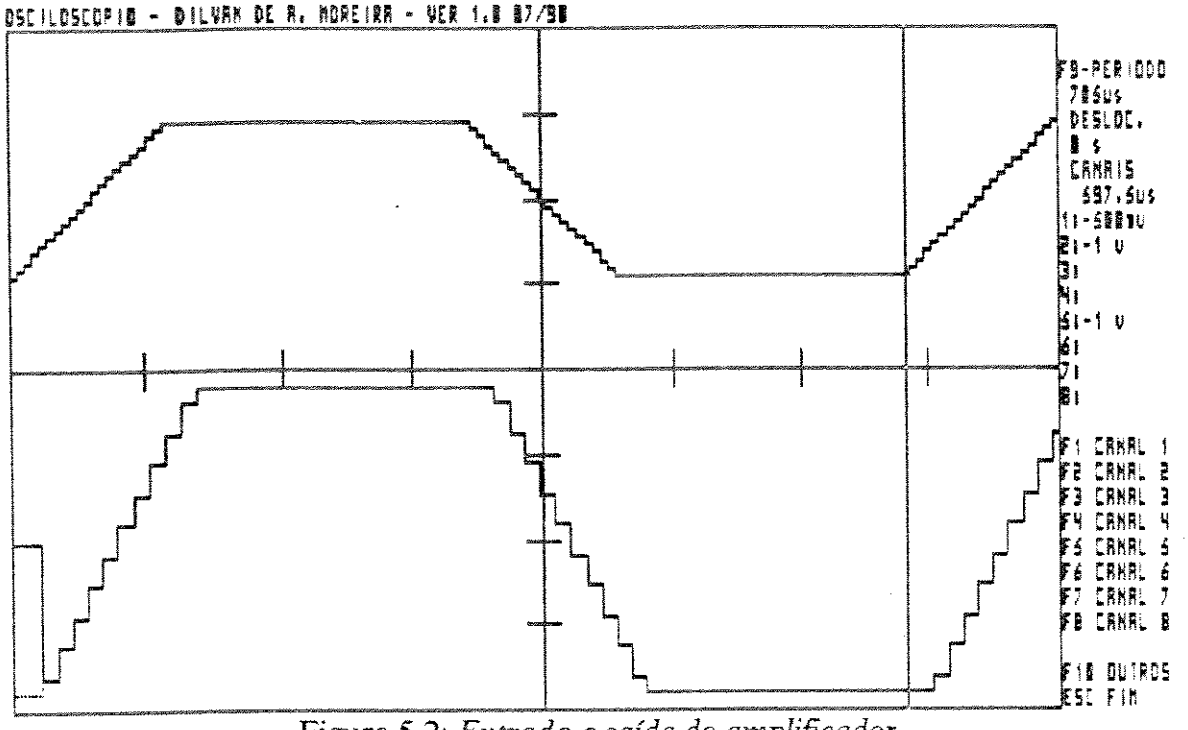


Figura 5.2: *Entrada e saída do amplificador*

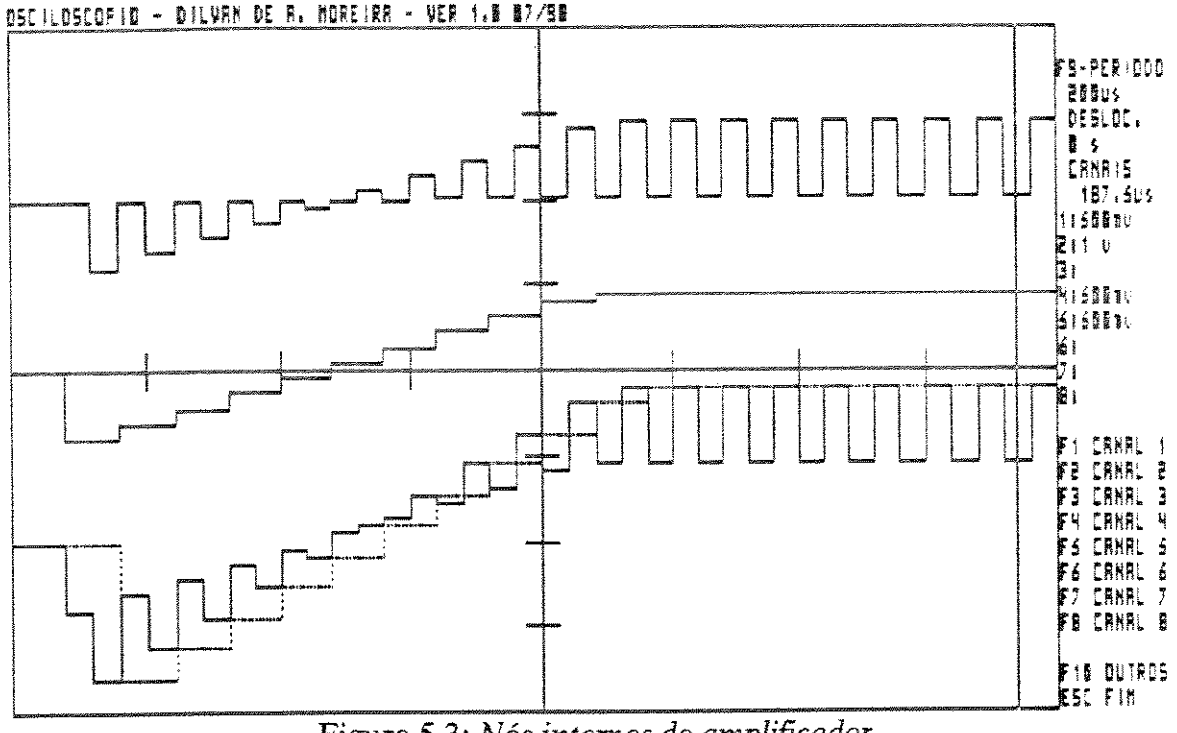


Figura 5.3: *Nós internos do amplificador*

armazenado em todos os capacitores quando a simulação começa.

Para uma melhor visualização do funcionamento do circuito, podem ser vistos na Fig. 5.3 as formas de onda em outros três pontos do circuito. De cima para baixo tem-se: a entrada do somador (Nó 2), que vem da entrada do circuito; a entrada do somador (Nó 3), que vem da realimentação; e a saída do somador (Nó 4), que vai ser amostrada na saída. Sobreposto ao último sinal, em pontilhado, tem-se o sinal de saída (Nó 5), pode-se observar a soma dos sinais dos nós 2 e 3 em 4, e o efeito da amostragem no nó 5.

5.2 Simulação com o efeito de injeção de cargas

Para simular o efeito de injeção de cargas no SWITCAP é recomendado, no manual do usuário do SWITCAP [10], o subcircuito da Fig. 5.4 para ser usado no lugar de cada chave analógica (Foram usados capacitores de 1 pF). Trata-se de uma aproximação grosseira uma vez que leva em conta apenas a existência de capacitâncias fixas entre o gate-source, gate-dreno, dreno-substrato e source substrato. O QUANTICO, como já foi mostrado, leva em consideração as capacitancias parasitas e efeitos como o desaparecimento da capacitância do canal, o valor de VT, etc.

