MARCELO LOYO PRADO

UMA FERRAMENTA COMPUTACIONAL PARA A GERAÇÃO AUTOMÁTICA DE SÍMBOLOS E ESQUEMÁTICOS DE COMPONENTES ELETRÔNICOS

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Manoel Luís de Aguiar

São Carlos
2010

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

P896f

Prado, Marcelo Loyo
Uma ferramenta computacional para a geração automática de símbolos e esquemáticos de componentes eletrônicos / Marcelo Loyo Prado ; orientador Manoel Luís de Aguiar. -- São Carlos, 2010.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2010.

1. Circuitos integrados. 2. Modelo de componente. 3. Gerador de esquemáticos. 4. Gerador de símbolos. 5. Linguagem SKILL. 6. Ambiente de desenvolvimento cadence I. Título.

AGRADECIMENTOS

Aos professores Manoel Luís de Aguiar e José Carlos de Melo Vieira, por todos os conselhos e apoio à minha ida à Alemanha.

A Uwe Knöchel e Pietro Brenner, por toda a orientação e confiança em mim depositada ao me darem a oportunidade de trabalhar num projeto tão importante.

A todos os professores e funcionários da Universidade de São Paulo, que de algum modo colaboraram para me transformar num engenheiro.

Ao Instituto Fraunhofer, por possiblitar que eu aprendesse tanto durante minha primeira experiência de trabalho.

Aos meus amigos Jáder, Daniel e Rodrigo, por fazerem meus anos de graduação serem tão divertidos.

A todos os amigos que fiz em Dresden, por fazerem deste último ano o melhor da minha vida.

Ao meu pai e irmã, por toda a ajuda, sem a qual minha ida à Alemanha jamais teria sido possível.

A minha avó, por me inspirar a ser sempre o melhor que eu posso ser.

A minha mãe, cujo sacrifício me deu a determinação necessária para realizar meus sonhos.

“Tantos de nossos sonhos no começo parecem impossíveis,
então passam a parecer improváveis,
e então, quando reunimos a determinação,
eles logo tornam-se inevitáveis.”
Christopher Reeve

RESUMO

PRADO, M. L. **Uma ferramenta computacional para a geração automática de símbolos e esquemáticos de componentes eletrônicos.** Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010, 73p.

A etapa de projeto de um circuito eletrônico é de fundamental importância no desenvolvimento de qualquer sistema. Dessa maneira, é essencial que se possa sistematizar e automatizar vários dos estágios envolvidos no projeto de circuitos, a fim de se evitar desperdício de recursos e também erros, que poderiam acarretar em atrasos no projeto e até perdas no desempenho do produto final. Sendo assim, o objetivo deste trabalho é o desenvolvimento de uma ferramenta computacional capaz de gerar automaticamente símbolos e esquemáticos de componentes eletrônicos através da importação de arquivos de modelos dos mesmos. O trabalho foi desenvolvido no Instituto Fraunhofer para Circuitos Integrados, localizado em Dresden, Alemanha, e é parte integrante de um grande projeto desenvolvido em conjunto entre várias empresas e institutos de pesquisa. Seus produtos finais serão destinados a empresas públicas e privadas do setor de desenvolvimento de circuitos integrados das mais diversas áreas da indústria de eletrônicos. A ferramenta foi implementada em linguagem SKILL, sendo todo o trabalho realizado no Ambiente de Desenvolvimento Cadence. A validação da ferramenta foi realizada através de testes com modelos de componentes reais fornecidos por empresas participantes do projeto, sendo que os resultados vieram a comprovar seu grande potencial de aplicação no ambiente industrial.

Palavras-chaves: circuito integrado; modelo de componente; gerador de símbolos; gerador de esquemáticos; linguagem SKILL; Ambiente de Desenvolvimento Cadence.

ABSTRACT

PRADO, M. L. **A computational tool for the automatic generation of symbols and schematics from electronic components**. Course Conclusion Work – Engineering School of São Carlos, University of São Paulo, São Carlos, 2010, 73p.

The project phase of electronic circuits is of fundamental importance in the development of any system. In this manner, is essencial to systematize and automatize many of the stages involved in the project of circuits with the objective of avoiding waste of resources and also errors, which could lead to project delays and even losses in the final product performance. Therefore, the objective of this work is to develop a computational tool able to automatically generate symbols and schematics from electronic components through the importation of their model files. The work was developed in the Fraunhofer Institute for Integrated Circuits, located in Dresden, Germany, and is an integrating part of a bigger project developed between many companies and research institutes. Its final products are destined to public and private companies from the integrated circuits development sector from several areas of the Electronics Industry. The tool was implemented in SKILL language, with the whole work being developed in the Cadence Design Environment. The tool validation was performed through tests with models from real components, supplied by project partner companies, and the results came to prove its great potential for the application in the industrial environment.

Keywords: integrated circuit; component model; symbol generator; schematics generator; SKILL language; Cadence Design Environment.

SUMÁRIO

LISTA DE FIGURAS xiii
LISTA DE TABELAS xv
LISTA DE ABREVIATURAS E SIGLAS xvii
1. INTRODUÇÃO 1
2. O AMBIENTE DE DESENVOLVIMENTO CADENCE 4
2.1 – A Linguagem SKILL 4
2.2 – CIW - Janela de Interface Comum (*Common Interface Window*) 5
2.3 – O Sistema de Bibliotecas do Cadence 6
2.4 – O Compositor de Esquemáticos 8
2.5 – O Compositor de Símbolos 10
3. A FERRAMENTA IECM 13
3.1 – Criação da GUI e Funções Principais da IECM 15
3.1.1 - GUI 15
3.1.2 – *Browser* de importação de modelos 19
3.1.3 – Chamada de Módulos 21
3.2 – Leitura e *Parsing* de Modelos 22
3.2.1 – Leitura e Análise de Erros de *Component Specifications* 22
3.2.2 - Análise de Erros de Modelos Spectre 30
3.2.3 – Análise de Erros de Modelos Touchstone 32
3.3 – A Geração das Vistas do Componente 34
3.3.1 – Geração do Símbolo 34
3.3.2 – Geração do Esquemático de Pinos 43
4. RESULTADOS DOS TESTES COM A IECM 45
4.1 – A Geração do Símbolo. 48
4.2 – A Geração do Esquemático de Pinos. 51
5. CONCLUSÕES 54
REFERÊNCIAS BIBLIOGRÁFICAS 56
Apêndice A – Lista dos arquivos e funções da ferramenta IECM 57
Apêndice B – Arquivo *ifxIecmBrowser.il* 58
Apêndice C – Arquivo *ifxIecmDataEntryFormatSpecification.il* 63
Apêndice D – Carta do gerente do projeto no Instituto Fraunhofer 68

LISTA DE FIGURAS

Figura 1 – Janela de Interface Comum (CIW). 5
Figura 2 - Estrutura hierárquica das pastas das bibliotecas, células e vistas. 7
Figura 3 - *Browser* para gerenciamento das bibliotecas, células e vistas do ambiente *Cadence*. 7
Figura 4 - Janela do Compositor de Esquemáticos. 8
Figura 5 - Janela do Compositor de Símbolos. 10
Figura 6 - Exemplo de um esquemático hierárquico utilizando o símbolo do amplificador operacional 11
Figura 7 - Diagrama de fluxo de execução da IECM. 14
Figura 8 - GUI da Ferramenta IECM. 15
Figura 9 - Seção *1) Read and parse -Component Specification- file*. 16
Figura 10 - Seção *2) Define target lib and target cell for model import*. 17
Figura 11 - Seção *3) Specify Spectre Subckt Model View to be imported*. 17
Figura 12 - Seção *4) Specify S-Parameter Model View to be imported*. 18
Figura 13 - Seção *5) Specify Titan Subckt Model View to be imported*. 18
Figura 14 - Seção de seleção de vistas a serem geradas. 18
Figura 15 - *Browser* para auxiliar na importação de modelos 20
Figura 16 - Exemplo de um arquivo de *Component Specification* 23
Figura 17 - Mensagem enviada ao usuário caso não hajam erros na *Component Specification*. 28
Figura 18 - Mensagem enviada ao usuário caso hajam erros na *Component Specification*. 28
Figura 19 - *Log File* gerado pelo *Parser* contendo os resultados da análise da *Component Specification*. 29
Figura 20 - *Errors File* gerado pelo *Parser* contendo o relatório de erros da *Component Specification*. 30
Figura 21 - Ilustração dos estágios da criação do símbolo do componente. 34
Figura 22 - Ilustração da posição dos elementos do símbolo. 35
Figura 23 - Exemplo de um símbolo gerado pela ferramenta IECM. 42
Figura 24 – Exemplo de um esquemático de pinos gerado pela ferramenta IECM. 44
Figura 25 - *Component Specification* usada nos testes. 46
Figura 26 - Símbolo do componente *Test_Component* estruturado por *SigClass*. 49
Figura 27 - Símbolo do componente *Test_Component* estruturado por *PinType*. 50

Figura 28 - Símbolo do componente *Test_Component* estruturado por *SymPinPos*.....51
Figura 29 - Esquemático de pinos do componente *Test_Component* estruturado por *SigClass*. 51
Figura 30 - Esquemático de pinos do componente *Test_Component* estruturado por *PinType*. 53
Figura 31 - Esquemático de pinos do componente *Test_Component* estruturado por *SymPinPos*. 53

LISTA DE TABELAS

Tabela 1 – Colunas de propriedades que devem estar sempre presentes na *Component Specification* 24

Tabela 2 – Elementos usados para compor o símbolo e funções responsáveis por suas criações 38

Tabela 3 – Campos da variável *ifxIecmGlobalVar*, usada para definir as dimensões do símbolo. 39

Tabela 4 – Figuras criadas pelo programa para cada tipo de elemento do símbolo 39

LISTA DE ABREVIATURAS E SIGLAS

EDA – *Electronic Design Automation*

CAD – *Computer-Aided Design*

CIW – *Common Interface Window*

IECM – *Import of External Component Models*

GUI – *Graphical User Interface*

1. INTRODUÇÃO

A etapa de projeto de circuitos eletrônicos é de grande importância no desenvolvimento de qualquer sistema, seja ele simples ou complexo, pois é nessa etapa que será definida a configuração final do sistema que se deseja construir e consequemente o desempenho e custo que ele terá. Muitos recursos, como tempo, mão de obra e dinheiro, são gastos nos vários estágios da fase de projeto, portanto é essencial que se possa sistematizar e automatizar vários desses estágios, a fim de se evitar desperdício de recursos e também erros, que poderiam acarretar em atrasos no projeto e até mesmo perdas no desempenho do produto final.

Entre os diversos estágios da etapa de projeto do sistema, destaca-se a elaboração dos esquemáticos do circuito, com a correspondente escolha dos melhores componentes a serem utilizados e a disposição espacial dos vários elementos e suas interconexões. Na elaboração desses esquemáticos é importante ter-se representações de cada parte componente do sistema, para que se possa usar uma abordagem modular ao projeto, representando o sistema completo através de vários níveis hierárquicos, desde níveis elevados, que possuam módulos complexos como microprocessadores, até níveis inferiores, onde elementos simples como resistores e capacitores possam ser representados. Para essa abordagem modular, é necessário que se desenvolva os vários tipos de representações de cada um dos componentes, também chamadas de "vistas" dos componentes, entre elas destacando-se o símbolo do componente e seu esquemático propriamente dito. Se por um lado o símbolo do componente permite uma vizualização simplificada do mesmo, mostrando apenas características principais como seu nome e pinos externos para que se possa conectá-lo a outros componentes, por outro o esquemático permite ao projetista ver toda a estrutura interna do componente, com todos os componentes menores que o compõe e as interconexões entre eles.

Tendo-se em vista esses fatos, a implementação de programas de computador que possam automatizar essa criação de símbolos e esquemáticos de componentes, usando como entrada modelos que descrevam a estrutura interna do componente, é de grande valia para o aprimoramento dos métodos atuais de desenvolvimento de circuitos eletrônicos em qualquer área de aplicação. Essa geração automática de símbolos e esquemáticos de componentes tem como objetivo, não apenas auxiliar o engenheiro projetista de circuitos em seu trabalho, como também facilitar a criação de bibliotecas contendo símbolos e esquemáticos de componentes, capazes de serem utilizados em simulações de cada componente individual ou até mesmo de sistemas complexos compostos por muitos componentes.

A implementação de uma ferramenta computacional capaz de realizar essas tarefas e sua posterior validação, através de testes realizados com modelos de componentes semelhantes aos para os quais a ferramenta será destinada quando passar a ser usada em ambiente industrial, é precisamente o objetivo do trabalho descrito neste documento. O desenvolvimento de tal ferramenta é parte integrante

de um projeto muito maior, envolvendo aproximadamente 20 empresas e institutos de pesquisa, visando a criação de um conjunto de ferramentas computacionais voltadas à área de desenvolvimento de circuitos integrados.

Este trabalho de conclusão de curso ocupa-se de apresentar os resultados do período de estágio do aluno no *Fraunhofer Institut für Integrierte Schaltungen* (Instituto Fraunhofer para Circuitos Integrados), localizado em Dresden, na Alemanha, quando participou do desenvolvimento de tal ferramenta. Esse instituto é um dos vários institutos integrantes da *Fraunhofer Gesellschaft*, uma das maiores organizações de pesquisa aplicada do mundo. Durante o período de estágio o aluno esteve sob a supervisão de Uwe Knöchel, gerente do Grupo de Circuitos Analógicos e de Rádio-frequência do Instituto Fraunhofer, e em contato com Pietro Brenner, da empresa *Infineon Technologies AG*, uma das várias empresas participantes do projeto, os quais o orientaram durante as diversas etapas de desenvolvimento do trabalho.

O programa da ferramenta foi implementado na linguagem de programação SKILL. O ambiente de desenvolvimento do projeto foi o *Cadence Design Environment* (Ambiente de Desenvolvimento Cadence), o qual além de permitir o desenvolvimento de programas em SKILL, possue uma plataforma com ferramentas para diversas aplicações, em especial ferramentas voltadas ao desenvolvimento de circuitos, como por exemplo, aplicativos para a confecção de diversas formas de representação de circuitos, como símbolos e esquemáticos, e simulação dos circuitos representados pelos mesmos.

Embora o objetivo a longo prazo do projeto do qual o aluno tomou parte seja a criação de uma ferramenta capaz de gerar também outros tipos de representação do componente, apenas a geração do símbolo e do esquemático de pinos do componente estava efetivamente implementada no momento em que o aluno deixou o projeto no Instituto Fraunhofer, e por esse motivo este documento trata de explicar apenas o processo de geração destes, deixando de lado outros possíveis tipos de vistas que a ferramenta possa vir a ser capaz de gerar no futuro.

Cabe aqui dizer que este trabalho de conclusão de curso tem a intenção de explicar as diferentes etapas do processo de geração das vistas de componentes eletrônicos no ambiente Cadence e de que maneira cada uma dessas etapas é realizada pelos módulos da ferramenta criada, sem se preocupar em dar explicações detalhadas sobre o código do programa da ferramenta no entanto. Optou-se por não se incluir neste documento o código em SKILL do programa da ferramenta, já que este, além de possuir uma estrutura muito complexa para ser explicada em detalhes neste documento, também possuí uma extensão de aproximadamente 3000 linhas distribuídas em 9 arquivos, e portanto, sua inclusão aumentaria desnecessariamente o tamanho deste trabalho. Contudo, o apêndice B apresenta o conteúdo de um desses arquivos, referente ao módulo do *Browser* utilizado para a importação dos modelos de componentes para a ferramenta, apenas para exemplificar a estrutura do código no qual a ferramenta foi programada.

Vale ressaltar que a ferramenta final advinda deste trabalho, após ter sido submetida a testes pelo aluno e por parceiros do projeto em aplicações semelhantes às quais ela se destina, se mostrou muito bem-sucedida e por este motivo em breve será aplicada em ambiente industrial.

O capítulo 2 deste documento visa fornecer uma breve explicação dos principais aspectos do Ambiente de Desenvolvimento Cadence, com suas ferramentas, linguagem de programação SKILL e estrutura de bibliotecas.

Cabe ao capítulo 3 explicar toda a estrutura da ferramenta produto deste projeto, com detalhes de sua lógica de funcionamento, seus módulos componentes e todos os processos envolvidos desde a importação dos modelos de componentes até a geração final das vistas dos mesmos.

O capítulo 4, por sua vez, serve para apresentar os resultados dos testes realizados com a ferramenta e consequente validação da mesma, restando ao capítulo 5 as conclusões e considerações finais sobre o trabalho realizado.

2. O AMBIENTE DE DESENVOLVIMENTO CADENCE

O ambiente de desenvolvimento Cadence, utilizado durante todo o desenvolvimento desse projeto, consiste num *software* criado pela *Cadence Design Systems*, uma das companhias líderes mundiais em automação de projeto de eletrônicos (EDA - *Electronic Design Automation*). O ambiente Cadence pode ser instalado apenas em certos sistemas operacionais, sendo que o sistema operacional disponível no Instituto Fraunhofer é o sistema UNIX, no qual todo este trabalho foi desenvolvido.

Cadence é um ambiente de automação de projeto de eletrônicos, que permite integrar em um único ambiente de trabalho diferentes aplicativos e ferramentas, permitindo desenvolver todos os estágios do projeto de circuitos integrados de um único ambiente [1]. Essas ferramentas são completamente gerais, suportando diferentes tipos de tecnologias de fabricação. Existem muitas versões diferentes do conjunto de ferramentas Cadence voltadas ao desenvolvimento de circuitos integrados, sendo que as ferramentas utilizadas neste projeto pertencem à versão Virtuoso.

Não é o objetivo deste trabalho dar uma explicação de todos os aplicativos e ferramentas disponíveis no ambiente de desenvolvimento Cadence. Ao invés disso, será dada uma breve descrição dos principais aspectos e ferramentas do ambiente Cadence envolvidos na realização deste projeto.

2.1 – A Linguagem SKILL

A ferramenta desenvolvida durante este projeto, foi inteiramente implementada na linguagem de programação SKILL. Cadence SKILL é uma linguagem de programação de alto nível, interativa, baseada na popular linguagem de inteligência artificial Lisp e usa uma sintaxe semelhante a linguagem C [2]. SKILL traz uma interface funcional para os subsistemas Cadence subjacentes e permite rapidamente e facilmente customizar aplicativos CAD e desenvolver novos aplicativos.

A linguagem SKILL é a linguagem de extensão das ferramentas Cadence e ela traz funções para, através de um aplicativo de interface de programação, acessar e controlar cada um dos componentes das ferramentas Cadence: o sistema de gerenciamento da interface de usuário, o banco de dados do projeto e os comandos de qualquer ferramenta integrada de projeto. Com a capacidade de extensão, projetistas podem facilmente acrescentar novas capacidades com complexas funções embutidas ao conjunto de ferramentas de projeto Cadence [3].

O poder de SKILL é derivado de grandes bibliotecas de subrotinas chamadas para manipular estruturas de dados de projetos como instâncias, conexões, células, etc. Sem essas rotinas embutidas, SKILL não teria atingido seu papel dominante na EDA na área de circuitos customizados. Entre as vantagens apresentadas pela linguagem SKILL, ressalta-se que ela é ideal para a prototipagem rápida, já que se pode incrementalmente validar cada estágio do algoritmo antes de incorporá-lo num programa maior, o que torna o processo de desenvolvimento de programas muito mais prático.

De especial importância na realização deste projeto, é o fato de que todas as ferramentas e interface do ambiente Cadence são implementados na linguagem de programação SKILL, e com isso, podem ser acessados e modificados pelo usuário através de comandos em SKILL na linha de entrada de comandos da CIW, a janela principal do ambiente Cadence. É justamente esta particularidade que permite a substituição de todo o processo manual de desenho de símbolos e esquemáticos de componentes por rotinas escritas em SKILL que reproduzam o mesmo trabalho automaticamente. Este aspecto do ambiente Cadence será melhor explicado logo adiante, quando as ferramentas Cadence de criação de símbolos e esquemáticos forem apresentadas.

O Apêndice B deste documento apresenta uma parte do programa da ferramenta desenvolvidada neste projeto, e serve como um bom exemplo da estrutura básica de um programa escrito em linguagem SKILL.

2.2 – CIW - Janela de Interface Comum (*Common Interface Window*)

A CIW é a janela principal do ambiente de desenvolvimento Cadence, a qual permite controlar cada parte do mesmo e pode ser executada através do comando "***icfb***" em qualquer janela de terminal do sistema UNIX. A Figura 1 mostra a CIW, através da qual todo o ambiente Cadence juntamente com suas ferramentas é executado.

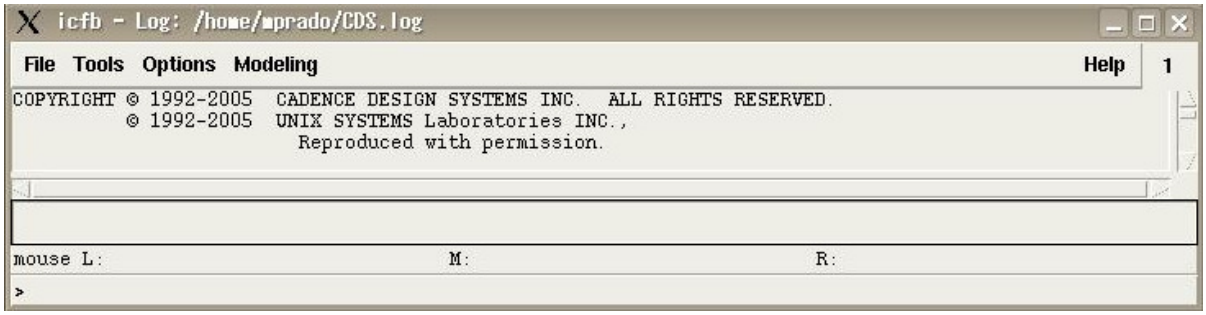


Figura 1 – Janela de Interface Comum (CIW).

Através da CIW pode-se realizar várias tarefas:

- Inicializar todas as ferramentas Cadence.
- Carregar a maior parte do ambiente Cadence.
- Abrir sessões de projeto.
- Vizualizar avisos, erros e outras mensagens informativas.
- Acessar a documentação de ajuda.
- Inicializar o gerenciador de bibliotecas do Cadence.
- Modificar as configurações do ambiente de trabalho Cadence.
- Utilizar a linha de entrada para executar comandos em SKILL.

A linha de entrada de comandos está localizada na parte inferior da CIW. Através dela, comandos na linguagem de programação SKILL podem ser executados tanto para o propósito de criação de novas ferramentas quanto para a modificação do ambiente de trabalho Cadence. Assim como os outros aplicativos do ambiente Cadence, a própria CIW é implementada em *SKILL*, de maneira que o usuário pode modificá-la se assim o desejar, como por exemplo para se criar novas abas de menus além das já existentes.

Todo o código do programa desenvolvido durante o projeto descrito neste documento foi executado nesta linha de entrada de comandos, tanto diretamente através da inserção do mesmo na linha, quanto indiretamente através de comandos de carregamento dos arquivos onde o código encontra-se salvo.

2.3 – O Sistema de Bibliotecas do Cadence

Um aspecto muito importante do ambiente de desenvolvimento Cadence é sua estrutura de bibliotecas (*Libraries*) e células (*Cells*), a qual fornece muitas funcionalidades e vantagens no desenvolvimento de circuitos integrados. Todas as entidades no Cadence são gerenciadas usando bibliotecas [1]. Cada biblioteca cumpre o papel de armazenar uma ou mais células, as quais por sua vez servem ao propósito de representar desde instâncias de componentes simples, como resistores e capacitores, até circuitos integrados extremamente complexos. Cada célula possui um ou mais tipos de vistas (*Cellviews*), as quais nada mais são do que diferentes formas de representação da célula a qual pertencem. A criação automática de tais vistas é o objetivo principal a que se destina a ferramenta desenvolvida no decorrer deste projeto.

Para melhor explicar a estrutura hierárquica do sistema de bibliotecas do Cadence através de um exemplo, o circuito de um microprocessador pode estar armazenado numa biblioteca chamada *microprocessador*, e tal biblioteca conter os diferentes componentes do microprocessador (como por exemplo resistores, capacitores e transístores). Cada um desses blocos (células) pode possuir vários tipos de vistas, as quais podem ser desde um símbolo simplificado (vista símbolo) até um esquemático detalhado (vista esquemático), que inclua as principais características e propriedades do componente, permitindo que o mesmo possa ser usado em simulações quando inserido em esquemáticos de circuitos maiores, como o do microprocessador.

Essa estrutura hierárquica de bibliotecas, células e vistas é similar, e de fato fisicamente corresponde, a um **diretório** (**biblioteca**) contendo **subdiretórios** (**células**), cada um contendo **arquivos** (**vistas**) [1]. Isso pode ser melhor compreendido através de uma análise da árvore de pastas onde as mesmas são armazenadas no sistema UNIX, como por exemplo a mostrada na Figura 2. Neste exemplo, *cadence* é o diretório raiz onde as bibliotecas *lnaSimple*, *ltfm_test* e *mylib* são armazenadas. A biblioteca *mylib* em específico encontra-se com seu conteúdo mostrado, onde pode-se ver as várias células que a compõe, sendo a primeira a célula *actual* e a última a *test3*. Ainda na Figura 2, o

conteúdo da pasta correspondente à célula *RX_V50_LNAHB_v1* encontra-se mostrado à direita, onde fazem-se presentes as vistas *spectre, schematic* e *symbol*.

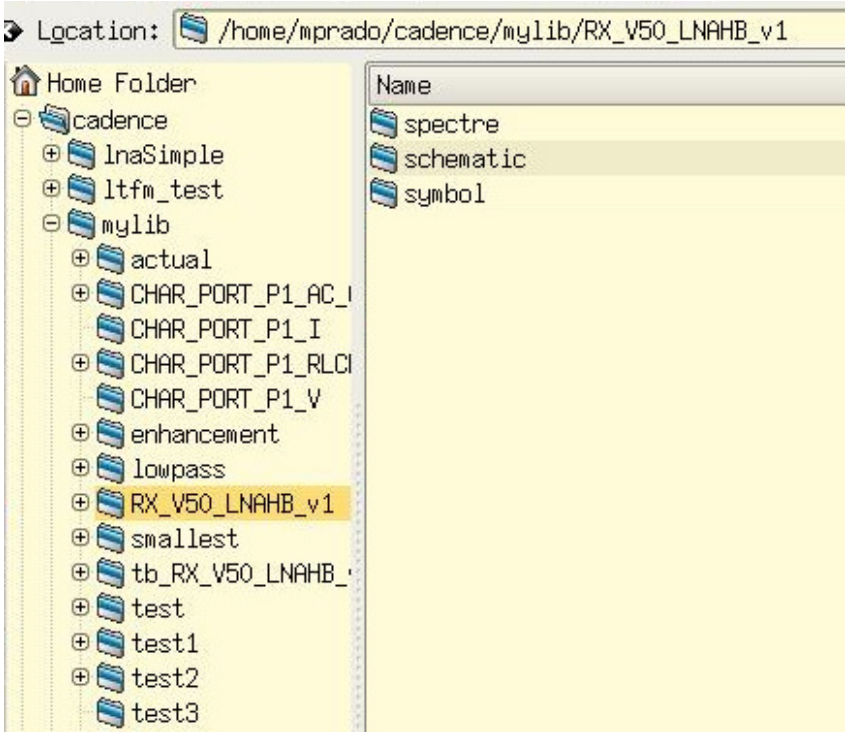


Figura 2 - Estrutura hierárquica das pastas das bibliotecas, células e vistas.

Visando uma maior facilidade de acesso e gerenciamento de tal estrutura de bibliotecas, células e vistas, o ambiente Cadence dispõe do *Browser* mostrado na Figura 3, o qual pode ser acessado selecionando ***Library Manager*** na aba de menu ***Tools*** da CIW.

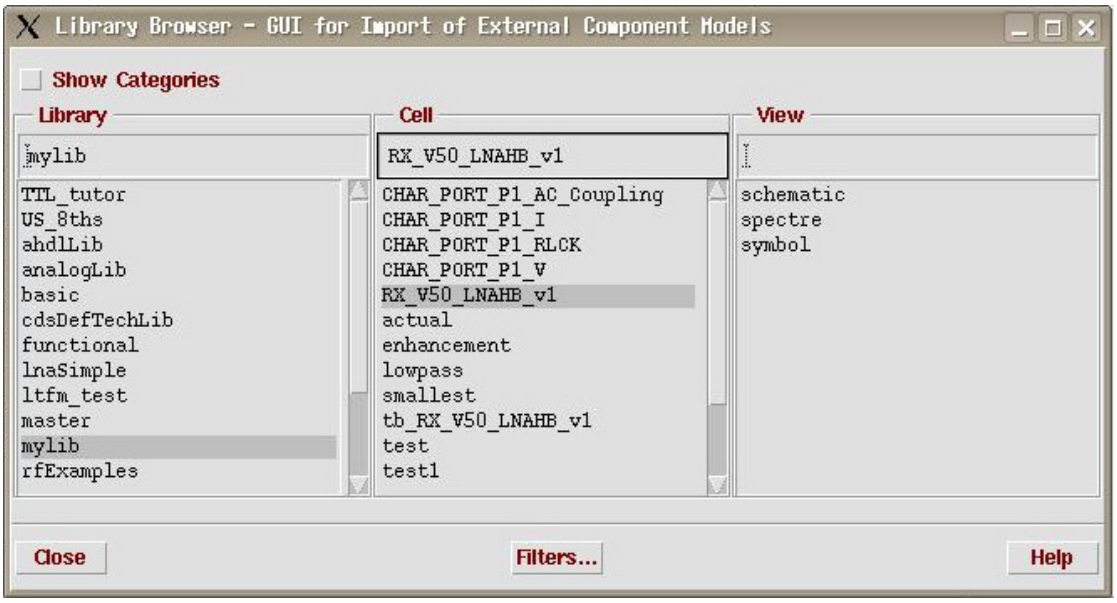


Figura 3 - *Browser* para gerenciamento das bibliotecas, células e vistas do ambiente *Cadence*.