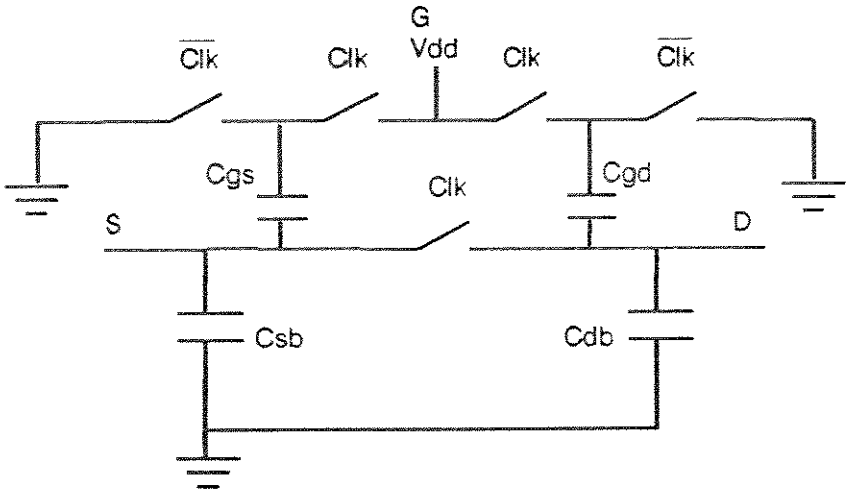


Figura 5.4: *Modelo para chave analógica*

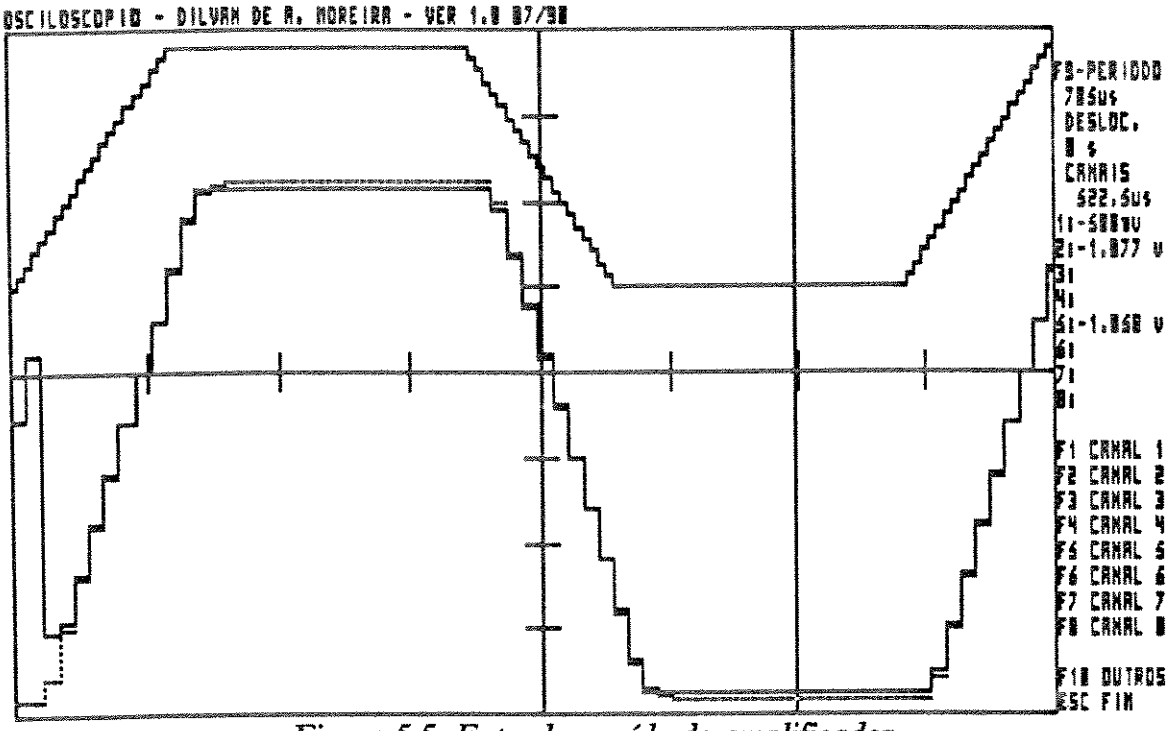


Figura 5.5: *Entrada e saída do amplificador*

O resultado de ambas simulações, incluindo o efeito de injeção de cargas, é mostrado na Fig. 5.5.

Na parte de cima da figura tem-se o sinal de entrada; abaixo estão o sinal de saída gerado pelo QUANTICO (traço cheio) e o sinal gerado pelo SWITCAP (pontilhado). A diferença observada entre eles pode ser atribuida à diferença no modelamento do efeito de injeção de cargas nos dois simuladores. O modelamento do QUANTICO, sendo mais sofisticado, produz um resultado mais completo que o outro.

Também, nessa forma de onda existe uma diferença no início devido às mesmas causas comentadas anteriormente. O efeito gerado é acrescido pela presença das capacitâncias parasitas. No caso do QUANTICO, os capacitores parasitas são arbitrariamente carregados com zero Coulombs no início da simulação. Para que estas capacitâncias assumam os valores de cargas em regime, passam se alguns ciclos de clock.

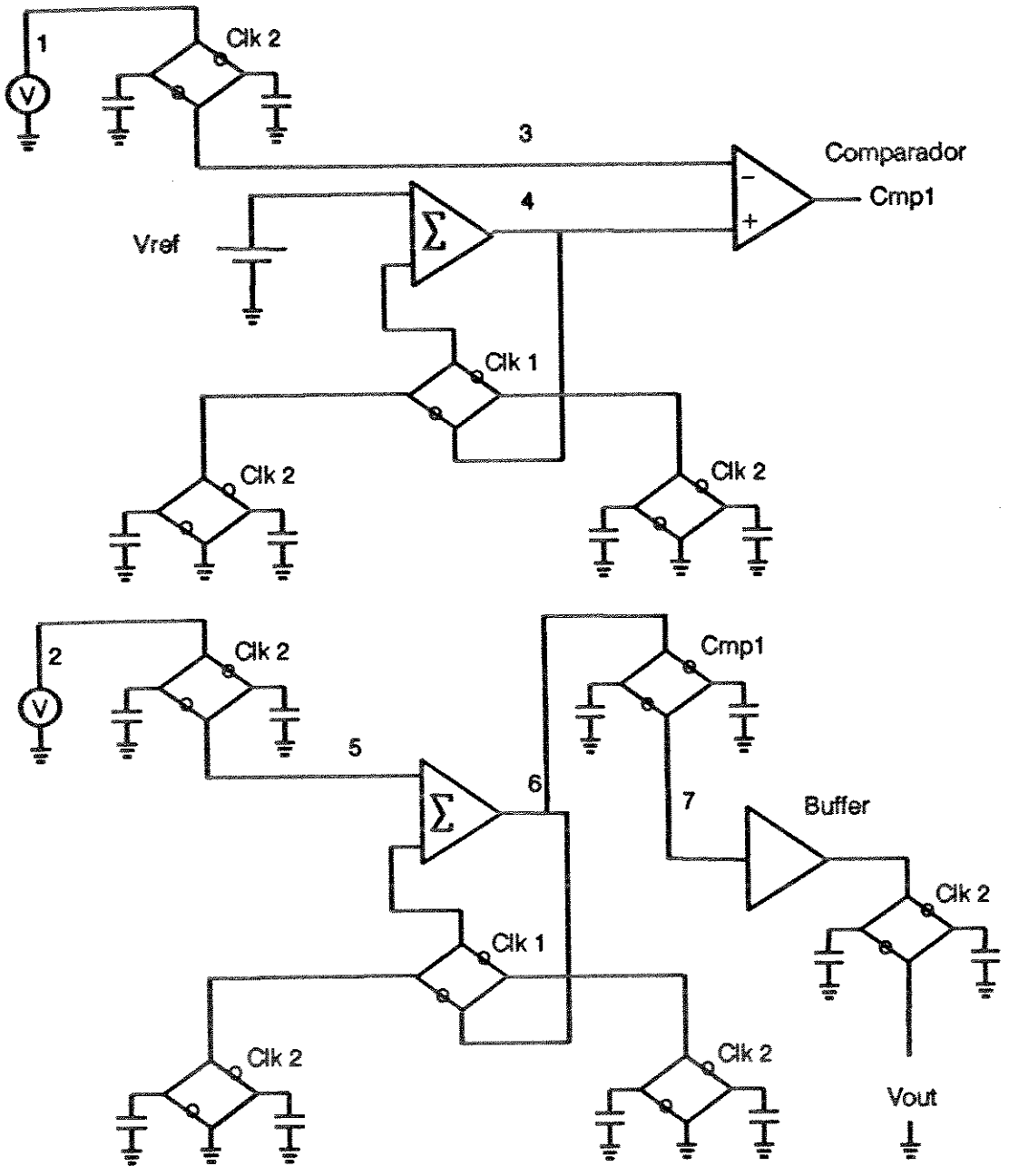


Figura 5.6: *Circuito multiplicador*

O tempo de simulação entre o QUANTICO e o SIMULA, nos dois casos, foi da mesma ordem de grandeza.

5.3 Simulação de um multiplicador de tensões

A Fig. 5.6 mostra um esquema de um multiplicador de dois quadrantes. Este circuito, mais complexo que o circuito mostrado na Fig. 5.1, inclui o acionamento de chaves através de comparadores de tensão. Esta característica particular do circuito impede a sua simulação através do SWITCAP; aliás, esta foi uma das motivações para o desenvolvimento do QUANTICO.

O resultado da simulação deste circuito é mostrado na Fig. 5.7. Os traços superiores correspondem às tensões dos sinais de entrada V(1) e V(2) e o traço mais inferior corresponde a saída Vout.