Através desse Browser pode-se vizualizar de maneira mais prática as bibliotecas, células e vistas armazenadas na árvore de pastas do diretório de trabalho do Cadence, tornando assim mais fácil abrir, copiar ou excluir células e vistas.

2.4 – O Compositor de Esquemáticos

O ambiente Cadence dispõe de um aplicativo voltado à composição e edição de esquemáticos de circuitos eletrônicos, o *Cadence Composer Schematic Editor*. O compositor de esquemáticos provê um meio simples e intuitivo de desenhar, posicionar e interconectar os componentes que compõem o circuito que se quer projetar [1]. O esquemático resultante serve ao propósito de descrever as principais propriedades elétricas de todos os componentes e suas interconexões. Também incluídos no esquemático estão as conexões de alimentação e do terra do componente, assim como todos os pinos de entrada e saída de sinais do circuito.

Pode-se abrir o compositor de esquemáticos Cadence através do menu ***Tools*** da CIW. Com ele pode-se inicar uma nova sessão de projeto numa janela em branco do editor ou então abrir um esquemático já existente para modificá-lo. A janela do compositor de esquemáticos pode ser vista na Figura 4. Nela, encontra-se implementado o esquemático de um amplificador operacional.

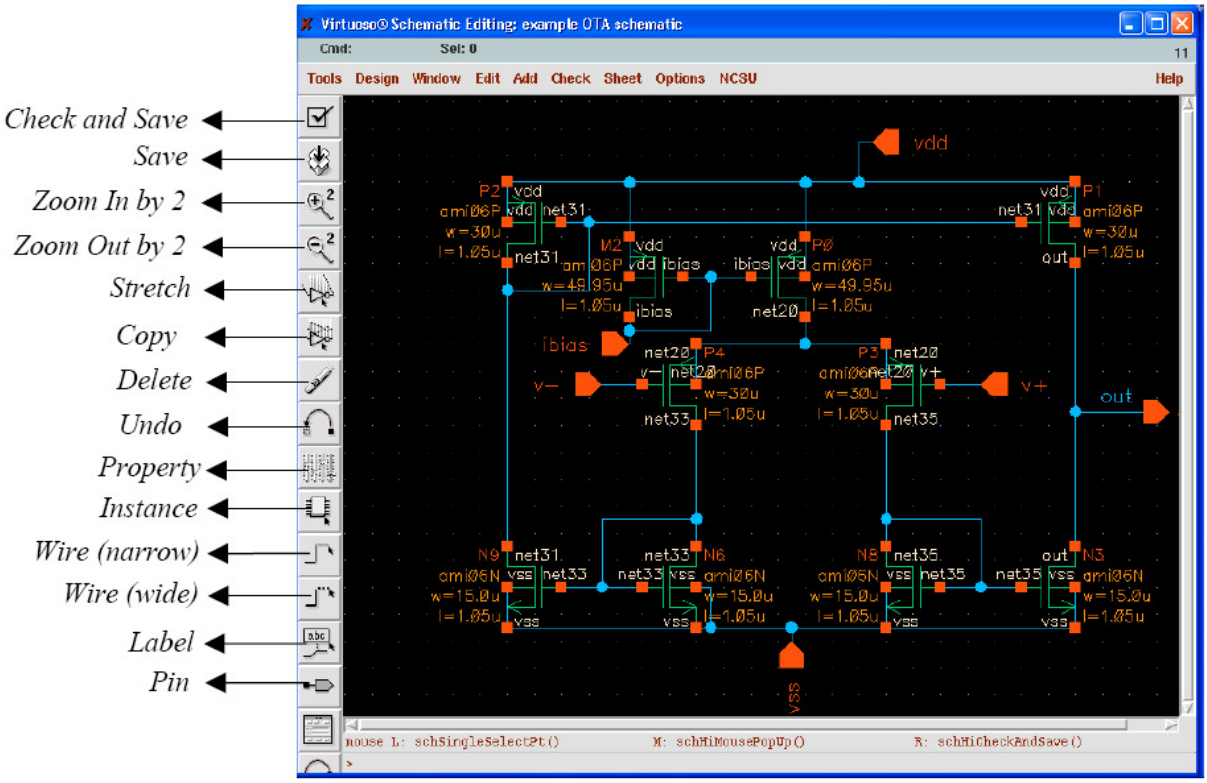


Figura 4 - Janela do Compositor de Esquemáticos [1].

Pode-se usar tanto os menus localizados na parte superior da janela quanto os botões localizados no lado esquerdo para trabalhar nos esquemáticos [1]. No compositor de esquemáticos pode-se basicamente:

- **Criar componentes**: através do botão ***Instance*** pode-se criar no esquemático uma instância de qualquer componente existente nas bibliotecas, como resistores, fontes de tensão ou até mesmo circuitos integrados complexos. Pinos de componentes são criados através do botão ***Pin***.
- **Interconectar componentes**: através do botão ***Wire***, pode-se criar fios para interligar os terminais de componentes. Rótulos (criados a partir do botão ***Label***) também podem ser usados para cumprir esta função, já que dois terminais que possuam o mesmo rótulo são considerados interconectados pelo editor.
- **Configurar as propriedades das instâncias**: pode-se modificar as propriedades de qualquer instância de componente presente no esquemático, como valores de resistores e dimensões de transistores, bastando para isso clicar na instância em questão e depois no botão ***Property***, para que uma janela se abra permitindo modificar cada propriedade do componente.

Existem basicamente duas maneiras de se criar um esquemático:

a. **Esquemático Não-Hierárquico**: Todo o circuito do esquemático é introduzido no mesmo nível. Isso só é possível em projetos pequenos.
b. **Esquemático Hierárquico** Se o projeto for muito grande, deve-se criar pequenos blocos do circuito em um nível hierárquico inferior e depois criar símbolos para estes blocos, para então usá-los como componentes básicos em níveis hierárquicos superiores do projeto.

Quando um circuito consiste de componentes hierarquicamente menores (ou módulos), é geralmente muito benéfico usar a abordagem hierárquica, identificando primeiro tais módulos no projeto e então atribuindo a cada um deles um símbolo correspondente para representar o circuito do módulo. Essa etapa simplifica enormemente a representação esquemática do sistema completo. A vista símbolo do circuito de um módulo nada mais é do que um ícone que representa todos os componentes dentro deste módulo.

A criação do esquemático da topologia de um circuito é geralmente seguida pela criação de um símbolo que represente o circuito inteiro. A aparência padrão de um símbolo é um simples bloco retangular com os pinos de entrada e saída do módulo, mas essa aparência pode ser facilmente modificada se assim se desejar. Uma vez criado um símbolo para o módulo, este símbolo pode agora ser usado como bloco de construção de outro módulo, e assim por diante, permitindo ao projetista de circuitos criar um sistema completo consistindo de múltiplos níveis hierárquicos.

2.5 – O Compositor de Símbolos

Para criar o símbolo de um módulo é necessário primeiro ter criado pinos para o módulo no compositor de esquemáticos através do botão ***Pin*** [1]. Os pinos podem ser do tipo entrada, saída ou entrada/saída, permitindo a entrada e saída de sinais, alimentação e conexão de terra do módulo. Esses serão os pinos acessíveis de fora do símbolo.

Feito isto, selecionando ***Design > Create Cellview > From Cellview*** na aba de menu do compositor de esquemáticos, aparecerá uma janela para a criação de uma vista símbolo a partir da vista esquemático do módulo. A janela do compositor de símbolos já apresentará uma aparência padrão para o símbolo, que consiste num simples retângulo com os pinos externos do módulo saindo dele e pode ser editada pelo usuário (mudando formas ou a distribuição dos pinos, por exemplo) se este assim desejar. A Figura 5 mostra a janela do compositor de símbolos do ambiente Cadence e as ações associadas a cada um dos botões do lado esquerdo da janela. Nesta figura em específico, vê-se que o formato retangular simples padrão do símbolo foi modificado para que se obtivesse o símbolo triangular usual de um amplificador operacional, sendo que o retângulo externo serve apenas de caixa de seleção do símbolo (através da qual o usuário pode selecioná-lo para executar ações como arrastá-lo) e não aparecerá visualmente no símbolo, quando este for usado na composição de circuitos. Neste caso, os pinos **v+**, **v-**, **vdd**, **out**, **vss** e **ibias** são os pinos que permitirão a conexão do módulo com componentes externos em esquemáticos**.**

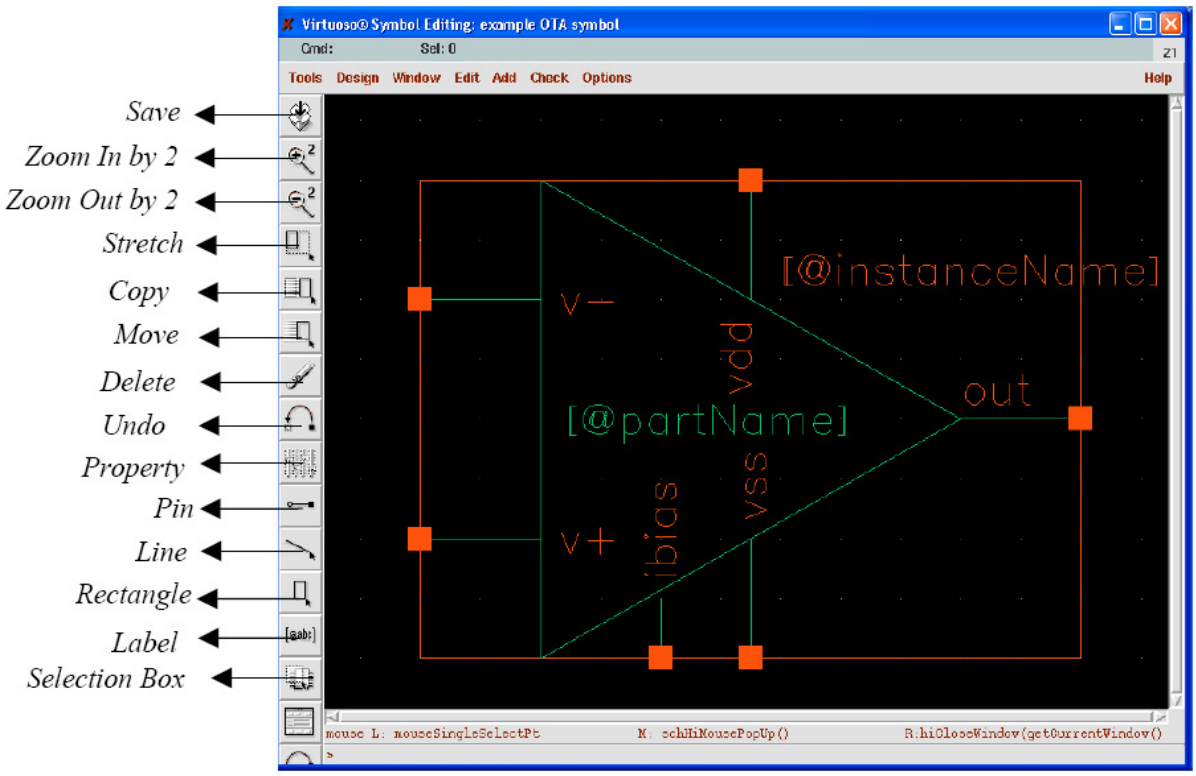


Figura 5 - Janela do Compositor de Símbolos [1].

Uma vez criado um símbolo para o módulo, este poderá ser usado sempre que se quiser criar instâncias do módulo em esquemáticos hierarquicamente superiores, bastando selecioná-lo através do botão ***Instance*** no compositor de esquemáticos, da mesma maneira que se faria com qualquer outro componente. A Figura 6 mostra o esquemático de um circuito que consiste numa instância do amplificador operacional com seus pinos conectados a resistores, capacitores e fontes de corrente e tensão.

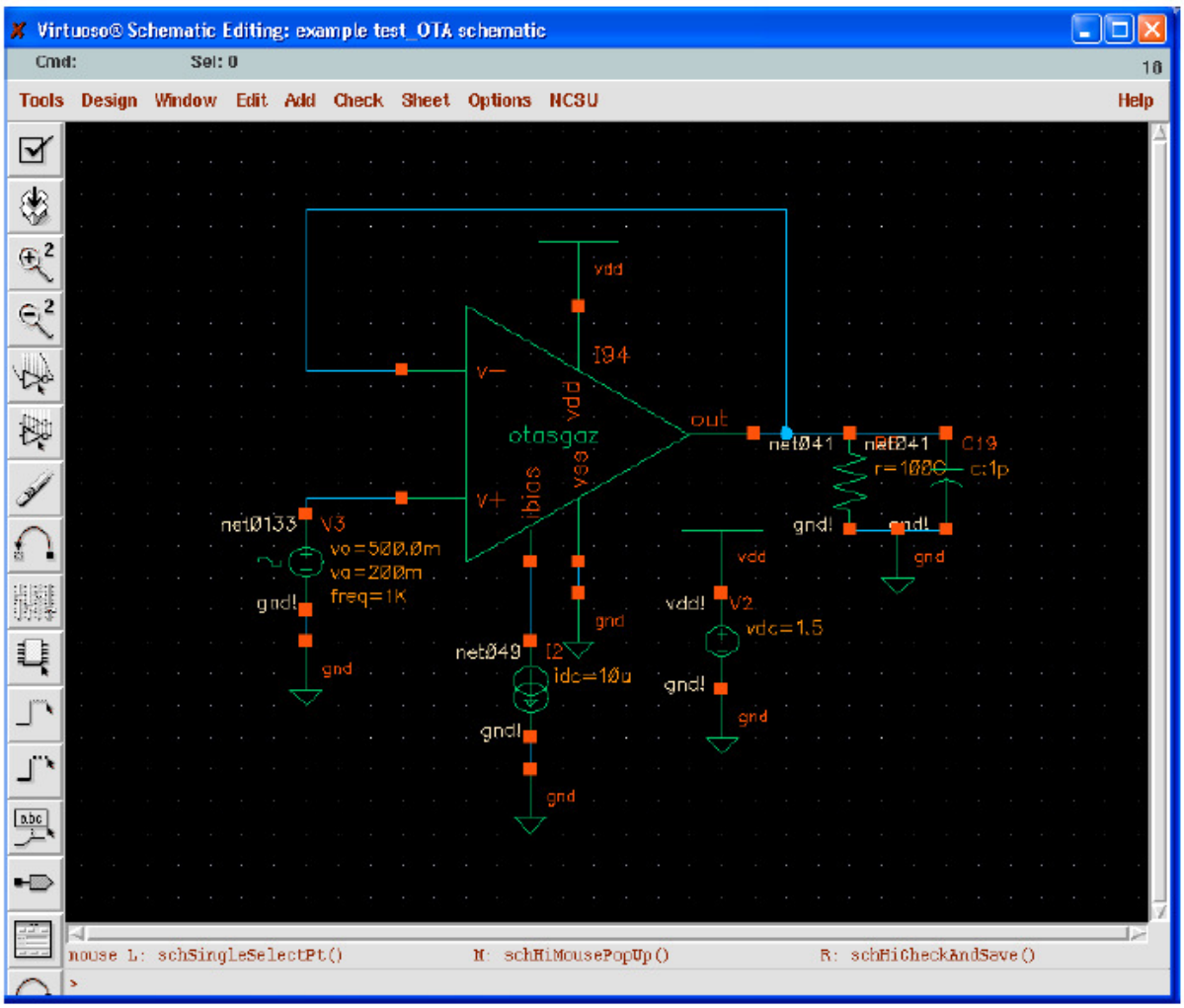


Figura 6 - Exemplo de um esquemático hierárquico utilizando o símbolo do amplificador operacional [1].