Algumas formas de onda em nós internos do circuito são mostradas nas figuras 5.8 e 5.9. A Fig. 5.8 mostra a tensão de referência Vref ser acumulada (V(4)) até atingir a tensão V(3) (Parte de cima), que é uma amostragem da tensão V(1). Quando isto acontece o comparador comuta, gerando pulsos

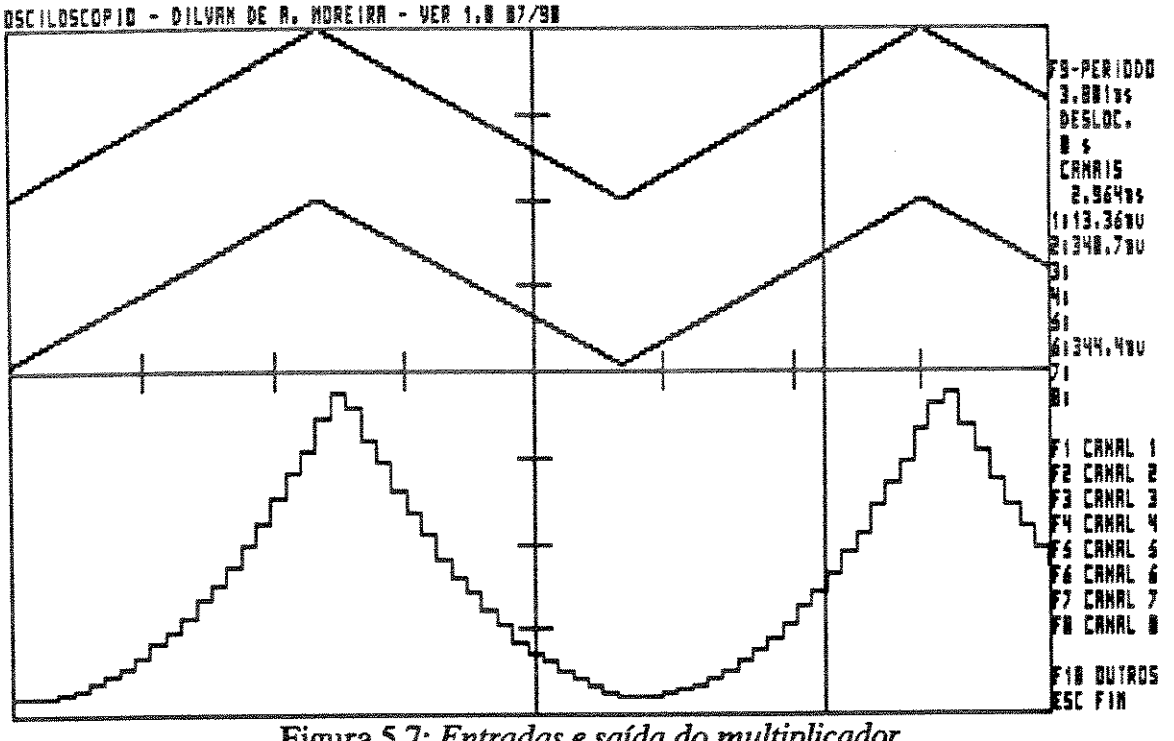


Figura 5.7: *Entradas e saída do multiplicador*

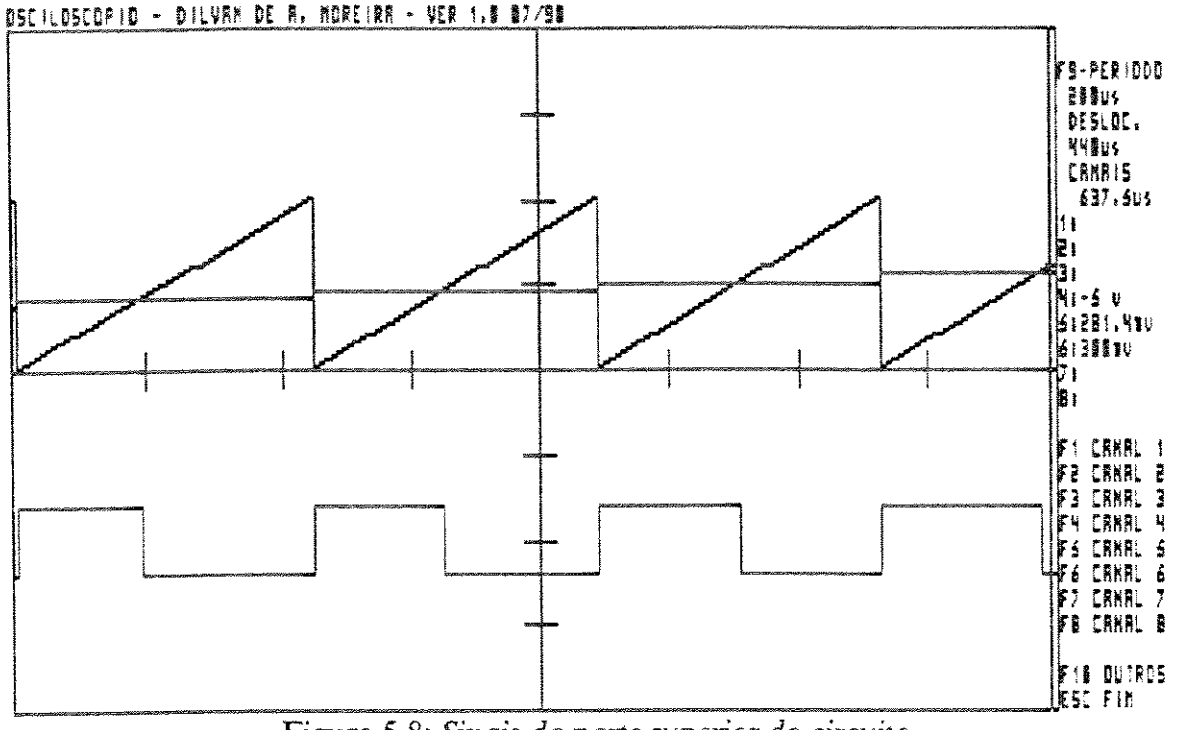


Figura 5.8: *Sinais da parte superior do circuito*

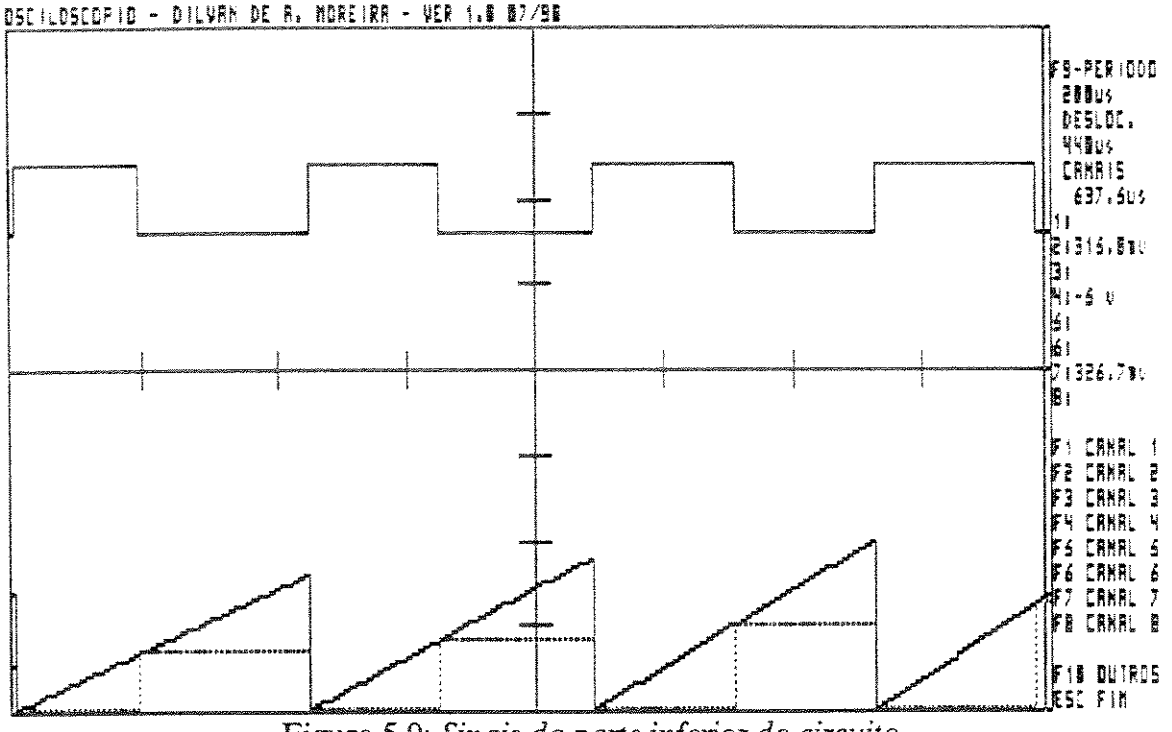


Figura 5.9: *Sinais da parte inferior do circuito*

(Parte de baixo Fig. 5.8). As larguras desses pulsos são assim proporcionais à tensão V(1) quantizada por Vref.

A Fig. 5.9 mostra a tensão V(5), que é uma amostragem da tensão V(2), ser também acumulada concomitantemente com Vref. Quando o comparador comuta (Parte de cima), o valor de V(5) acumulado, que é V(6), é amostrado num capacitor V(7) (Parte de baixo, V(7) pontilhado) e deste passa a saída na mudança do clock 2. Como o número de vezes que V(5) é acumulada, antes de ser amostrada para a saída depende de V(3) / Vref e Vref é constante, tem-se na saída V(6) = V(3) . V(5) ou V(1) . V(2).

Conclusão

De tal forma nós podemos dizer que a porta agora está aberta, pela primeira vez, para um novo método repleto de numerosos e maravilhosos resultados que em anos futuros comandará a atenção de outras mentes. Galileu Galilei.

O QUANTICO foi desenvolvido para auxiliar o desenvolvimento de projetos dentro de uma classe específica de circuitos. Procurou-se explorar todas as possibilidades oferecidas por essa classe de circuitos, para permitir a inclusão na simulação do maior número de parâmetros de interesse.