Fica evidente que neste caso o uso do símbolo do amplificador operacional é muito mais conveniente do que incluir o esquemático completo do amplificador operacional, tornando o esquemático do sistema completo muito mais simples e prático. Pode-se alternar facilmente entre os níveis hierárquicos superiores e inferiores do sistema através dos menus do compositor de esquemáticos.

O objetivo principal da ferramenta criada neste projeto é justamente automatizar a criação de tais esquemáticos e símbolos de componentes, usando apenas modelos descritivos dos componentes para isso. Isso é possível pois, assim como com todos os aplicativos do ambiente de desenvolvimento

Cadence, cada botão ou elemento do compositor de esquemáticos e do compositor de símbolos está implementado em linguagem SKILL, sendo dessa maneira possível substituir todas as ações manuais que seriam realizadas pelo usuário (como a criação de instâncias, formas geométricas e rótulos) por rotinas compostas por comandos em SKILL que façam exatamente a mesma coisa.

3. A FERRAMENTA IECM

O objetivo final desse projeto foi o desenvolvimento de uma ferramenta capaz de importar diferentes tipos de modelos de componentes eletrônicos, organizar as informações contidas em tais modelos, checar por erros de sintaxe e formato tanto nos arquivos quanto nas informações contidas nos mesmos e por fim gerar como saída as vistas do símbolo e do esquemático do componente descrito pelos modelos importados para a ferramenta. Essa geração automática de símbolos e esquemáticos visa eliminar o processo laborioso envolvido na criação manual dos mesmos, poupando assim muito tempo e trabalho nessa etapa do projeto de circuitos.

A criação automática das vistas é possível graças à existência de funções SKILL dedicadas especificamente à criação dos elementos constituintes das vistas, como por exemplo instâncias, pinos, fios, figuras geométricas, campos de texto e basicamente qualquer outro elemento disponível no ambiente de trabalho dos compositores de esquemáticos e símbolos do Cadence. Como cada parte das ferramentas do ambiente Cadence é implementada em SKILL, é efetivamente possível substituir cada ação manual do usuário nas ferramentas Cadence por comandos SKILL equivalentes, e é justamente nesse fato que se apóia a lógica de funcionamento do programa da ferramenta desenvolvida neste projeto. Dessa maneira, ao invés da seleção de um pino na barra de ferramentas da janela do compositor de esquemáticos e subsequente posicionamento manual do pino na vista, o programa chama alternativamente a função SKILL responsável pela criação dos pinos no compositor de esquemáticos, passando como argumentos as coordenadas onde o pino deve ser posicionado. De forma semelhante, cada elemento constituinte das vistas das célula pode ser criado de maneira automática, dispensando assim o processo trabalhoso e demorado geralmente desempenhado manualmente pelo projetista.

É importante ressaltar que um dos objetivos principais de longo prazo da ferramenta criada durante este projeto é possibilitar a criação de grandes bibliotecas de células de componentes contendo diversos tipos de vistas, que forneçam aos projetistas de circuitos símbolos e esquemáticos de componentes capazes de serem utilizados em simulações e análises dos circuitos sendo projetados. Essas vistas de componentes, por sua vez, após submetidas a simulações em conjunto com vistas de outros componentes, poderiam ser associadas para constituírem circuitos maiores compostos por muitos componentes, para os quais a ferramenta criaria igualmente vistas que pudessem ser associadas para constituir circuitos ainda maiores e hierarquicamente superiores aos anteriores, e assim sucessivamente, até que se atinja o nível hierárquico mais superior e se tenha as respectivas vistas representativas do sistema completo que se deseja construir.

Devido ao processo de geração de vistas a partir da importação de modelos de componentes, a ferramenta final foi batizada de ***Import of External Component Models*** e abreviada de **IECM**. A IECM é composta por vários módulos, cada um deles servindo a diferentes propósitos no processo de

leitura dos modelos do componente e geração final das vistas do mesmo. Cabe ressaltar que embora cada módulo seja responsável por funções diferentes e específicas dentro da estrutura da IECM, ainda assim possuem entre si uma interdependência e sequencialidade no fluxo de execução da IECM de maneira que, para a geração bem sucedida dos produtos finais, é essencial o correto funcionamento de cada um dos módulos envolvidos na execução da IECM.

Para uma melhor vizualização das várias etapas que se sucedem desde a importação dos modelos de componentes até a saída de seus símbolos e esquemáticos correspondentes, um diagrama do fluxo de execução da IECM pode ser visto na Figura 7.

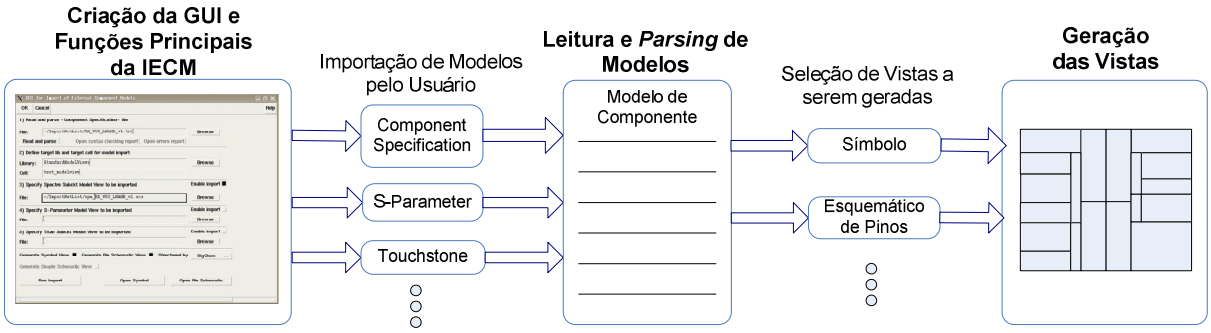


Figura 7 - Diagrama de fluxo de execução da IECM.

Para uma compreensão mais prática da estrutura do código, o mesmo se encontra dividido em três estágios:

- Criação da GUI e funções principais da IECM.
- Leitura e *Parsing* de modelos.
- Geração das Vistas.

Os módulos do programa desenvolvidos para a execução de cada estágio da ferramenta IECM são explicados nas próximas seções deste documento, com os nomes dos arquivos onde os módulos foram implementados e as principais funções que os compõem. A seção 3.1 é destinada ao estágio responsável pela criação da GUI da IECM e pelas funções principais do programa, as quais coordenam a chamada dos vários módulos da ferramenta. A seção 3.2 cuida de explicar todo o processo envolvido na importação de modelos de componentes para a ferramenta. Finalmente, os módulos responsáveis pela geração das vistas do componente usando as informações retiradas dos modelos importados são explicados em detalhes na seção 3.3.

3.1 – Criação da GUI e Funções Principais da IECM

3.1.1 - GUI

Arquivo:

ifxIecmGUI.il

Funções:

ifxIecmImportExternalModelsGUI()

ifxIecmImportSpecFileButtonCB()

A linguagem SKILL possui funções específicas para a criação de estruturas chamadas formulários. Estes formulários servem ao propósito de criar uma interface gráfica para o usuário, através de menus, janelas de mensagens, botões de seleções de opções e vários outros elementos que permitem uma interação entre o usuário e os programas desenvolvidos no ambiente Cadence. Esses formulários foram utilizados para se construir a Interface Gráfica de Usuário ou GUI (*Graphical User Interface*) da IECM, a qual pode ser vista na Figura 8.

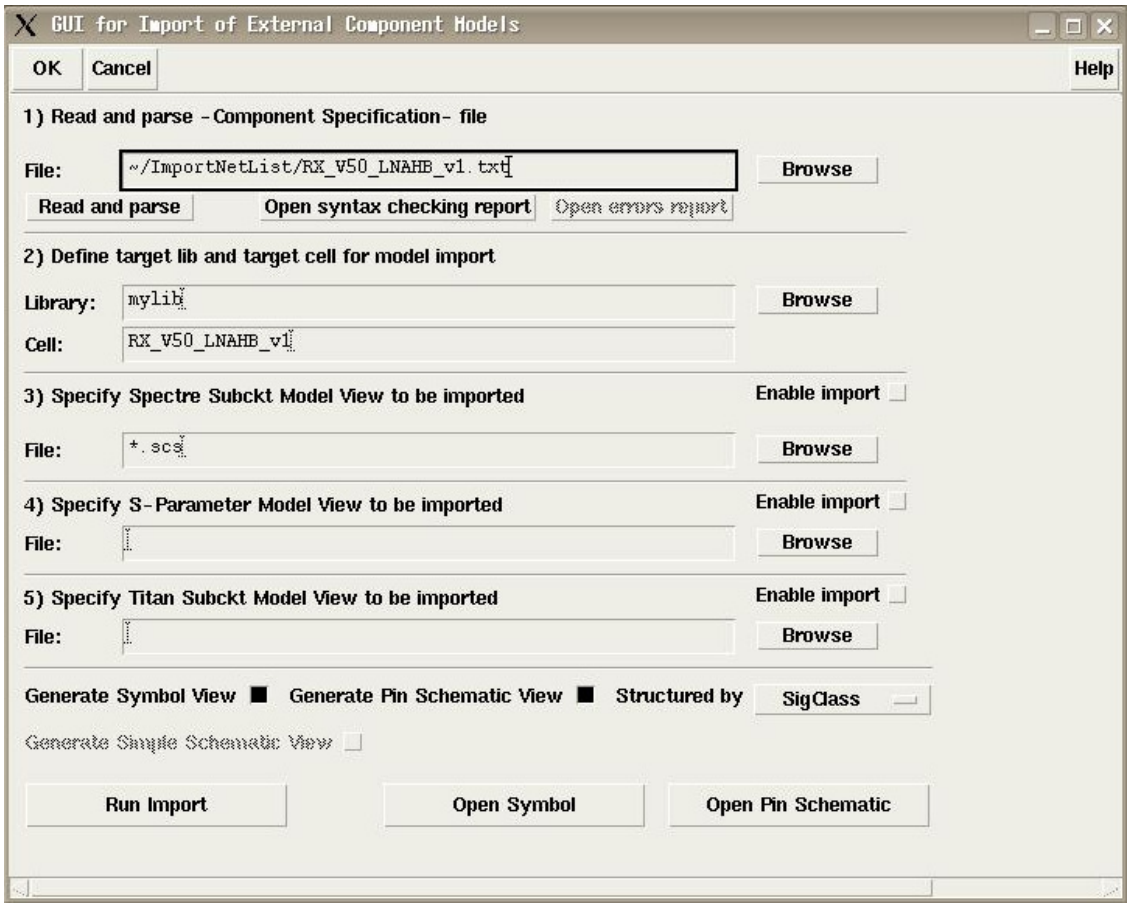


Figura 8 - GUI da Ferramenta IECM.

Cada área na GUI serve a um processo ou estágio de execução específico da ferramenta IECM conforme descrito a seguir:

1) Read and parse -Component Specification- file (Leitura e *Parsing* do arquivo da *Component Specification*): Figura 9.

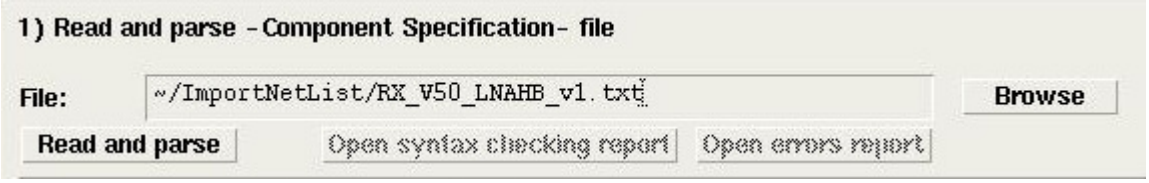


Figura 9 - Seção *1) Read and parse -Component Specification- file*.

Essa seção é usada para importar o arquivo da *Component Specification*, a qual contém as informações do componente que serão usadas na geração do símbolo e do esquemático do componente. Aqui se tem o botão ***Browse*** para chamar um *Browser* que permite que o usuário selecione um arquivo através das estruturas de pastas do computador ao invés de ter que digitar o caminho completo do arquivo no campo ***File***. Explicações e detalhes sobre este *Browser* (o qual foi criado especialmente para a importação de modelos para a IECM) são dados na seção 3.1.2 deste documento.

Após a seleção do arquivo da *Component Specification*, o usuário pode pressionar o botão ***Read and parse*** para chamar um *Parser* para analisar os dados contidos no arquivo e procurar erros em relação ao formato padronizado que a *Component Specification* deve ter. Com isso, todo o conteúdo da *Component Specification* é submetido a uma busca de erros, organizado e armazenado numa variável do programa, para que assim os dados do componente possam ser posteriormente usados na geração de suas vistas. Tanto a estrutura padrão da *Component Specification* quanto o *Parser* e as funções em SKILL nas quais ele é implementado são explicados em detalhes na seção 3.2.1 deste documento.

Após o arquivo ter sido analisado pelo *Parser*, o botão ***Open syntax checking report*** é ativado para que o usuário possa ver os resultados gerados pelo *Parser* para a análise de cada linha da *Component Specification*, e caso erros tenham sido encontrados, o botão ***Open errors report*** é ativado também para que possam ser vistos apenas os erros encontrados na *Component Specification*.

Caso erros tenham sido encontrados, o programa envia uma mensagem avisando o usuário e sugerindo que tais erros sejam corrigidos antes de prosseguir com a geração das vistas do componente, já que certos erros na *Component Specification* podem impedir a geração correta das vistas.

2) Define target lib and target cell for model import (Definir biblioteca e célula de destino para importação de modelo): Figura 10.

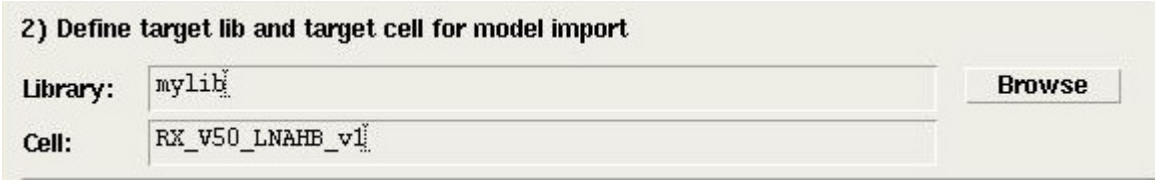


Figura 10 - Seção *2) Define target lib and target cell for model import*.

Essa seção contém dois campos que permitem que o usuário digite os nomes da biblioteca e célula onde as vistas geradas pela IECM serão armazenadas. Esta seção também possui um botão ***Browse***, o qual chama o *Browser* de gerenciamento de bibliotecas do *Cadence* (apresentado na seção 2.4 deste documento) para a seleção de uma biblioteca e célula já existentes. Se a célula escolhida como destino já existir, o programa perguntará ao usuário se ele deseja que as vistas da mesma sejam substituídas pelas novas vistas geradas pela IECM.

3) Specify Spectre Subckt Model View to be imported (Especificar modelo Spectre de subcircuito a ser importado): Figura 11.

Figura 11 - Seção *3) Specify Spectre Subckt Model View to be imported*.

Permite ao usuário selecionar um modelo Spectre do componente a ser importado para a ferramenta. Esta seção é composta pelo campo ***File***, o botão ***Browse*** (que chama o mesmo *Browser* da seção de importação da *Component Specification*) e o botão ***Enable import***, que habilita a importação do arquivo Spectre. Quando ***Enable import*** é ativado, o arquivo definido no campo ***File*** é analisado por um *Parser* (diferente do usado para checar a *Component Specification*) para verificar se o arquivo sendo importado é válido e segue a estrutura padrão exigida para modelos Spectre. Caso tenham sido encontrados erros durante a análise do arquivo do modelo Spectre, o usuário é avisado pelo programa para corrigí-los. A estrutura padrão exigida para modelos Spectre, assim como o *Parser* criado para a análise dos mesmos, encontram-se explicados na seção 3.2.2 deste documento.

4) Specify S-Parameter Model View to be imported (Especificar modelo S-Parameter a ser importado): Figura 12.

Figura 12 - Seção *4) Specify S-Parameter Model View to be imported*.

Permite ao usuário selecionar um modelo Touchstone (um tipo específico de modelo S-parameter) a ser importado para a ferramenta. Esta seção é similar à seção anterior e, mais uma vez, se o botão ***Enable import*** é ativado pelo usuário, o arquivo do modelo Touchstone é analisado por um *Parser* (criado especificamente para a checagem de modelos Touchstone) para verificar se o modelo sendo importado é válido. Após a análise do arquivo do modelo Touchstone, caso erros tenham sido encontrados, o usuário é avisado para corrigí-los. Explicações e detalhes sobre modelos Touchstone, formato padrão exigido para eles e o *Parser* implementado para analisar tais modelos encontram-se na seção 3.2.3 deste documento.

5) Specify Titan Subckt Model View to be imported (Especificar modelo Titan a ser importado): Figura 13.

Figura 13 - Seção *5) Specify Titan Subckt Model View to be imported*.

Essa seção servirá ao propósito de permitir ao usuário importar modelos Titan do componente para a ferramenta. O módulo da IECM correspondente a essa seção ainda não começou a ser implementado e por esse motivo essa seção não é utilizada na versão atual da IECM.

Seção de seleção de vistas a serem geradas: Figura 14.

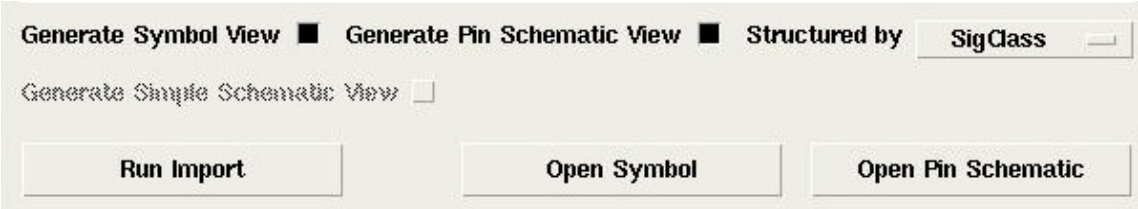


Figura 14 - Seção de seleção de vistas a serem geradas.

Na parte superior desta área, o usuário escolhe os tipos de vistas que a ferramenta *Import of External Component Models* irá gerar através do uso dos dados contidos na *Component Specification*. Os tipos de vistas que podem ser gerados são: **Símbolo** e **Esquemático de Pinos**. Em versões futuras da IECM será também possível selecionar nesta seção um esquemático simples do componente para ser gerado, mas em sua versão atual tal opção encontra-se desabilitada.

O botão ***Structured by*** (canto superior direito) permite ao usuário selecionar através de qual das seguintes propriedades dos pinos do componente as vistas serão estruturadas: *SigClass*, *PinDescription*, *SymPinPos*, *XtSevClass* e *PinType*. A propriedade selecionada aqui determinará a configuração final que as vistas do símbolo e do esquemático de pinos do componente terão. Detalhes deste recurso são dados na seção 3.3.1 deste documento.

O botão ***Run Import*** (canto inferior esquerdo) realiza várias ações sequencialmente, começando por importar a *Component Specification* e checar por erros nele (exatamente como o que acontece ao se pressionar o botão ***Read and parse*** da seção 1 da GUI), e prosseguindo com a chamada dos módulos do programa responsáveis pela geração das vistas selecionadas pelo usuário: símbolo, esquemático de pinos ou ambos. O motivo da existência do botão ***Read and Parse***, presente na seção 1 da GUI, é apenas permitir que o usuário possa checar por erros na *Component Specification* e corrigí-los antes de prosseguir com o pressionamento do botão ***Run Import***, e consequente geração das vistas do componente, sendo que nos casos em que já se saiba de antemão que a *Component Specification* não possuí erros, o usuário pode simplesmente pressionar ***Run Import*** sem utilizar previamente ***Read and parse***. Todos os passos executados pelo programa para a geração das vistas do componente são explicados na seção 3.3 deste documento.

Finalmente, terminada a geração das vistas, os botões ***Open Symbol*** e ***Open Pin Schematic*** abrem as vistas do símbolo e do esquemático de pinos gerados para vizualização.

3.1.2 – *Browser* de importação de modelos

Arquivo:

ifxIecmBrowse.il

Funções:

ifxIecmBrowser()

ifxIecmFileBoxSingleClickCB()

ifxIecmDirBoxSingleClickCB()

ifxIecmDirBoxDoubleClickCB()

O *Browser* implementado com a finalidade de auxiliar o usuário a importar modelos para a ferramenta IECM pode ser visto na Figura 15. O *Browser* possui duas áreas distintas, uma para a seleção de diretórios (lado esquerdo) e outra para a seleção de arquivos (lado direito).

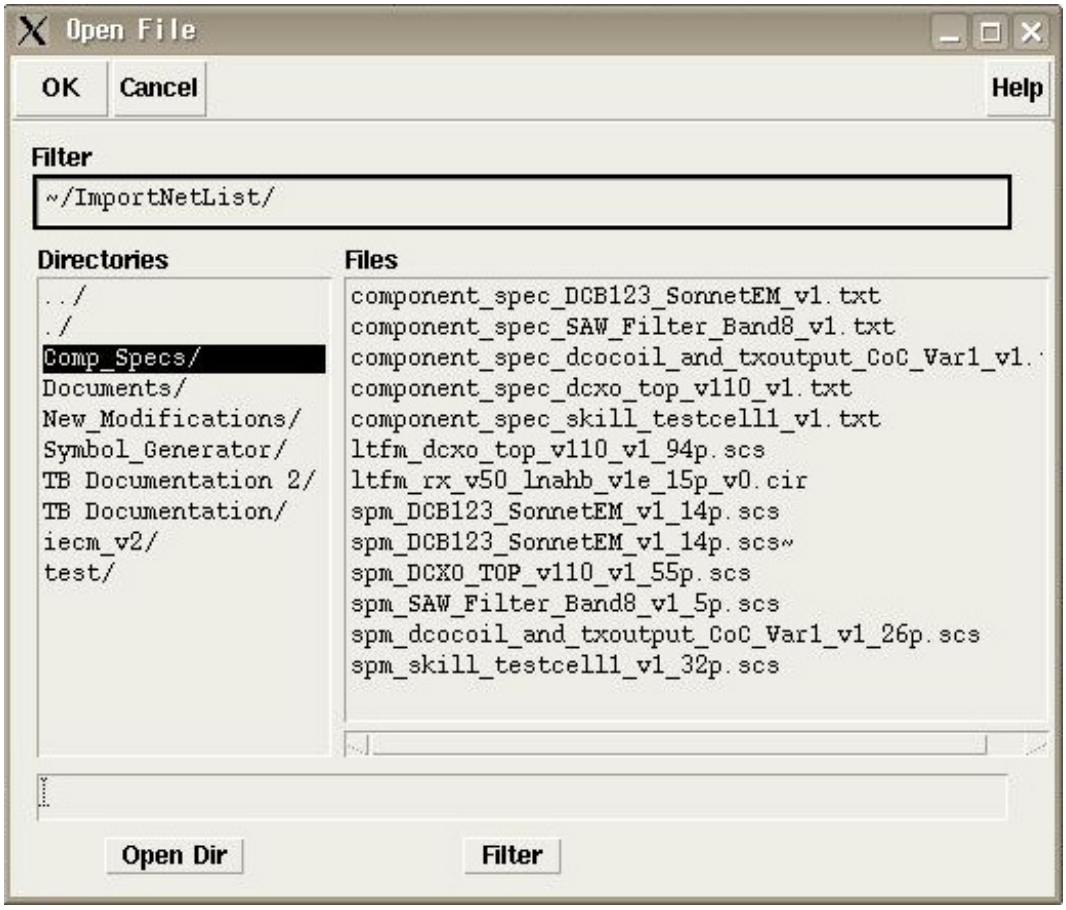


Figura 15 - *Browser* para auxiliar na importação de modelos.

Quando um diretório na área de seleção de diretórios recebe um único clique, todos os arquivos dentro deste diretório são mostrados na área de seleção de arquivos. Quando um diretório é clicado duas vezes, a área de diretórios é atualizada para mostrar todos os subdiretórios dentro do diretório clicado e a área de seleção de arquivos é atualizada para mostrar os arquivos dentro do diretório clicado. Além disso, o campo localizado na parte superior da janela do *Browser* é atualizado com o caminho do diretório atual sendo exibido. Como uma alternativa para o clique duplo na área de diretórios, há o botão ***Open Dir***, que tem o mesmo efeito que o clique duplo em um diretório e foi criado para resolver problemas de tempos de resposta do duplo clique do mouse encontrados em certos sistemas.

O botão ***Filter*** permite que o usuário digite o caminho de um diretório no campo ***Filter*** (na parte superior da janela) e ao pressionar o botão de filtro, mostrar todo seu conteúdo na área de diretórios e na área de arquivos.

Quando um arquivo é selecionado na área de seleção de arquivos, o campo na parte inferior da janela do *Browser* é atualizado com o caminho completo do arquivo. Um clique duplo em um arquivo na área de seleção de arquivos, fecha o *Browser* e retorna o caminho completo deste arquivo para o campo ***File*** da GUI.

3.1.3 – Chamada de Módulos

Arquivo:

ifxIecmCallModules.il

Funções:

ifxIecmMain()

ifxIecmInitWorkSpace()

Quando o usuário pressiona o botão ***Run Import*** na GUI, a função *ifxIecmMain()* é chamada para controlar quais dos módulos da ferramenta IECM devem ser executados. Dependendo das opções selecionadas pelo usuário através da GUI, a ferramenta irá executar algumas tarefas e não executar outras, como por exemplo, procurar por erros no arquivo de modelo Spectre mas não no arquivo de modelo Touchtone, gerar a vista símbolo do componente mas não a vista esquemático de pinos, ou qualquer outra das muitas combinações possíveis que atenda aos objetivos do usuário.

A função *ifxIecmMain()* começa sua execução chamando a função *ifxIecmInitWorkSpace()* para inicializar as variáveis globais de trabalho do programa e criar um diretório de trabalho para armazenar as vistas das células geradas pela IECM.

Após a execução de *ifxIecmInitWorkSpace()*, a função *ifxIecmMain()* chama então o *Parser* responsável pela análise da *Component Specification* e recebe do mesmo a variável *CompSpec*, a qual contém todas as informações da *Component Specification* organizadas para que possam ser usadas por todos os módulos da ferramenta IECM posteriormente. Se não foram encontrados erros na *Component Specification*, ou foram encontrados erros mas o usuário optou por prosseguir com a geração das vistas do componente, *ifxIecmMain()* prossegue com a geração das vistas, verificando uma por uma as opções selecionadas pelo usuário na GUI:

- Se o botão ***Generate Symbol View*** está habilitado, a função *ifxIecmCreateSymbol()* é chamada para criar a vista do símbolo do componente.
- Se o botão ***Enable import*** da seção de importação de modelos Spectre está habilitado, a função *ifxIecmCheckSpectre()* é chamada para analisar e verificar a validade do modelo Spectre sendo importado.
- Se o botão ***Enable import*** da seção de importação de modelos S-Parameter está habilitado, a função *ifxIecmCheckTouchstone()* é chamada para analisar e verificar a validade do modelo Touchstone sendo importado.
- Se o botão ***Generate Pin Schematic View*** está habilitado, a função *ifxIecmCreatePinSchematic()* é chamada para gerar a vista do esquemático de pinos do componente.

3.2 – Leitura e *Parsing* de Modelos

De fundamental importância no fluxo de execução da IECM, são os módulos do programa responsáveis pela leitura dos modelos de componentes sendo importados, organização das informações contidas nos mesmos, e por fim, checagem de erros de sintaxe e de formato destas, processo também conhecido como *parsing*. Esses modelos são também chamados de *netlists*, arquivos de texto escritos em código ASCII que descrevem as propriedades elétricas e estruturais de um componente e podem ser usados como entrada em simuladores específicos para cada tipo de *netlist*, para realizar simulações e análises dos componentes descritos por estas. Existe um número enorme de tipos diferentes de *netlists*, cada um possuindo sintaxe e características próprias.

As próximas seções visam descrever as ações tomadas pelo programa com cada um dos três tipos de *netlists* que a IECM é capaz de importar: ***Component Specification***, **Spectre** e **Touchstone**. Não é intenção deste trabalho explicar em detalhes a estrutura ou as características destes três tipos de *netlists*, sendo que as próximas seções se restringirão apenas às seções das *netlists* que são analisadas ou manipuladas de alguma maneira pelo programa da IECM durante a importação delas. Uma atenção especial é dada à *Component Specification*, já que são exatamente as informações obtidas dela que são usadas pela ferramenta para a geração das vistas do componente.

3.2.1 – Leitura e Análise de Erros de *Component Specifications*

Arquivos:

ifxIecmCreateComponentDB.il

ifxIecmDataEntryFormatSpecification.il

Funções:

ifxIecmReadComponentSpec()

ifxIecmParseComponentSpec()

A *Component Specification*

As informações necessárias para a geração das vistas da célula do componente são extraídas do arquivo de especificação de componente, conhecido como *Component Specification*, o qual deve ser importado pelo usuário através da seção *1) Read and parse -Component Specification- file* da GUI. Neste arquivo, devem estar presentes não apenas informações sobre o componente como um todo, como também sobre as propriedades individuais de cada pino do componente.