O QUANTICO permite a inclusão de elementos parasitas aos modelos das chaves analógicas, diferindo da maioria dos simuladores para SCN's, que consideram como primitivas dispositivos ideais. Nas SCN's isso causa erros nos sinais simulados, dificultando o dimensionamento dos elementos em circuitos reais.

Sua estrutura, em 3 partes, com funções bem definidas, e o fato de ter sido escrito em C, favorecem a sua portabilidade e futuras atualizações.

O fato do QUANTICO trabalhar com estruturas de dados que representam a topologia do circuito, além de diminuir espaço e aumentar a rapidez, permite a inclusão de avisos quando condições impróprias aparecem durante a simulação. Isso contribui para que não existam resultados de simulações como no SPICE, por exemplo, onde às vezes é possível aparecer uma corrente de milhares de amperes num simples transistor de sinais. O pós processador gráfico (OSC) permite a análise rápida dos dados e grande facilidade de operação, uma vez que seus comandos se parecem com os dos osciloscópios e por usar menus na sua operação.

Listam-se abaixo alguns pontos onde deve ser dispendido algum esforço, visando melhorar o desempenho do programa:

- Na parte referente ao modelamento do efeito de injeção de cargas, o fenômeno da formação do canal do transistor precisa ser melhor estudado. A bibliografia consultada enfoca bem a abertura do transistor (extinção do canal), mas não traz muita informação sobre a operação inversa. O próprio fenômeno da abertura das chaves analógicas ainda é tema de pesquisa.
- Na parte referente às facilidades que o programa oferece à análise, poder-se-ia incluir no módulo OSC: um analisador de espectro, para permitir medidas como distorção harmônica, banda passante, etc; uma opção para permitir a soma e subtração entre os canais; a possibilidade de utilização de *mouse*.
- Na parte referente à flexibilidade do programa, seria muito interessante que o módulo GERA, além de ler os arquivos no padrão SPICE, aceitasse descrições em EDIF 2.0 (*Electronic Design Iterchange Format*) [24], que é uma linguagem de descrição projetada para ser um padrão de intercâmbio de projetos eletrônicos. Ela é usada atualmente em várias estações de trabalho. Isso permitiria o uso do QUANTICO em ambientes de projeto já existentes para estações de trabalho.

Todas essas características poderiam ser estudadas numa futura transferência do QUANTICO para o ambiente UNIX de uma estação de trabalho.

ANEXOS

ANEXO 1

MANUAL DO QUANTICO

I.1 Introdução

O manual do QUANTICO tem o objetivo de servir como um guia ao usuário do programa. O QUANTICO roda em computadores compatíveis com o IBM PC, com 512 Kilobytes de memória e com sistema operacional MS-DOS versão 2.0 ou posterior. O programa necessita de tela gráfica, suportando os padrões CGA, EGA, VGA ou Hércules.

O sistema é dividido em 3 módulos: GERA, SIMULA e OSC. Para a instalação é necessário apenas copiar todos os programas para o sub-diretório de trabalho, em disco rígido ou flexível. Os arquivos de entrada devem ter extenção NOME_ARQ.CIR. São gerados um arquivo intermediário, NOME_ARQ.TMP, e o arquivo final, que contém os dados da simulação, NOME_ARQ.DAT.

Para executar o programa, basta digitar:

```
C:> QT NOME_ARQ
```

Esse comando fará toda a simulação e chamará o módulo osciloscópio (OSC). Se, depois da simulação, se desejar rever os dados deve-se digitar:

```
C:> OSC NOME_ARQ
```

O conteúdo do diskete do QUANTICO :

QT.BAT	Arquivo de comandos para rodar todos os módulos.
GERA.EXE	Lê o arquivo de entrada e gera arquivo .TMP para o simulador SIMULA.EXE.
SIMULA.EXE	Faz a simulação propriamente dita a partir do arquivo .TMP, gerando o arquivo de dados .DAT.
OSC.EXE	Lê o arquivo de dados .DAT e mostra os valores como se fosse um osciloscópio multicanais.
CGA.BGI	Arquivo gráfico para uso da placa CGA
EGAVGA.BGI	Arquivo gráfico para uso das placas EGA e VGA
HERC.BGI	Arquivo gráfico para uso da placa Hércules

I.2 Dados

Nesta seção será mostrado como descrever um circuito para o QUANTICO. No final deste anexo são mostrados alguns exemplos. O QUANTICO segue, basicamente, a sintaxe da linguagem de descrição usada no programa SPICE [21]. Sendo assim, usuários familiarizados com esse programa não terão maiores dificuldades.

Algumas considerações gerais:

- O QUANTICO distingue as letras minúsculas das maiúsculas. Todas os comandos são definidos em letras maiúsculas. As minúsculas são ignoradas.

- A primeira linha é o título e pode conter qualquer tipo de caracter. Ela é ignorada e só existe para manter a compatibilidade com o SPICE.
- A última linha deve ser ".END".
- Linhas de comentários são marcadas com "*" na primeira coluna e podem conter qualquer texto.
- O SPICE permite linhas de continuação (iniciadas por "+"). Isso não é possível no QUANTICO.
- A ordem das linhas não tem importância. Exceto pela linha de título, definição de subcircuitos e a linha ".END", .
- O número de brancos entre itens não é significativo. Tabs são equivalentes a brancos.
- Alguns nós são necessariamente conectados ao terra (Ex: o segundo nó dos capacitores). Eles se comportam como se estivessem ligados a uma fonte DC de 0 V.

I.2.1 Nomes

Nomes podem conter até 12 caracteres, números ou letras, com exceção do primeiro, que deve ser obrigatoriamente uma letra.

I.2.2 Nós

Os nós podem ser nomes ou números, não necessitando ser sequenciais. Exemplos: 0, 1, 345, ENTRADA, SAIDA, etc.

I.2.3 Valores

Os valores são escritos em notação padrão para ponto flutuante, com escalas e sufixos opcionais. Exemplo: 1, 1.02, 1E23, 1UF, 1E-10NF, etc.

Os sufixos reconhecidos pelo QUANTICO são:

P = 1E-12 N = 1E-9 U = 1E-6 M = 1E-3

K = 1E3 MEG = 1E6 G = 1E9 T = 1E12

I.2.4 Dispositivos

Cada dispositivo no circuito é representado por uma linha que não começa por ".". A primeira letra do nome de um dispositivo determina o seu tipo. Exemplos : "R" - resistores, "C" - capacitores, "M" - transistores MOS, etc. A ordem dos dispositivos não tem importância.

I.3 GUIA DE REFERÊNCIA

Esta seção mostra a lista de todos os comandos aceitos pelo QUANTICO. Uma descrição completa é dada para cada comando.

A notação usada :

Item	Exemplo	Descrição
nome	C1	String alfanumérica, menor ou igual a 12.
sufixo escala	P	P, N, U, M, K, MEG, G, T
sufixo unida.	F	Letra indicando a unidade de um número.
maiúsculas	.MODEL	Requer essa string literalmente.
real	1E-3	Número real com sufixos de escala e unidade opcionais.
int	2	Número inteiro.
()	(nome <Modelo>)	Agrupamento.
" "	"("	Indica que o símbolo faz parte do comando.
< >	<Modelo>	Indica comentário.
[]	[COX=real]	Indica opcionalidade.
{}	{nome}	Indica possível repetição.

Título

(qualquer texto)

Exemplo: `AMPLIFICADOR GANHO 2`

Ao contrário dos outros comandos, o título é identificado por sua posição. Ele vem no início do arquivo e é ignorado.

Somador

Anome (nome <nó-e1>) (nome <nó-e2>) (nome <nó-s>)

Exemplos:
```
ADDER1 5 4 SAIDA
A2 6 3 4
```

Os nós *nó-e1* e *nó-e2* representam as entradas do somador e *nó-s* a saída. A tensão de saída V(nó-s) é igual a V(nó-e1) + V(nó-e2).

Buffer

Bnome (nome <nó-e>) (nome <nó-s>)

Exemplos:
```
BUFF1 1 30
B2 7 3
```

O *nó-e* representa a entrada do buffer e *nó-s*, a saída. A tensão de saída V(nó-s) é igual a V(nó-e).

Capacitor

Cnome (nome <nó+>) (nome <nó->) (real <capac>) [V = real]

Exemplos:
```
C1 ENTRADA 0 20PF
CLOAD 7 3 15E-3 V=0
```

O *nó+* representa o ponto do circuito onde será ligado o capacitor; o *nó-* será considerado ligado ao terra. *Capac* é o valor da capacitância e V é a tensão inicial do capacitor, seu valor *default* é 0.

Divisor

Dnome (nome <nó-e>) (nome <nó-s>)

Exemplos:
```
DIV1 PONTO1 30
D2 76 3
```

O *nó-e* representa a entrada do divisor por 2 e *nó-s* a sua saída. A tensão de saída V(nó-s) é igual a V(nó-e) / 2.

Comparador

Knome (nome <nó-e+>) (nome <nó-e->) (nome <nó-s>)

Exemplos:
```
K1 5 30 SAIDA
KOMP2 6 3 4
```

O *nó-e+* representa a entrada não-inversora do comparador, o *nó-e-*, a entrada inversora e *nó-s*, a saída. A tensão de saída V(*nó-s*) é igual Vdd quando V(nó-e+) >= V(nó-e-). E igual a Vss quando V(nó-e+) < V(nó-e-). O nó de saída só pode ser ligado a gates de transistores; em outros casos haverá uma mensagem de erro.

MosFet

Mnome (nome <Dr>) (nome <G>) (nome <Sr>) (nome <Modelo>) [W = real] [L = real]

Exemplos:
```
M1 2 3 4 N
M3 5 CLK1 7 TIPOA W=1 L=2
```

Dr dreno, *G* gate, *Sr* fonte. O nó *G* tem que estar ligado a um sinal de clock ou à saída de um comparador. Os nós *Dr* e *Sr* não podem ser ligados entre si, nem com um sinal de clock nem à saída de um comparador. Se alguma dessas condições ocorrer, um erro de sintaxe será gerado.

L e W são, respectivamente, o comprimento e a largura efetivos do canal. Os valores default são L = 10 um e W = 10 um.

O nome *Modelo*, nome do modelo do transistor, deve ser o mesmo do comando .MODEL. O único modelo existente para os transistores é o NMOS:

NMOS ["(" [INV] [COX = real] [COL= real] ")"]

Nesse modelo tem-se, entre o gate e o source e o gate e o dreno, as capacitâncias de *overlaping* (COL - capacitância do transistor desligado), e de óxido (COX - capacitância adicionada ao transistor ligado). Assim, quando o transistor é ligado, a capacitância de óxido é somada à capacitância de *overlaping*, a primeira é função da área do transistor e é multiplicada por L vezes W, a segunda é função da largura do canal e é multiplicada por W. O comando INV permite que o transistor seja ligado com zero, ou seja acrescenta automaticamente um inversor ao gate do transistor.

Todos esses parâmetros são opcionais. Os valores default são COX = 1 PF/m^2 (capacitância/área), COL = 1 PF/m (capacitância/comprimento) e sem INV.

Os transistores desse modelo sempre desempenham o papel de chaves analógicas controladas por um clock ou um comparador.

Subtrator

Snome (nome <nó-e+>) (nome <nó-e->) (nome <nó-s>)

Exemplos:
```
SUB2 ENT1 ENT2 7
S2 6 3 4
```

O *nó-e+* representa a entrada não-inversora do subtrator, o *nó-e-*, a entrada inversora e *nó-s*, a saída. A tensão de saída V(nó-s) é igual a V(nó-e+) - V(nó-e-).

Fonte de tensão independente

Vnome (nome <nó+>) (nome <nó->) complemFonte.

Exemplos:
```
V1 2 3 DC 3.0V
```

```
VCLK1 5 4 CLK(2 1 0 1 1)
```

Esse elemento é uma fonte de tensão. Corrente positiva flui do *nó+* para o *nó-*.

A seguir, nas descrições das especificações de transiente (complemFonte) possíveis, **tempo** representa o valor do relógio da simulação e **TSTEP** representa o valor de dt, que é a duração de cada passo do relógio da simulação.

DC (real <v1>)

Exemplos:
```
V1 2 3 DC 3.0V
V2 5 6 DC 1
```

A opção DC apresenta sempre o valor de tensão *v1*.

CLK "(" (int <duty_cicle>) (estLógico {estLógico}) ")"

Exemplos:
```
V1 2 3 CLK(2 1 0 1 1)
VCLK1 5 4 CLK (1 1 0 0 1)
```

A opção CLK apresenta um trem de pulsos de VSS a VDD. A duração de cada estado lógico do sinal é de *duty_cicle* vezes o TSTEP. Os estados lógicos permitidos são *estLógico*, podendo chegar até 16. A sequência de estados lógicos é repetida indefinidamente. Esse sinal é sincronisado com o relógio da simulação.

Uma fonte desse tipo só pode ser ligada a gates de transistores.

PULSE "(" (real <v1>) (real <v2>) (real <td>) (real <tr>) (real <tf>) (real <pw>) (real <per>) ")"

Exemplos:
```
V1 2 3 PULSE(-1V 1V 5U .1US .1US 2US 10US)
V3 2 3 PULSE(2 3 5E-3 1E-3 2E-3 1E-3 4E-3)
```

Nome	Significado	Unidade
v1	Tensão Inicial	Volts
v2	Tensão do Pulso	Volts
td	Delay	Seg.
tr	Tempo de Subida	Seg.
tf	Tempo de Descida	Seg.
pw	Largura do Pulso	Seg.
per	Período	Seg.

A opção PULSO faz a tensão começar em *v1* e continuar por *td* segundos. Então, a tensão vai linearmente de *v1* a *v2* durante os próximos *tr* segundos, ficando em *v2* por *pw* segundos. Em seguida ela vai, linearmente, de *v2* de volta a *v1*, durante os próximos *tf* segundos, permanecendo em *v1* por **per - (tr + pw + tf)** segundos. O ciclo é então repetido, exceto pelo delay inicial de *td* segundos.

PWL "(" (real <t1> real <v1>) (real <t2> real <v2>) { (real <tn> real <vn>) } ")"

Exemplos:
```
V1 2 3 PWL(0 -1V 1US 0V 10US 5V)
V5 5 3 PWL(0 2 3E-3 4 5E-3 5)
```

Nome	Significado	Unidade
tn	Tempo da mudança	Volts
vn	Tensão da mudança	Volts

A opção PWL descreve um trem de pulsos por partes. Cada par de valores de tempo-tensão especifica a duração e a tensão de um pedaço do pulso.

SFFM "(" (real <voff>) (real <vampl>) (real <fc>) (real <mod>) (real <fm>) ")"

Exemplos:
```
V1 2 3 SFFM(1.1V 1V 20KHZ 5 1KHZ)
V5 2 3 SFFM(2.0 3 1E3 2 300)
```

Nome	Significado	Unidade
voff	Tensão de offset	Volts
vampl	Tensão alternada	Volts
fc	Frequência Portadora	Hertz
mod	Índice de Modulação	
fm	Frequência do Sinal	Hertz

A opção SFFM (Single-Frequency FM) faz a tensão seguir a seguinte equação:

V = Voff + Vampl * seno((2 * pi * fc * tempo) + mod * seno(2 * pi * fm * tempo)

SIN "(" (real <voff>) (real <vampl>) (real <freq>) (real <td>) (real <df>) (real <phase>) ")"

Exemplos:
```
V1 2 3 SIN(1.1V 1V 20KHZ 1US 0.1 2)
V1 2 3 SIN(2.0 3 1E3 1E-3 0.2 3)
```

Nome	Significado	Unidade
voff	Tensão de offset	Volts
vampl	Tensão alternada	Volts
freq	Frequência	Hertz
td	Delay	Seg.
df	Fator de Decaimento	1/seg.
phase	Fase	Radianos

A opção SIN faz a tensão começar em 0 e continuar assim por *td* segundos. Então a tensão se torna uma onda exponencial senoidal descrita pela equação:

V = Voff + Vampl * exp(-(tempo - td) * df) * seno(2 * pi * freq * (tempo - td) - phase)

Chamada de Subcircuito

Xnome (nome <nó1>) (nome <nó2>) {(nome <nó-n>)} (nome sub-circuito)

Exemplos:
```
XBUFF 2 3 AMPLI
X1 100 103 105 LIMIT
```

No1, *no2* e *nó-n* são os nós comuns da chamada e do circuito. *Subcircuito* é o nome da definição do subcircuito (veja .SUBCKT). Deve haver o mesmo número de nós na chamada e na definição do subcircuito.

Esse comando causa a inserção do subcircuito referenciado no circuito, com os nós dados substituindo os nós comuns na definição. Ele permite a definição de um bloco de circuito, que pode ser usado em vários lugares.

As chamadas de subcircuito podem ser aninhadas. Isso é, pode-se ter uma chamada ao subcircuito A, cuja definição contém uma chamada para o subcircuito B. O aninhamento pode ser até qualquer nível, mas não pode ser circular. Por exemplo, se a definição do subcircuito A contém a chamada ao subcircuito B, então a definição do subcircuito B não pode conter uma chamada ao subcircuito A.

Fim do Circuito

.END

Exemplo: `.END`

O comando .END marca o fim do circuito. Todos os dados e comandos devem vir antes dele. Quando o comando .END é encontrado, a leitura do arquivo de comandos pára e as análises têm início.

Fim da Definição de Subcircuitos

.ENDS

Exemplo: `.ENDS`

O comando .ENDS marca o fim da definição do subcircuito (começado por um comando .SUBCKT).

Modelos

.MODEL (nome Modelo) (nome Tipo) ["(" Definição ")"]

Exemplo:
```
.MODEL NMOS (COX=2PF COL=1.2E-12)
.MODEL NMOS (INV COL=1E-12)
```

O comando .MODEL define um conjunto de parâmetros que podem ser referenciados pelos dispositivos do circuito. *Modelo* é o nome que o dispositivo usa para referenciar um modelo particular.