A *Component Specification* nada mais é do que um tipo específico de modelo de descrição de componentes, inventado pelo grupo do projeto especificamente para o uso com a IECM. Assim como

outros tipos de modelos, a *Component Specification* deve ser escrita respeitando-se uma sintaxe específica e os formatos e valores das entradas de cada uma das seções que compõem o modelo devem respeitar os formatos e valores pré-estabelecidos no arquivo *ifxIecmDataEntryFormatSpecification.il*, o qual pode ser visto no apêndice C deste documento.

Um exemplo de *Component Specification* pode ser visto na Figura 16.

```
* BEGIN IMPORT HEADER
* ==========================================
* BEGIN COMMENTS
* This is a Component Specification Example.
*
* END COMMENTS
* ==========================================
* BEGIN COMPONENT DESCRIPTION
* DESIGNATION            , RX_V50_LNAHB_v1
* CDBA PATH              , U:\mprado\modelfiles\test
* TARGET LIB             , mylib
* TARGET CELL            , RX_V50_LNAHB_v1
* NUMBER OF PINS         , 13
* COMPONENT TYPE         , wiring_chip
* LOWER RIGHT COORDINATE , 20:-15
* END COMPONENT DESCRIPTION
* ==========================================
* BEGIN DEFINE PIN PROPERTY COLUMNES
* SubcktPin, PortNr, PinName, PinType, BusWidth, IntNetName, SigClass, XtSevClass, SymPinPos, HsupDomain, LsupDomain, LoadModel, NomDcBias, Xpos, Ypos
* END DEFINE PIN PROPERTY COLUMNES
* ==========================================
* BEGIN PIN PROPERTY DEFINITION
*  I          ,  1  , I        , input   ,  1  , I        , SIG_V_IN  , --- , left   , VDD1 , VSS1 , (V_DC (Vval=2.5))  , --- , --- , ---
*  Ix         ,  2  , Ix       , input   ,  1  , Ix       , SIG_V_IN  , --- , left   , VDD1 , VSS1 , (V_DC (Vval=2.5))  , --- , --- , ---
*  PON        ,  3  , PON      , input   ,  1  , PON      , SIG_V_IN  , --- , left   , VDD1 , VSS1 , (V_DC (Vval=2.5))  , --- , --- , ---
*  CLAMP_ON   ,  4  , CLAMP_ON , input   ,  1  , CLAMP_ON , SIG_V_IN  , --- , left   , VDD1 , VSS1 , (V_DC (Vval=2.5))  , --- , --- , ---
*  ibias      ,  5  , ibias    , input   ,  1  , ibias    , BIAS_I_IN , --- , bottom , VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , ---
*  O          ,  6  , O        , output  ,  1  , O        , SIG_V_OUT , --- , right  , VDD1 , VSS1 , (R (Rval=1k))      , --- , --- , ---
*  Ox         ,  7  , Ox       , output  ,  1  , Ox       , SIG_V_OUT , --- , right  , VDD1 , VSS1 , (R (Rval=1k))      , --- , --- , ---
*  GAIN       ,  8  , GAIN     , input   ,  1  , GAIN     , SIG_V_IN  , --- , right  , VDD1 , VSS1 , (V_DC (Vval=2.5))  , --- , --- , ---
*  R_trim<4:0> , (9;10;11;12;13) , R_trim<4:0> , input , 5 , R_trim<4:0> , SIG_V_IN , --- , right , VDD1 , VSS1 , (V_DC (Vval=2.5)) , --- , ---, ---
* END PIN PROPERTY DEFINITION
* ==========================================
* END IMPORT HEADER
```

Figura 16 - Exemplo de um arquivo de *Component Specification*.

O início de cada nova linha do modelo é identificada por um asterisco (*). Dessa maneira, linhas que não começarem com um asterisco são consideradas pelo programa uma continuação da linha anterior. Cada área da *Component Specification* é iniciada pela palavra “BEGIN” e encerrada por “END”. No topo do documento há uma área de comentários para que se possa incluir uma breve descrição ou comentários sobre o componente. Terminada a área de comentátios, começam as três seções da *Component Specification*, nas quais as características e propriedades do componente são definidas, e são justamente essas seções que serão lidas pelo *Parser* e usadas posteriormente na geração das vistas do componente. Para cada uma dessas seções, diferentes ações são tomadas dentro da função *ifxIecmReadComponentSpec()* do programa. As três seções que compõem a *Component Specification* são explicadas a seguir:

- DESCRIÇÃO DE COMPONENTE (*COMPONENT DESCRIPTION*)

As informações presentes nesta seção do *Component Specification* são lidas e verificadas quanto a presença de erros ou falta de informações. As seguintes informações devem estar sempre presentes nesta seção: designação do componente, biblioteca de destino, célula de destino, número de pinos, tipo de componente e coordenada do canto inferior direito do símbolo a ser gerado.

- DEFINIÇÃO DE COLUNAS DE PROPRIEDADES DOS PINOS (*DEFINE PIN PROPERTY COLUMNES*)

Esta seção é usada para declarar as colunas que devem estar presentes na seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS. Os nomes de cada coluna são separados entre si por vírgulas. O programa compara as colunas declaradas nesta seção com as colunas definidas na variável *ifxIecmTemplate_PinPropSpec* (declarada no arquivo *ifxIecmDataEntryFormatSpecification.il*), em busca de colunas faltantes ou extras. Colunas extras (colunas que não se encontrarem declaradas em *ifxIecmTemplate_PinPropSpec*), incluídas pelo usuário para fins de informação adicional, são permitidas mas não serão verificadas quanto à sintaxe ou formato pelo *Parser*.

- DEFINIÇÃO DE PROPRIEDADES DOS PINOS (*PIN PROPERTY DEFINITION*)

Cada linha nesta seção corresponde às propriedades de um pino ou de um barramento do componente. As colunas das propriedades são separadas entre si por vírgulas (“,”) e devem seguir a ordem definida na seção anterior DEFINIÇÃO DE COLUNAS DE PROPRIEDADES DOS PINOS. O programa lê cada linha da seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS e divide-a entre as colunas de propriedades, para que cada entrada possa ser analisada separadamente pelo *Parser* posteriormente. Todas as propriedades de cada pino do componente são definidas aqui, seguindo as especificações de sintaxe e de formato permitidos definidas centralmente na variável *ifxIecmDataEntryFormatSpec*, a qual é declarada no arquivo *ifxIecmDataEntryFormatSpecification.il*. Dessa maneira, para se fazer qualquer modificação nos valores e formatos permitidos para uma coluna de propriedade dos pinos basta apenas alterar-se no arquivo *ifxIecmDataEntryFormatSpecification.il* as definições de entradas permitidas para esta coluna.

Na Tabela 1 tem-se as colunas que devem estar sempre presentes e com suas entradas seguindo as especificações estabelecidas no arquivo *ifxIecmDataEntryFormatSpecification.il*.

Tabela 1 – Colunas de propriedades que devem estar sempre presentes na *Component Specification*

Coluna	**Formato das entradas**	**Descrição**
SubcktPin	string	Nome do pino em modelos *Spectre*, *Titan* e *Hspice*.
PortNr	número inteiro	Número do pino em modelos *Touchstone*.
PinName	string	Nome do pino nas vistas do símbolo e esquemáticos.
PinType	lista de opções	Direção do pino no símbolo e esquemáticos, podendo ser: entrada, saída ou entrada/saída.
BusWidth	número inteiro	Caso seja 1, essa porta do componente é um pino simples. Caso seja 2 ou maior, indica que a porta se trata de um barramento com número de pinos igual a

		entrada dessa coluna.
IntNetName	string	É usada para indicar quais pinos estão internamente conectados uns aos outros. Entre pinos com mesmo *IntNetName* deve existir apenas pequenas resistências e indutâncias em série.
SigClass	lista de opções	Define qual a classe dos sinais do pino.
XtSevClass	lista de opções	A classe de severidade do *crosstalk* do pino.
SymPinPos	lista de opções	Indica a qual lado do símbolo do componente pertence o pino.
HsupDomain	lista de opções	Define qual a tensão de alimentação positiva correspondente ao pino.
LsupDomain	lista de opções	Define qual a tensão de alimentação negativa correspondente ao pino.
LoadModel	modelo de estrutura	Define um modelo de impedância simples do pino e é geralmente usada para pinos dos tipos entrada e entrada/saída.
NomDcBias	modelo de estrutura	Define o modelo bias c.c. nominal do pino e é geralmente usada para pinos dos tipos saída e entrada/saída.
Xpos	número real	Permite definir a coordenada X do ponto onde o pino deve ser colocado no símbolo do componente. Na versão atual da IECM a entrada desta coluna não é utilizada.
Ypos	número real	Permite definir a coordenada Y do ponto onde o pino deve ser colocado no símbolo do componente. Na versão atual da IECM a entrada desta coluna não é utilizada.

Detalhes sobre o formato e sintaxe exigidos para as entradas de cada coluna de propriedade dos pinos podem ser encontrados no arquivo *ifxIecmDataEntryFormatSpecification.il*, o qual pode ser visto no apêndice C deste documento.

O *Parser* da *Component Specification*

Um *Parser* implementado nas funções *ifxIecmReadComponentSpec()* e *ifxIecmParseComponentSpec()* (ambas contidas no arquivo *ifxIecmCreateComponentDB.il*) é usado para verificar cada linha da *Component Specification*, à procura de erros de sintaxe e de formato das entradas de cada uma das colunas de propriedades do componente em relação às especificações de *ifxIecmDataEntryFormatSpecification.il*. Este *Parser* é um dos módulos mais importantes da IECM, já que as informações obtidas da *Component Specification* por ele são essenciais para a geração posterior de todas as vistas do componente.

Enquanto *ifxIecmReadComponentSpec()* é a responsável pela leitura e organização das informações contidas na *Component Specification*, a função *ifxIecmParseComponentSpec()* é responsável pela análise de todas as informações. Os seguintes tipos de erros são verificados pelo *Parser*:

- O caminho para o arquivo da *Component Specification* aponta para um arquivo existente.
- Todas as três seções DESCRIÇÃO DE COMPONENTE, DEFINIÇÃO DE COLUNAS DE PROPRIEDADES DOS PINOS e DEFINIÇÃO DE PROPRIEDADES DOS PINOS estão presentes.
- O tipo de componente definido na seção DESCRIÇÃO DE COMPONENTE é um dos elementos da lista de tipos permitidos definidos em *ifxIecmDataEntryFormatSpecification.il*.
- Função *ifxIecmCheckColumnsSection()*: As colunas da seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS são as mesmas que as especificadas no arquivo *ifxIecmDataEntryFormatSpecification.il*.
- Função *ifxIecmCheckLength()*: O número de colunas na seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS é o mesmo que o número de colunas especificadas na seção DEFINIÇÃO DE COLUNAS DE PROPRIEDADES DOS PINOS.
- Função *ifxIecmCheckUnique()*: A entrada das colunas que deveriam ter valores únicos não se repetem em qualquer outra linha da *Component Specification*. No caso da coluna *IntNetName*, verifica se a entrada na coluna *IntNetName* não se repete em qualquer uma das entradas da coluna *PinName*.
- Função *ifxIecmCheckBus()*: Verifica se o número de entradas em cada coluna de propriedade dos pinos e dos barramentos de pinos está correto, além de checar se a extensão das entradas que devem ter extensão está correta.
- Função *ifxIecmCheckNumbers()*: Verifica se a entrada das colunas que devem receber apenas números é realmente um número e se o número é do tipo especificado em *ifxIecmDataEntryFormatSpec*: inteiro ou real. Se for um número inteiro, verifica se o

número é limitado ao intervalo [low, high] definido em *ifxIecmDataEntryFormatSpec* para a coluna em questão.

- Função *ifxIecmCheckOptions()*: Verifica se a entrada da coluna está entre a lista de valores permitidos para esta coluna.
- Função *ifxIecmCheckStrings()*: Verifica se a entrada da coluna é uma string que contém apenas os caracteres permitidos.
- Função *ifxIecmCheckBinary()*: Verifica se a entrada da coluna é um número binário.
- Função *ifxIecmCreateModelStructure()*: Organiza as especificações para as estruturas de modelo (colunas *LoadModel* e *NomDcBias*) presentes em *ifxIecmDataEntryFormatSpec* e organiza as informações da entrada da coluna na *Component Specification* e em seguida verifica se há erros na entrada da coluna na *Component Specification* em relação à estrutura de modelo especificada em *ifxIecmDataEntryFormatSpec* para esta coluna em específico.
- Função *ifxIecmSpecialSyntax()*: Verifica se a coluna em específico aceita a sintaxe especial definida por **[-]+** e se a entrada da coluna na *Component Specification* respeita esta sintaxe especial. Quando especificado em *ifxIecmDataEntryFormatSpec* como um dos valores aceitos para uma coluna em específico (por exemplo, a coluna *SigClass*), a sintaxe especial [-]+ permite ao usuário não especificar nenhuma entrada para esta coluna em questão, escrevendo um número arbitrário de caracteres "-" e " " para indicar esta ausência de entradas.

Cada vez que o *Parser* realiza uma das checagens listadas acima, uma mensagem de texto é gerada contendo detalhes do resultado desta checagem. Todas as mensagens geradas são impressas no arquivo de texto *LogFile.txt*, o qual é salvo no diretório de trabalho da célula (pasta da célula de destino selecionada na GUI) e pode ser lido pelo usuário através do pressionamento do botão *Open syntax checking report* na GUI. Caso não sejam encontrados erros, o programa envia a mensagem mostrada na Figura 17 ao usuário.

No caso da checagem realizada pelo *Parser* apontar a presença de um erro na *Component Specification*, além da mensagem contendo os detalhes do erro ser impressa no aquivo *LogFile.txt*, uma cópia da mesma é impressa também em outro arquivo de texto chamado *ErrorsFile.txt*, localizado igualmente no diretório de trabalho da célula. O arquivo *ErrorsFile.txt*, o qual conterá apenas as mensagens de erros geradas pelo *Parser,* pode então ser lido pressionando-se o botão *Open errors report* na GUI, o qual é ativado apenas caso pelo menos um erro tenha sido encontrado. Caso erros tenham sido encontrados, o programa envia a mensagem mostrada na Figura 18 ao usuário, alertando-lhe disso e perguntando-lhe se deseja de fato prosseguir com a geração das vistas.