Tipo é o tipo do dispositivo e o único atualmente disponível, é o NMOS para transistores MOS. A definição desse modelo pode ser encontrada no comando MosFet. Apenas os transistores MOS podem referenciar esse modelo.

Definição traz a lista de parâmetros do modelo entre parênteses. Nenhum, alguns ou todos os parâmetros podem ter seus valores definidos. Valores default são usados para todos os parâmetros não inicializados.

Opções

.OPTIONS [VDD = real] [VSS = real] [VT = real] [L = real] [W = real] [LIMLOOP = int] [BEGINCMP = estLógico] [TEST = (NONE | VT)]

Exemplo:
```
.OPTIONS VDD=5.3V VT=1.4V LIMLOOP=10
.OPTIONS VDD=6 TEST=VT BEGINCMP=1
```

O comando .OPTIONS é usado para inicializar todas as opções, limites e parâmetros de controle para a análise efetuada.

As opções são simplesmente listadas em qualquer ordem. A tabela seguinte lista as opções disponíveis.

Opção	Significado	Unidade	Default
VDD	Tensão da fonte positiva	Volts	5 V
VSS	Tensão da fonte negativa	Volts	-5 V
VT	Tensão Vt dos transistores MOS	Volts	1 V
L	Valor default para o comprimento dos transistores	m	10 um
W	Valor default para a largura dos transistores	m	10 um
LIMLOOP	Max. número de ciclos de operação dos comparadores num ciclo de relógio		10
BEGINCMP	Estado inicial dos comparadores		1
TEST	NONE = Sem teste VT = Testa se a tensão nos nós ultrapassa VDD + Vt ou é menor que VSS		Testa se V nó > VDD ou < VSS

Probe

.PROBE { V"(" (nome <no>) ")"}.

Exemplo:
```
.PROBE V(SAIDA) V(8) V(7)
.PROBE
```

O comando .PROBE transcreve os resultados da análise para o arquivo .DAT, para o uso do programa OSC. A forma do comando, sem especificação de variáveis escreve o valor de tensão de todos os nós do circuito no arquivo .DAT. A forma do comando, com uma lista de nós especificados, transcreve apenas estes para o arquivo .DAT. O número de nós dessa lista é limitado apenas pelo número total de nós do circuito.

Esse comando pode reduzir o tamanho do arquivo .DAT, diminuindo o tempo necessário para escrita e permite que apenas os nós de interesse sejam gravados. Isso é importante, pois o programa OSC tem uma capacidade limitada ao tamanho da memória do computador, para armazenar os dados dos nós.

Definição de Subcircuitos

.SUBCKT (nome <sub-circuito>) (nome <nó1>) (nome <nó2>) { (nome <nó-n>) }

Exemplo:
```
.SUBCKT AMPLIF ENT1 ENT2 VDD VSS SAIDA
.SUBCKT AND 3 4 5
```

O comando .SUBCKT começa a definição de um subcircuito. Essa definição é terminada com um comando .ENDS. Todos os comandos entre .SUBCKT e .ENDS são incluídos na definição. Sempre que o circuito é chamado por um comando X..., todos os comandos na definição substituem o comando de chamada.

Sub-circuito é o nome do subcircuito e é usado, por um comando X..., para referenciar o subcircuito.

Nó1 a *nó-n* é a lista de nós. Deve existir o mesmo número de nós nos comandos de chamada de subcircuito e na definição. Quando o subcircuito é chamado, os nós verdadeiros (aqueles do comando de chamada) substituem os nós do argumento do comando de definição de subcircuito.

As definições dos subcircuitos não podem ser aninhadas, e devem conter apenas comandos de dispositivos (comandos que não começam com ".").

Parâmetros de simulação.

.TRAN (real <dt>) (real <total T>) (int <step>)

Exemplo:
```
.TRAN 1E-3 100E-3 1
.TRAN 2US  160US  2
```

O comando .TRAN define parâmetros obrigatórios da simulação. *Dt* é o período de cada ciclo do relógio de simulação, *total T* é o período total da simulação e *step* é a frequência com que o simulador escreve os valores dos nós no arquivo de saída. Assim, o comando .TRAN 1US 100US 2 significa que cada ciclo do relógio tem 1US de duração, que o tempo total de simulação tem 100 US e que de 2 em 2 pulsos do relógio serão armazenados os valores dos nós. Dessa forma, haverá

51 pontos disponíveis no arquivo de saída, 100US / 10US + o estado inicial.

Esse comando deve aparecer apenas uma vez, em qualquer parte do arquivo do circuito.

Comentários

*(qualquer texto)

Exemplo: `* Inicio de Circuito de Amplificacao`

Todas as linhas começadas por * são comentários. Elas são ignoradas pelo programa. O uso de comentários é fortemente recomendado.

I.4 Exemplos

Circuito amplificador, nesse circuito a amplificação é dada pela razão entre as frequências de clk1 e clk2:

```
CIRCUITO DO AMPLIFICADOR DE GANHO 2
.TRAN  5E-6   500E-6 0
V1  CLK1 0 CLK (1 1 0)
V3  CLK2 0 CLK (2 1 0)
.SUBCKT ATR 1 4 CLK
M1  1  CLK  2  N
M2  3  CLK  4  N
M3  1  CLK  3  P
M4  2  CLK  4  P
C1 2 0 100PF
C2 3 0 100PF
.ENDS
X1  1  4 CLK2 ATR
X2  6 16 CLK2 ATR
X3 10 17 CLK2 ATR
X4  9 15 CLK2 ATR
M1  9  CLK1 10  N
M2  6  CLK1  5  N
M1  9  CLK1  6  P
M2 10  CLK1  5  P
```

```
ADDER1 4 5 9
VDC1   16   0 DC 0
VDC2   17   0 DC 0
VDC3 1 0 SIN( 0 0.5 2000 0 0 0)
.MODEL N NMOS (     COX=1PF COL=0.5PF)
.MODEL P NMOS (INV COX=1PF COL=0.5PF)
.PROBE V(15) V(1) V(5) V(9) V(4) V(10) V(6) V(2)
.OPTION L=1 W=1
.END
```

Circuito multiplicador: esse circuito multiplica 2 sinais de tensão (V7 e V8):

```
Circuito multiplicador simples
.SUBCKT ATR 1 4 CLK
X1 1 2 3 4 CLK CHAVE
C1 2 0 100PF
C2 3 0 100PF
.ENDS
.SUBCKT CHAVE 1 2 3 4 CLK
M1   1   CLK   2   N
M2   3   CLK   4   N
M3   1   CLK   3   P
M4   2   CLK   4   P
.ENDS
.TRAN   2.22US   2.22MS 0
V1 CLK1 0 CLK ( 1 1 0)
V2 CLK2 0 CLK (50 1 0)
KOMP1   5   7 CMP1
X1   2   5   CLK2 ATR
X2   12 9   CLK2 ATR
X3   12 10 CLK2 ATR
X4   17 18 CLK2 ATR
X5   22 27 CMP1 ATR
X6   62 23 CLK2 ATR
X7   62 24 CLK2 ATR
X8   40 34 CLK2 ATR
X9   7   10 9   11 CLK1 CHAVE
X10 22 24 23 21 CLK1 CHAVE
ADD1   6 11   7
ADD2 21 18 22
```

```
ADD2 21 18 22
BUF1 27 40
V3 12 0 DC 0
V4  6 0 DC 0.01
V5 62 0 DC 0
V7 17 0 SIN (0.01V  0.01V 450 0 0 0)
V8  2 0 SIN (0.26V 0.24V 450 0 0 0)
.PROBE V(7) V(5) V(22) V(27) V(34) V(18) V(17)
.MODEL N NMOS (    COX=10MF COL=1UF)
.MODEL P NMOS (INV COX=10MF COL=1UF)
.OPTION VT=1.2
.END
```

ANEXO 2

GRAFO DA LINGUAGEM

O grafo da linguagem seguida pelo GERA é definido a seguir:

É usda a notação proposta por Nicklaus Wirth [22] [23] : as regras usam o sinal de igual = para relacionar identificadores a expressões, barra vertical | para *ou*, e aspas " " circundando caracteres; as chaves { } indicam repetição em qualquer número de vezes incluindo zero; colchetes [ ] indicando opcionalidade (i.e. zero ou 1 vez); parenteses ( ) são usados para agrupamento; as regras são terminadas por pontos. Foram acrescentados a essa notação os sinais de < >, que significam comentários à linguagem.

arquivo =	{comentario term} { {branco} [comando] term } comFim {sep tudo}.
comentario =	"*" tudo.
comando =	circuitoCom |subCircuito | comModelo | comProbe | comTran | comOptions.
subCircuito =	comSubCirc term circuitoCom term comFimSub.

circuitoCom = comCapacitor | comTransist | comFonte | comSomador | comSubtrator | comBuffer | comDivisor | comComp | comCallCirc.

comSubCirc = ".SUBCKT" sep nome <sub-circuito> sep nome <nó1> sep nome <nó2> { sep nome <nó> }.

comFimSub = ".ENDS".

comModelo = ".MODEL" sep nome sep "(" modNmos ")".

modNmos = "NMOS" ["(" [INV] ["COX" sep "=" real] ["COL" "=" real] ")"].

comProbe = ".PROBE" { sep "V" sep "(" nome <nó> ")"}.

comTran = ".TRAN" real <dt> real <total T> int <step>.

comOptions = ".OPTIONS" sep ("VDD" sep "=" real | "VSS" sep "=" real | "VT" sep "=" real | "L" sep "=" real | "W" sep "=" real | "LIMLOOP" sep "=" int | "BEGINCMP" sep "=" estLógico | "TEST" sep "=" ("NONE" | "VT").

comFim = ".END".

comCapacitor = "C"nome sep nome <nó+> sep nome <nó-> real <capac> ["V" sep "=" real].

comTransist = "M"nome sep nome <Dr> sep nome <G> sep nome <Sr> sep nome <Modelo> ["W" sep "=" sep real] ["L" sep "=" real].

comSomador = "A"nome sep nome <nó-e1> sep nome <nó-e2> sep nome <nó-s>.

comSubtrator = "S"nome sep nome <nó-e+> sep nome <nó-e-> sep nome <nó-s>.

comBuffer = "B"nome sep nome <nó-e> sep nome <nó-s>.

comDivisor = "D"nome sep nome <nó-e> sep nome <nó-s>.

comComp = "K"nome sep nome <nó-e+> sep nome <nó-e-> nome <nó-s>.

comCallCirc = "X"nome sep nome <nó> sep nome <nó> {sep nome <nó>} nome <sub_circuito>.

comFonte = "V"nome sep nome <nó+> sep nome <nó-> sep (complDC | complCLK | complSIN | complSFFM | complPULSE | complPWL).

complDC = "DC" real <v1>.

complCLK = "CLK" sep "(" int <duty_cicle> estLógico {estLógico} ")".

complSIN = "SIN" sep "(" real <voff> real <vampl> real <freq> real <td> real <df> real <phase> ")".

complSFFM = "SFFM" sep "(" real <voff> real <vampl> real <fc> real <mod> real <fm> ")".

complPULSE = "PULSE" sep "(" real <v1> real <v2> real <td> real <tf> real <pw> real <pw> real <per> ")".

complPWL = "PWL" sep "(" real <t1> real <v1> real <t2> real <v2> { real <tn> real <vn> } ")".

term = Return.

nome = letra {letra}.

real = {sep} {"-"} {digito} { "." digito {digito}} { "E" int } {mult} {unidade}.

int = {sep} {"-"} digito {digito} <maximo 12>.

digito = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9".

mult = "P"|"N"|"U"|"M"|" "|"K"|"MEG"|"G"|"T".

unidade = "V"|"F"|"S"|"HZ".

estLógico = "0"|"1".

letra = "_"|"-"|"."|"A"|...|"Z"|"0"|...|"9".

sep = branco {branco}.

branco = " "|"\r"|"\f"|"\b"|"\t"|"a"|...|"z".

tudo = Qualquer símbolo menos term e fim de arquivo.

ANEXO 3

ESTRUTURAS DE DADOS DO SIMULA

A definição das estruturas de dados partilhadas pelos programas SIMULA e GERA podem ser encontradas no arquivo VARLIB.C.

A lista da definição dos tipos de dados segue:

```
typedef char tipo_men_erro[80];
typedef char tipo_palavra[MAX_PALAVRA + 1];
typedef char tipo_arquivo[64];
```

Estrutura do banco de dados- itens gerais:

```
typedef unsigned int tipo_basico;

typedef struct {
   tipo_basico no;
   float       carga;
   float       capacitancia;
} item_cap;
typedef item_cap *point_cap;
```

```
typedef struct {
   tipo_basico n1;
   tipo_basico ctr;
   tipo_basico n2;
   char        inv;
   float       cap_aberta; /* Capacit. por lado */
   float       cap_fechada; /* Capacit. por lado*/
   char        estado;
   char        tran;
   char        status1;
   char        status2;
   float       carga1;
   float       carga2;
} item_switch;
typedef item_switch *point_switch;

typedef struct {
   tipo_basico num;
   tipo_palavra nome;
   int         duty_cicle;
   int         clock;
   int         ind_estado;
   char        num_estado;
   char        lista_estado[MAX_ESTADOS_CLOCK];
   tipo_basico num_switch;
   tipo_basico lista_switch;
} item_clock;
typedef item_clock *point_clock;

typedef struct {
   char        tipo;
   tipo_basico num_entrada;
   tipo_basico entrada[4];
} item_device;
typedef item_device *point_device;

typedef struct {
   tipo_basico  num;
   tipo_palavra nome;
   tipo_basico  e;
   tipo_basico  ein;
   char         estado;
```

```
   char          status;
   tipo_basico   num_switch;
   tipo_basico   lista_switch;
} item_cmp;
typedef item_cmp *point_cmp;

typedef struct {
   char         tipo;
   tipo_basico point_fonte_ind;
} item_fonte;
typedef item_fonte *point_fonte;

typedef struct str_no {
   char           status;
   tipo_basico    num;
   tipo_palavra   nome;
   char           tipo_fonte;
   tipo_basico    point_fonte;
   tipo_basico    num_switch;
   tipo_basico    lista_switch;
   struct str_no *cdr;
} item_no;
typedef struct str_no *point_no;
```

Estrutura do banco de dados - Itens especificos a cada device

```
typedef struct {
   float tensao;
} item_fonte_dc;
typedef item_fonte_dc *point_fonte_dc;

typedef struct {
   float voff;
   float vampl;
   float freq;
   float td;
   float df;
   float phase;
} item_senoide;
typedef item_senoide *point_senoide;
```

```
typedef struct {
   float voff;
   float vampl;
   float fc;
   float mod;
   float fm;
} item_sffm;
typedef item_sffm *point_sffm;

typedef struct {
   float v1;
   float v2;
   float tdelay;
   float trise;
   float tstand;
   float tfall;
   float ttotal;
} item_pulso;
typedef item_pulso *point_pulso;

typedef struct {
   char  num_estado;
   float lista_tempo[MAX_ESTADOS_CLOCK];
   float lista_tensao[MAX_ESTADOS_CLOCK];
} item_palavra;
typedef item_palavra *point_palavra;

typedef tipo_basico *point_geral;
```

Estruturas suplementares

```
typedef struct {
   char         flag;
   float        dt;
   float        total;
   tipo_basico  step;
   float        vcc;
   float        vee;
   float        vt;
   tipo_basico  max_cmp_loops;
   unsigned int begin_cmp;
   char         test;
```

```
    float       coef;
    float       l;
    float       w;
} item_tran;
```

ANEXO 4

CIRCUITOS USADOS NOS TESTES

Listagens dos circuitos simulados no QUANTICO:

```
Circuito do Amplificador de ganho 2
.TRAN 5E-6 700E-6 0
V1   CLK1 0 CLK (1 1 0 )
V3   CLK2 0 CLK (1 0 1 1 0 )

.SUBCKT ATR 1 4 CLK
M1   1   CLK   2   N
M2   3   CLK   4   N
M3   1   CLK   3   P
M4   2   CLK   4   P
C1 2 0 100PF
C2 3 0 100PF
.ENDS

X1   1   4 CLK2 ATR
X2   6  16 CLK2 ATR
X3  10  17 CLK2 ATR
X4   9  15 CLK2 ATR
M1   9   CLK1 10   N
M2   6   CLK1  5   N
M1   9   CLK1  6   P
M2  10   CLK1  5   P
ADDER1 4 5 9
```

```
VDC3 1 0 PULSE(-0.5 0.5 0 100E-6 100E-6 200E-6 600E-6)
VDC1  16  0 DC 0
VDC2  17  0 DC 0

* Modelos usados para chaves analógicas ideais
.MODEL N NMOS
.MODEL P NMOS (INV)

* Modelos usados para chaves analógicas reais
.MODEL N NMOS (    COX=2UF )
.MODEL P NMOS (INV COX=2UF )
.PROBE V(15) V(1) V(5) V(9) V(4) V(10)
.END



Circuito multiplicador
.SUBCKT ATR 1 4 CLK
M1  1  CLK  2  N
M2  3  CLK  4  N
M3  1  CLK  3  P
M4  2  CLK  4  P
C1 2 0 100PF
C2 3 0 100PF
.ENDS

.SUBCKT CHAVE 1 2 3 4 CLK
M1  1  CLK  2  N
M2  3  CLK  4  N
M3  1  CLK  3  P
M4  2  CLK  4  P
.ENDS

.TRAN  1.1US  3.8MS 0
V1 CLK1 0 CLK ( 1 1 0)
V2 CLK2 0 CLK (50 1 0)
KOMP1  5  7 CMP1
X1  2   5  CLK2 ATR
X2  12  9  CLK2 ATR
X3  12  10 CLK2 ATR
X4  17  18 CLK2 ATR
X5  22  27 CMP1 ATR
X6  62  23 CLK2 ATR
X7  62  24 CLK2 ATR
X8  40  34 CLK2 ATR
X9  7  10 9  11 CLK1 CHAVE
X10 22 24 23 21 CLK1 CHAVE
ADD1  6 11  7
ADD2 21 18 22
BUF1 27 40
V3 12 0 DC 0
```

```
V4  6 0 DC 0.01
V5 62 0 DC 0
V7 17 0 PULSE (0    0.02 0 1.1111MS 1.1111MS 1PS 2.2222MS)
V8  2 0 PULSE (0.02 0.5  0 1.1111MS 1.1111MS 1PS 2.2222MS)
.PROBE V(2) V(17) V(34) V(CMP1) V(6) V(22) V(7) V(5) V(27)
.MODEL N NMOS
.MODEL P NMOS (INV)
.END
```