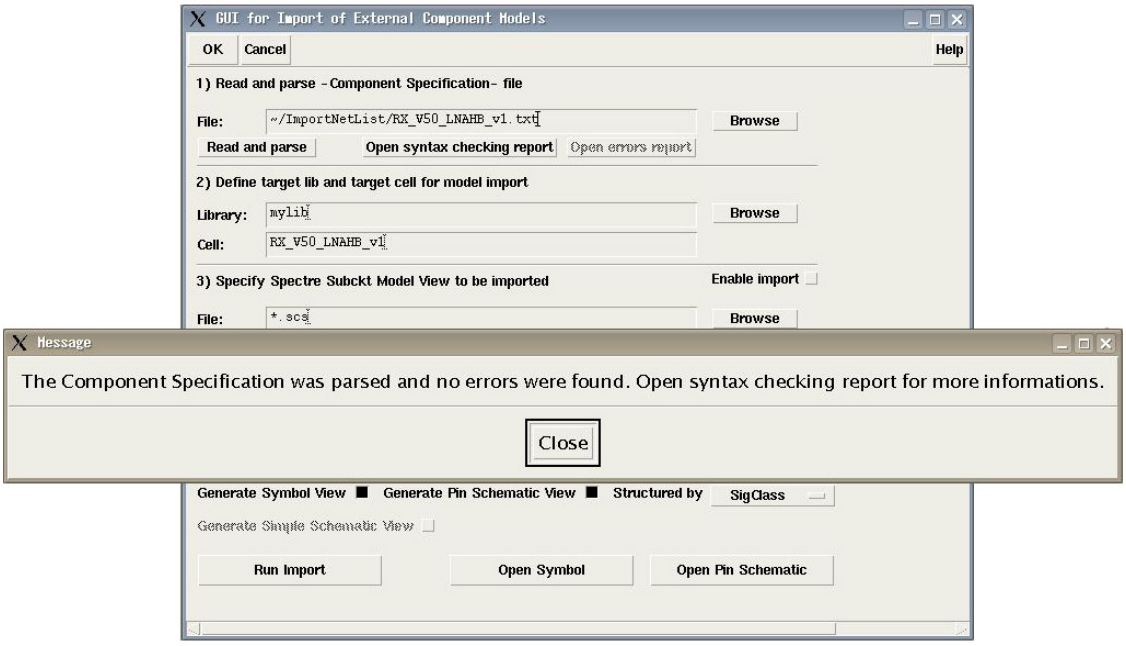


Figura 17 - Mensagem enviada ao usuário caso não hajam erros na *Component Specification.*

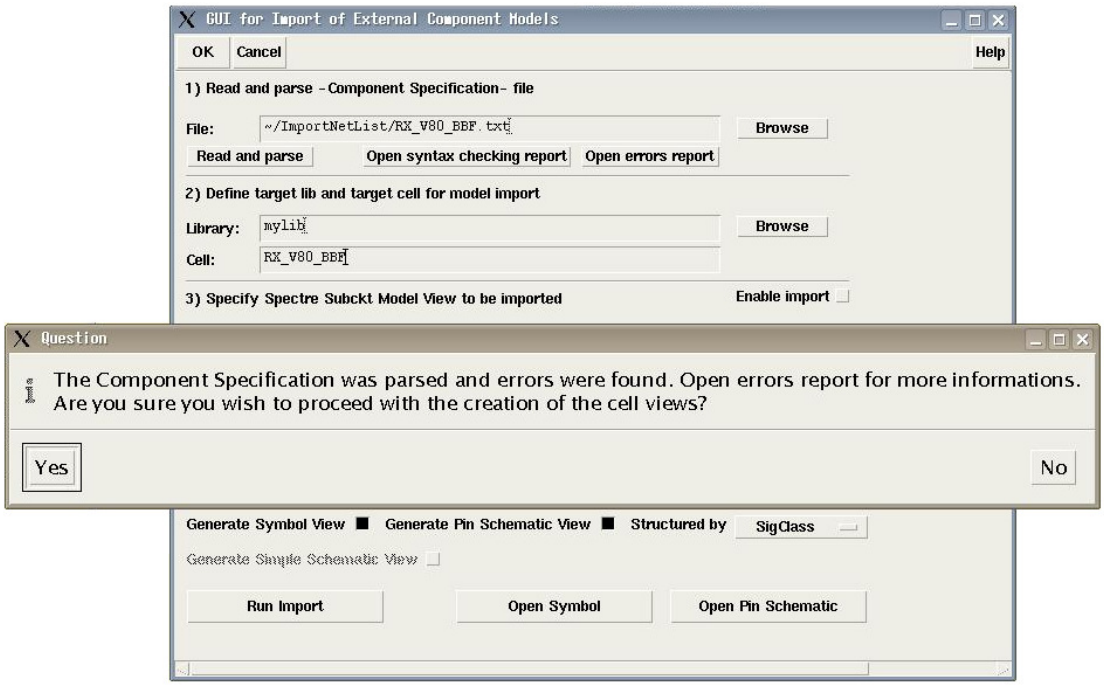


Figura 18 - Mensagem enviada ao usuário caso hajam erros na *Component Specification.*

Um exemplo do arquivo *LogFile.txt* gerado pelo *Parser* contendo todos os resultados da análise da *Component Specification* pode ser visto na Figura 19. Na Figura 20 tem-se o *ErrorsFile.txt*, gerado pelo *Parser* exclusivamente nos casos em que erros tenham sido encontrados e possuindo uma lista de todos os erros.

```
/home/mprado/cadence/mylib/RX_V50_LNAHB_v1/LogFile.txt
File                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                Help 13

[                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                LOG FILE

ERROR SUMMARY
=============
All the lines of the Component Specification File have been read and NO errors were found!

PARSING MESSAGES
================

The Component Type >wiring_chip< defined in the COMPONENT DESCRIPTION section is in the allowed options list.

The DEFINE PIN PROPERTY COLUMNES section has more Columns than the defined in the Template!
The Template have <15> Columns but <16> were found in the DEFINE PIN PROPERTY COLUMNES section.

All the columns defined in the Template were found in the DEFINE PIN PROPERTY COLUMNES section.

The DEFINE PIN PROPERTY COLUMNES section has extra columns that were not defined in the Template!
The following extra Columns were found: <("PinDescription")>.

========================================================================== Next line 1 ==========================================================================
The PIN PROPERTY DEFINITION Section has <16> Columns as the defined in the DEFINE PIN PROPERTY COLUMNES section.

Column Name     Status                                                                Parsing Result
===========     ======       ==========================================================================================================================

SubcktPin         OK    ==>   >I< is a valid String.

--------------------------------------------------------------------------------------------------------------------------------------------------------------

PortNr            OK    ==>   >1< is a valid Integer Number in the allowed interval [1,999].

--------------------------------------------------------------------------------------------------------------------------------------------------------------

PinName           OK    ==>   >I< is a valid String.

--------------------------------------------------------------------------------------------------------------------------------------------------------------

PinType           OK    ==>   >input< is element of the allowed options list.

--------------------------------------------------------------------------------------------------------------------------------------------------------------

BusWidth          OK    ==>   >1< is a valid Integer Number in the allowed interval [1,99].

--------------------------------------------------------------------------------------------------------------------------------------------------------------

IntNetName        OK    ==>   >I_IntName< is a valid String.

--------------------------------------------------------------------------------------------------------------------------------------------------------------

SigClass          OK    ==>   >SIG_V_IN< is element of the allowed options list.
```

Figura 19 - *Log File* gerado pelo *Parser* contendo os resultados da análise da *Component Specification.*

Apesar de alguns tipos de erros na seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS da *Component Specification* não impedirem a geração bem-sucedida da vista do símbolo e do esquemático de pinos da célula do componente, é sempre recomendado que o usuário da IECM corrija todos os erros encontrados pelo *Parser* antes de prosseguir com a geração das vistas. As colunas *PinName*, *PinType*, *BusWidth*, *SigClass* e *SymPinPos*, em específico, jamais podem ter erros, caso contrário o símbolo não será gerado corretamente.

Após a leitura e análise de todas as linhas da *Component Specification*, os dados contidos em suas três seções são organizados e armazenados de forma estruturada dentro de vetores na variável *CompSpec*. Estes vetores recebem os nomes de cada coluna de propriedade dos pinos (*SubcktPin*, *PortNr* e assim por diante) e cada um de seus índices é usado para acessar as entradas da linha de um pino específico. Assim, para se recuperar as entradas do primeiro pino do componente basta chamar os vetores usando o índice zero, como por exemplo, **PinName[0]** ou **SigClass[0]**. Dessa maneira, o programa pode acessar as entradas contidas na *Component Specification* facilmente durante o processo de criação das vistas da célula do componente.

```
/home/mprado/cadence/mylib/RX_V80_BBF/ErrorsFile.txt
File                                                                                                                    Help   14

                                                                                ERRORS LOG

ERROR ==> It wasn't possible to find the COMPONENT DESCRIPTION section!

ERROR SUMMARY
=============
All the lines of the Component Specification File have been read and ERRORS were found!

Line 1 have ERRORS in the Columns: ("IntNetName")
Line 2 have ERRORS in the Columns: ("IntNetName")
Line 3 have ERRORS in the Columns: ("IntNetName")
Line 4 have ERRORS in the Columns: ("IntNetName")
Line 5 have ERRORS in the Columns: ("IntNetName")
Line 6 have ERRORS in the Columns: ("IntNetName")
Line 7 have ERRORS in the Columns: ("IntNetName")
Line 8 have ERRORS in the Columns: ("IntNetName")
Line 9 have ERRORS in the Columns: ("IntNetName")
Line 10 have ERRORS in the Columns: ("IntNetName")
Line 11 have ERRORS in the Columns: ("IntNetName")
Line 12 have ERRORS in the Columns: ("IntNetName")
Line 13 have ERRORS in the Columns: ("IntNetName")
Line 14 have ERRORS in the Columns: ("IntNetName")
Line 15 have ERRORS in the Columns: ("IntNetName")
Line 16 have ERRORS in the Columns: ("IntNetName")
Line 17 have ERRORS in the Columns: ("IntNetName")
Line 18 have ERRORS in the Columns: ("IntNetName")
Line 19 have ERRORS in the Columns: ("IntNetName")
Line 20 have ERRORS in the Columns: ("IntNetName")
Line 21 have ERRORS in the Columns: ("IntNetName")
Line 22 have ERRORS in the Columns: ("IntNetName")

ERROR MESSAGES
==============

COMPONENT DESCRIPTION section ==> Component Type is not present!

DEFINE PIN PROPERTY COLUMNES section ==> The DEFINE PIN PROPERTY COLUMNES section has fewer Columns than the defined in the Template!
The section should have at least <15> Columns but only <13> were found.

DEFINE PIN PROPERTY COLUMNES section ==> Some of the columns defined in the Template could not be found in the Component Specification File!
The following Columns are missing: <("PortNr" "Xpos" "Ypos")>.

Line 1  IntNetName       ==>   >INi< should be unique but it already exists in the column <PinName> of the line 1!

Line 2  IntNetName       ==>   >INix< should be unique but it already exists in the column <PinName> of the line 2!

Line 3  IntNetName       ==>   >INq< should be unique but it already exists in the column <PinName> of the line 3!

Line 4  IntNetName       ==>   >INqx< should be unique but it already exists in the column <PinName> of the line 4!

Line 5  IntNetName       ==>   >VBG< should be unique but it already exists in the column <PinName> of the line 5!

Line 6  IntNetName       ==>   >Ib< should be unique but it already exists in the column <PinName> of the line 6!

Line 7  IntNetName       ==>   >BBF_ON< should be unique but it already exists in the column <PinName> of the line 7!

Line 8  IntNetName       ==>   >R_FUSE<4:0>< should be unique but it already exists in the column <PinName> of the line 8!

Line 9  IntNetName       ==>   >BBFG<2:0>< should be unique but it already exists in the column <PinName> of the line 9!

Line 10 IntNetName       ==>   >ANALOGMUX<3:0>< should be unique but it already exists in the column <PinName> of the line 10!

Line 11 IntNetName       ==>   >ADC_TST< should be unique but it already exists in the column <PinName> of the line 11!

Line 12 IntNetName       ==>   >ADCNFL_TST< should be unique but it already exists in the column <PinName> of the line 12!
```

Figura 20 - *Errors File* gerado pelo *Parser* contendo o relatório de erros da *Component Specification.*

3.2.2 – Análise de Erros de Modelos Spectre

Arquivo:

ifxIecmCheckModelValidity.il

Função:

ifxIecmCheckSpectre()

A linguagem das *netlists* Spectre é baseada na linguagem própria das *netlists* do programa de simulação de circuitos SPICE, mas possuí alguns recursos adicionais que SPICE não possuí. Arquivos de modelos Spectre devem sempre possuir a extensão "**.scs**". O circuito é descrito através de uma série de linhas de elementos, que definem a topologia do circuito e os valores dos elementos, e uma série de linhas de controle, que definem os parâmetros do modelo e os controles de simulação [4]. Cada elemento no circuito é especificado por uma linha de elemento que contém o nome do elemento, os nós do circuito aos quais o elemento está conectado, e os valores dos parâmetros que determinam as características elétricas do elemento. Campos numa linha são separados por um ou mais espaços em branco, uma vírgula, sinal de igual, ou parêntese esquerdo ou direito.

Entre as diversas seções presentes num modelo Spectre, cabe aqui explicar apenas a seção de definição de subcircuitos, já que é nela que o programa da IECM busca por erros quando um modelo Spectre é importado para a IECM. Uma definição de subcircuito contém uma lista de elementos SPICE, um nome e especifica os nós do circuito que o conectam ao circuito principal [5]. Subcircuitos podem conter elementos básicos de circuitos, outras definições de circuitos, modelos de dispositivos e chamadas a subcircuitos definidos internamente ou externamente. A forma geral de uma definição de subcircuito é:

```
subckt  NomeDoSubckt Nó1 Nó2 Nó3 ...

Linhas dos elementos do circuito

ends
```

- A definição do subcircuito começa com a linha **subckt**, seguida do nome do subcircuito.
- **Nó1**, **Nó2**, **Nó3**, e assim por diante, são os nós externos do subcircuito.
- As linhas dos elementos do circuito definem o subcircuito.
- A linha **ends** encerra a definição do subcircuito.

Um exemplo de definição de subcircuito seria:

```
subckt top_subcir in out_model
 I14 (out_model 0) sub_cir2
 I0 (net1 out_model 0) lowpass_cir1 c=1n r=50 l=10u
 R1 (in net1) resistor r=50
ends top_subcir
```

Quando um modelo Spectre é importado através da seção *3) Specify Spectre Subckt Model View to be imported* da GUI, com a correspondente ativação do botão ***Enable import***, um *Parser* implementado na função *ifxIecmCheckSpectre()* é usado para procurar por erros no arquivo, de forma semelhante ao que é feito com a *Component Specification*. Este *Parser* faz as seguintes checagens:

- Verifica se a extensão do arquivo é de fato "**.scs**".
- Obtém todas as informações contidas nas seções **subckt** do modelo, o que consiste nos nomes de cada subcircuito, elementos que o definem e nomes de seus subcircuitos hierarquicamente inferiores.
- Determina qual dos subcircuitos definidos nas seções **subckt** é hierarquicamente superior a todos os outros subcircuitos definidos no modelo.

- Verifica cada um dos nomes dos elementos nas seções **subckt** quanto à presença de caracteres não permitidos. Imprime todos os erros em um arquivo de log chamado *SpectreLog.txt*, salvo no diretório de trabalho da célula.

Em versões futuras da ferramenta IECM, os dados contidos no modelo Spectre importado serão utilizados para a geração de outros tipos de vistas do componente, diferentes do símbolo e esquemático de pinos. Em sua versão atual, no entanto, a IECM apenas é capaz de importar os modelos Spectre e checar por erros nos mesmos, sem contudo utilizar as informações obtidas deles.