Listagens dos circuitos simulados no SWITCAP:

```
TITLE: Amplificador de ganho 2, ideal

options;
   width132;
   report;
end;

timing;
   period 10e-6;
   clock cka 1 (0 1/2);
   clock ckb 2 (0 1);
end;

subckt (1 3) delay (k:clock);
   s1 (1 2)  clock;
   s2 (2 3) #clock;
   s3 (4 3)  clock;
   s4 (1 4) #clock;
   c1 (2 0) 100e-12;
   c2 (4 0) 100e-12;
end;

subckt (1 3) debuf (k:clock);
   s1 (1 2)  clock;
   s2 (2 3) #clock;
   s3 (4 3)  clock;
   s4 (1 4) #clock;
   c1 (2 0) 100e-12;
   c2 (4 0) 100e-12;
end;

subckt (1 2 3 4) tap (k:clock);
   s1 (1 2)  clock;
   s2 (2 3) #clock;
   s3 (4 3)  clock;
   s4 (1 4) #clock;
end;
```

```
subckt (1 2 3) adder (p:gain);
   einv (4 0 0 2) 1.0;
   esum (3 0 1 4) gain;
end;

circuit;
   xa (1  3) debuf (ckb);
   xb (5  7) debuf (ckb);
   xc (9  0) delay (ckb);
   xd (11 0) delay (ckb);
   xe (5 9 10 11) tap (cka);
   xsum (3 10 5) adder (1.0);
   vin (1 0);
end;

analyze tran;
   time 0+ 70 1/2;
   set vin pulse -0.5 0.5 5e-6 100e-6 100e-6 200e-6 600e-6;
   plot v(7) v(1);
   print v(7) v(1) v(3) v(10) v(5)
end;

end;



TITLE: Amplificador de ganho 2, com clock feedthrogh

options;
   width132;
   report;
end;

timing;
   period 10e-6;
   clock cka 1 (0 1/2);
   clock ckb 2 (0 1);
end;

subckt (s d g b) ch (k:clock);
   s1  (s  d)  clock;
   ssg (g sg)  clock;
   ss0 (b sg) #clock;
   sdg (g dg)  clock;
   sd0 (b dg) #clock;
   cgs (s sg)  1e-12;
   cgd (d dg)  1e-12;
   csb (s  b)  1e-12;
   cdb (d  b)  1e-12;
end;
```

```
subckt (1 3 g b) delay (k:clock);
   x1 (1 2 g b) ch ( clock);
   x2 (2 3 g b) ch (#clock);
   x3 (4 3 g b) ch ( clock);
   x4 (1 4 g b) ch (#clock);
   c1 (2 0) 100e-12;
   c2 (4 0) 100e-12;
end;

subckt (1 3 g b) debuf (k:clock);
   x1 (1 2 g b) ch ( clock);
   x2 (2 3 g b) ch (#clock);
   x3 (4 3 g b) ch ( clock);
   x4 (1 4 g b) ch (#clock);
   c1 (2 0) 100e-12;
   c2 (4 0) 100e-12;
end;

subckt (1 2 3 4 g b) tap (k:clock);
   x1 (1 2 g b) ch ( clock);
   x2 (2 3 g b) ch (#clock);
   x3 (4 3 g b) ch ( clock);
   x4 (1 4 g b) ch (#clock);
end;

subckt (1 2 3) adder (p:gain);
   einv (4 0 0 2) 1.0;
   esum (3 0 1 4) gain;
end;

circuit;
   xa (1  3 g b) debuf (ckb);
   xb (5  7 g b) debuf (ckb);
   xc (9  0 g b) delay (ckb);
   xd (11 0 g b) delay (ckb);
   xe (5 9 10 11 g b) tap (cka);
   xsum (3 10 5) adder (1.0);
   vin (1 0);
   vb  (b 0);
   vg  (g 0);
end;

analyze tran;
   time 0+ 70 1/2;
   set vin pulse -0.5 0.5 5e-6 100e-6 100e-6 200e-6 600e-6;
   set vg dc 5;
   set vb dc -5;
   plot v(7) v(1);
   print v(7) v(1) v(3) v(10) v(5)
end;
end;
```

REFERÊNCIAS

[1] Y. Tsividis, "Principles of Operation and Analysis of Switched-Capacitor Circuits," *Proc. IEEE,* vol. 71, no. 8, pp. 926-940, Aug. 1983.

[2] Y. Tsividis, "Analysis of Switched Capacitive Networks," *IEEE Trans. Circuits Syst.,* vol. CAS-26, no. 11, pp. 935-946, Nov. 1979.

[3] M. Liou, Y. L. Kuo, and C. F. Lee, "Computer-Aided Analysis of SC Circuits", *Proceedings of the IEEE,* vol 71, pp 987-1005, Aug. 83.

[4] D. MacQuigg, "Residual Charge on a Switched Capacitor," *IEEE J. Solid-State Circuits*, vol. SC-18, pp 811-813, Dec. 1983.

[5] B. J. Sheu and C. Hu, "Switched-Induced Error Voltage on a Switched Capacitor," *IEEE J. Solid-State Circuits,* vol. SC-19, pp 519-525, Aug. 1984.

[6] W. B. Wilson, H. Z. Massoud, E. J. Swanson, R. George Jr, and R. B. Fair, "Measurement and Modeling of Charge Feedthrough in n-Channel MOS Analog Switches," *IEEE J. Solid-State Circuits,* vol. SC-20, pp 1206-1213, Dec. 1985.

[7] P. Van Peterghem and W Sansen, "Single versus complementary switches: a discurssion of clock feedthrough in S. C. circuits", in *ESSCIRC'86, Delft,* 1986, B4.13-B4.15.

[8] B. J. Sheu, J. Shieh, and M. Patil, "Modeling Charge Injection in MOS Analog Switches," *IEEE Trans. Circuits Syst.,* vol. CAS-34, pp 214-216, Feb. 1987.

[9] L.W. Nagel, *SPICE2: A COMPUTER PROGRAM TO SIMULATE SEMICONDUCTOR CIRCUITS,* Memorandum No. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, 09/05/1975.

[10] *User's Guide for Switcap - Version 5,* Columbia University, New York NY, August 87.

[11] A. C. R. Silva, *Contribuição à Minimização e Simulação de Circuitos Lógicos,* Tese de Mestrado, FEE - UNICAMP, Novembro 1989.

[12] R.A. Saleh, J.E. Kleckner, and A.R. Newton, *SPLICE Version 1.7 User's Guide,* Dep. of Electrical Engineering and Computer Science, University of California, Berkeley CA.

[13] M. M. Tarik and R. T. Yeh, "Rapid Prototyping in Software Development," *Computer,* Vol. 22, No. 5, May 1989, pp 9-10

[14] K. Christian, *Sistema Operacional UNIX,* Editora Campus, Rio de Janeiro RJ, 1985, pp 238-250.

[15] W.W. Setzer e J.H. de Melo, *A Construção de um Compilador,* Editora Campus, Rio de Janeiro RJ, 1988.

[16] E. Rich, *Inteligência Artificial,* McGraw-Hill, São Paulo SP, 1988, Capítulo 3.

[17] B.H. McCormick, T.A. De Fant, and M.D. Brown, eds., "Visualization in Scientific Computing," *Computer Graphics*, Vol.21, No. 6, Nov. 1987.

[18] T.A. DeFanti, M.D. Brown, and B.H. McCornick, "Visualization Expanding Scientific and Engineering Research Opportunities," *Computer*, Vol.22, No. 8, Aug. 1989, pp 12-25.

[19] D.A. Moreira, *EDCHIP - Manual do Sistema Dinâmico de Edição de Chips, Versão 3.X*, UNICAMP, Campinas SP, 11/11/89.

[20] D.A. Moreira e M.M. Fratel Jr., "Edchip - Editor Dinâmico de Chips," *Anais do SBCCI 88, III Simpósio Brasileiro de Concepção de Circuitos Integrados*, Gramado RS, Abril 1988, pp 13-22.

[21] *PSpice*, MicroSim Corp., Tustin CA, 1984.

[22] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Massachusetts, 1979, pp 91-144.

[23] N. Wirth, "What Can We Do about the Unnecessary Diversity of Notations for Syntatic Definitions?," *Communications of the ACM*, November 1977.

[24] *EDIF Electronic Design Interchange Format, Version 2 0 0*, EDIF Steering Comitee, Electronics Industries Association, March 20, 1987.