3.2.3 –Análise de Erros de Modelos Touchstone

Arquivo:

ifxIecmCheckModelValidity.il

Função:

ifxIecmCheckTouchstone()

Um arquivo Touchstone (também conhecido como arquivo SnP) é um arquivo de texto ASCII usado para documentar os parâmetros de uma rede de n-portas de um dispositivo ativo ou de uma rede passiva de interconexão [6]. Os arquivos Touchstone têm sido aceitos como um padrão para a transferência de dados de redes dependentes de frequência de n-portas. Conforme especificado pela *Agilent Corporation* (criadora do Touchstone), além de seguir um formato específico, todo arquivo Touchstone deve possuir uma *Option Line*, a qual possui o seguinte formato:

 # <unidade de frequência> <parâmetro> <formato> R <n>

Onde:

- **#** - marca o início da *Option Line*
- **unidade de frequência** - especifica a unidade de freqüência. Os valores válidos para esse campo são GHz, MHz, KHz, Hz. O valor padrão é GHz.
- **parâmetro** - especifica que tipo de dados de parâmetros da rede estão contidos no arquivo. Os valores válidos são S, Y, Z, H, G. O valor padrão é S.
- **formato** - especifica o formato dos pares de dados dos parâmetros da rede. Os valores válidos são DB, MA e RI. O valor padrão é MA.
- **R <n>** - especifica a resistência de referência em ohms, onde *n* é um número positivo de ohms (a impedância real pela qual os parâmetros são normalizados). A resistência de referência padrão é 50 ohms.

Os parâmetros da *Option Line* são separados por um ou mais espaços em branco e a linha é terminada com um caractere de nova linha. Se um parâmetro estiver faltando ele assume automaticamente o valor padrão. Com a exceção do caractere de abertura **#** e do valor seguindo **R**, os parâmetros da *Option Line* podem aparecer em qualquer ordem. Um exemplo de *Option Line* válida é:

KHz Y RI R 100

Por convenção, nomes de arquivos Touchstone sempre usam a extensão ***.snp***, onde **n** é o número de portas de rede do dispositivo sendo descrito. Por exemplo, um arquivo Touchstone contendo os parâmetros de rede de um dispositivo com 2 portas se chamaria **nome_do_arquivo.s2p**.

Embora em sua versão atual a IECM ainda não use para nenhum propósito os dados contidos nos modelos Touchstone, no futuro esses dados serão usados para possibilitar que as vistas geradas pela ferramenta sejam usadas em simulações do componente e, por esse motivo, a IECM já possui um módulo dedicado exclusivamente à importação e análise desse tipo específico de modelo.

Caso um modelo Touchstone do componente esteja sendo importado através da seção *4) Specify S-Parameter Model View to be imported* da GUI, com a correspondente ativação do botão ***Enable import***, um *Parser* implementado na função *ifxIecmCheckTouchstone()* é usado para verificar o formato e validade do arquivo. Este *Parser* realiza as seguintes análises com o arquivo Touchstone:

- Verifica se a extensão do arquivo está correta. A extensão deve ser sempre ***.snp***, onde **n** é o número de pinos descritos na *Component Specification* do componente.
- Localiza a *Option Line* dentro do arquivo e obtém as entradas de cada um de seus campos.
- Verifica se cada entrada da *Option Line* é uma entrada válida e se não é uma entrada que define um campo que já foi definido por outra entrada da *Option Line*.
- Imprime todos os erros em um arquivo de log chamado *TouchstoneLog.txt*, o qual é salvo no diretório de trabalho da célula.

Assim como para os modelos Spectre, embora a IECM já seja capaz de importar modelos Touchstone e checar por erros nas informações contidas nos mesmos, a ferramenta ainda não é capaz de utilizar tais informações em sua versão atual, mas em versões futuras elas serão utilizadas para a geração de outros tipos de vistas do componente.

3.3 – A Geração das Vistas do Componente

3.3.1 – Geração do Símbolo

Arquivos:

ifxIecmCreateSymbol.il

ifxIecmFigs.il

Funções:

ifxIecmCreateSymbol()

ifxIecmAddElements()

ifxIecmCheckCV()

ifxIecmHeaderFig()

ifxIecmStructPropertyFig()

ifxIecmGroupFig()

ifxIecmPinFigL()

ifxIecmPinFigR()

ifxIecmSupplyFigUp()

ifxIecmSupplyFigBottom()

ifxIecmDummyFig()

ifxIecmCheckbBox()

O processo de geração da vista do símbolo do componente segue vários estágios desde a importação da *Component Specification* até a geração do símbolo propriamente dito. A Figura 21 mostra uma vista geral simplificada destes estágios.

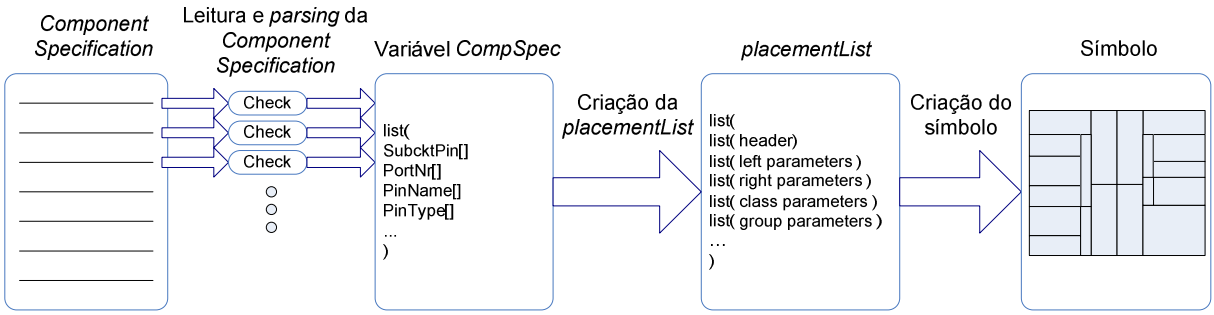


Figura 21 - Ilustração dos estágios da criação do símbolo do componente.

Finalizado o estágio de leitura e *parsing* da *Component Specification* importada, resta aos módulos do programa responsáveis pela geração das vistas da célula do componente, a utilização das

informações contidas na variável *CompSpec* para a criação da variável *placementList* e posterior criação do símbolo. O processo de criação do símbolo se inicia quando a função *ifxIecmMain()* (já explicada na seção 3.1.3) chama a função *ifxIecmCreateSymbol()*, para que esta controle todo o processo de criação dos elementos que compõem o símbolo, chamando para isso as funções responsáveis pela execução de cada etapa desse processo.

A Figura 22 ilustra a posição dos diferentes elementos do símbolo e os pontos utilizados como origem para o posicionamento dos pinos em cada um dos lados do símbolo, assim como a direção que esse posicionamento segue.

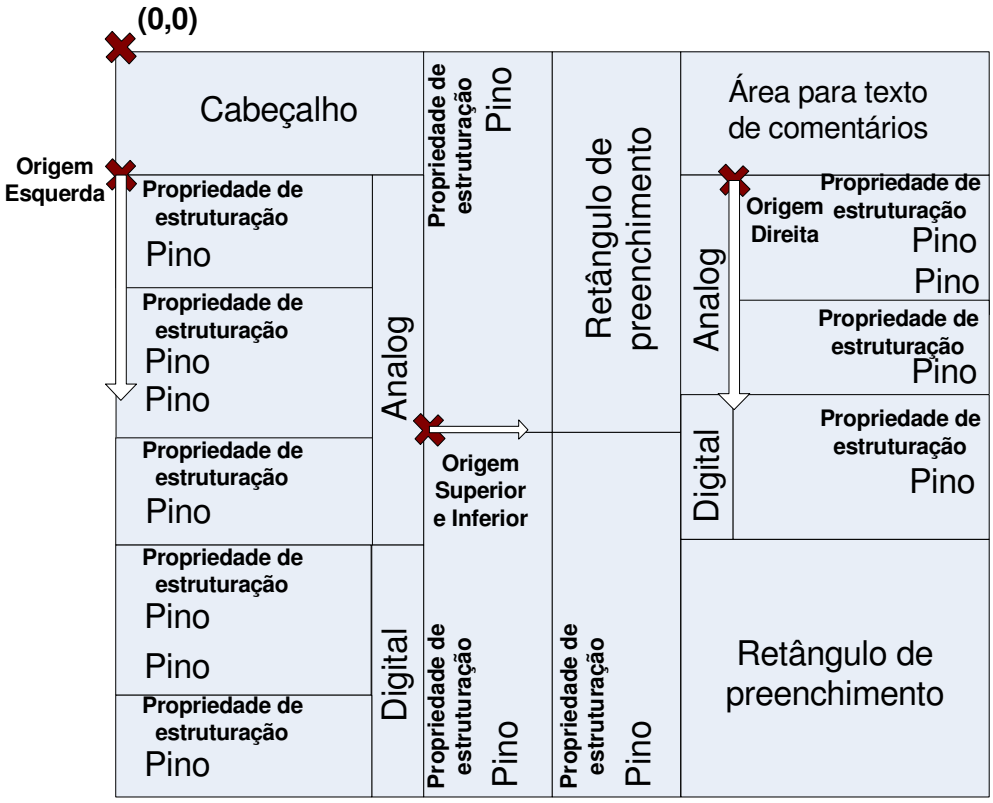


Figura 22 - Ilustração da posição dos elementos do símbolo.

A estrutura, as dimensões e o aspecto final do símbolo dependem principalmente do número e características dos pinos do componente, os quais são colocados logo após o cabeçalho no símbolo. Cabe mencionar, que além dos pinos simples, pode-se ter também barramentos compostos por múltiplos pinos nos componentes descritos pela *Component Specification*, mas como as figuras e propriedades dos barramentos são exatamente as mesmas que as usadas para os pinos simples na criação do símbolo, pode-se com efeito considerar os barramentos como pinos simples em todas as etapas de criação da vista do símbolo do componente e, por praticidade, todos os pinos simples e barramentos são referenciados simplesmente como pinos durante as explicações desse trabalho.

Os pinos do símbolo são agrupados de acordo com a propriedade de estruturação selecionada pelo usuário através do botão ***Structured by***, localizado na parte inferior da GUI. A propriedade de

estruturação pode ser qualquer uma das seguintes colunas de propriedades de pinos presentes na *Component Specification*: *SigClass*, *PinDescription*, *SymPinPos*, *XtSevClass* e *PinType*. Dessa maneira, se por exemplo a coluna de propriedade *SigClass* é selecionada como a propriedade de estruturação, todos os pinos que tenham a mesma entrada na coluna de propriedade *SigClass* na *Component Specification* serão agrupados em um retângulo em comum, enquanto pinos com *SigClass* diferentes serão colocados em retângulos separados no símbolo. O rótulo **Propriedade de estruturação**, visto na ilustração da Figura 22, é substituído pela propriedade do grupo de pinos em específico, o qual no caso de um símbolo estruturado de acordo com *SigClass*, por exemplo, pode ser SIG_V_IN, LOGIC_IN ou qualquer outra classe de sinais a qual um grupo de pinos pertença. Através da estruturação do símbolo de acordo com propriedades específicas dos pinos, o símbolo do componente pode apresentar *layouts* que atendam melhor os objetivos e preferências do usuário da IECM e que ao mesmo tempo otimizem as dimensões finais do símbolo para que o mesmo seja o mais compacto possível.

Os pinos localizados nos lados superior e inferior do símbolo não passam pelo processo de agrupamento que os pinos dos lados esquerdo e direito passam, já que os lados superior e inferior são reservados aos pinos de alimentação do componente, e por esse motivo, foi decidido pelos responsáveis pelo projeto que seria mais vantajoso ter-se um retângulo individual para cada pino destes lados, visando ter-se uma melhor vizualização dos mesmos. Além disso, o rótulo dos retângulos dos pinos dos lados superior e inferior do símbolo sempre mostram a propriedade *SigClass* dos pinos, independentemente da propriedade de estruturação selecionada pelo usuário na GUI.

Além da propriedade de estruturação escolhida pelo usuário, quatro propriedades dos pinos determinam o aspecto final que as regiões dos pinos terão:

- **Classe de sinal**: dependendo da classe de sinal definida na coluna *SigClass* na *Component Specification*, cada pino irá pertencer ou ao grupo analógico ou ao grupo digital. A indicação de quais pinos são analógicos e quais são digitais no símbolo é feita através dos retângulos verticais com os rótulos **Analog** e **Digital**, posicionados ao lado dos retângulos dos pinos, como pode-se ver na Figura 19. A definição de quais classes de sinais pertencem ao grupo analógico e quais ao grupo digital, é feita através dos campos *Analog* e *Digital* da variável *ifxIecmDataEntryFormatSpec*, dentro do arquivo *ifxIecmDataEntryFormatSpecification.il*, como pode-se ver no apêndice C.
- **Lado no símbolo**: a coluna *SymPinPos* na *Component Specification* serve justamente ao propósito de definir para o programa da IECM de que lado do símbolo cada pino deve ser posicionado: esquerdo, direito, superior ou inferior.
- **Tipo do pino**: cada pino pode ser do tipo entrada(*input*), saída(*output*) ou entrada/saída(*input/output*). Os tipos dos pinos colocados no símbolo corresponderão ao especificado na coluna *PinType* na *Component Specification* e possuem as seguintes diferenças visuais no símbolo para que possam ser distinguidos uns dos outros: pinos de

entrada de sinais possuem sua ponta virada para o símbolo, pinos de saída possuem sua ponta virada para fora do símbolo e pinos de entrada/saída possuem duas pontas.

- **Nome do pino**: juntamente com a figura do pino, o programa cria também um rótulo com o nome do pino, que corresponde à entrada da coluna *PinName* na *Component Specification*.

Para que o símbolo final possua uma aparência simétrica e retangular, retângulos são criados para preencher espaços vazios nos lados do símbolo que, por possuírem menos pinos, forem menores do que o lado oposto.

O programa também cria um campo para que o projetista possa inserir comentários de texto ou informações adicionais sobre o componente no símbolo gerado pela IECM.

Os processos de criação da *placementList* e dos elementos do símbolo são explicados a seguir.

Criação da *placementList*

Caso o usuário tenha selecionado na GUI a opção de gerar o símbolo, a ferramenta IECM irá utilizar as informações da *Component Specification*, as quais foram analisadas, organizadas e por fim armazenadas na variável *CompSpec*, para a criação de uma lista dos elementos a serem criados para compor o símbolo do componente. Tal lista é armazenada numa variável que recebe o nome *placementList* no código do programa e todas as coordenadas e informações necessárias para a criação de cada um dos elementos do símbolo são armazenadas dentro dela (no caso da criação de um pino por exemplo, consiste nas coordenadas, tipo, classe de sinal e lado do símbolo do pino), para que possam posteriormente serem passadas, juntamente com o endereço da vista onde o símbolo deve ser gerado, como argumento para a função *ifxIecmAddElements()*.

A função *ifxIecmAddElements()* é por sua vez a responsável por chamar a função apropriada para a criação de cada um dos elementos do símbolo. Todas as funções responsáveis pela criação dos vários elementos que compõem o símbolo encontram-se implementadas no arquivo *ifxIecmFigs.il*. Os elementos utilizados para compor o símbolo do componente, juntamente com suas descrições e funções responsáveis por suas criações são mostrados na Tabela 2.

Cabe dizer, que a mesma figura é utilizada para a criação dos pinos dos quatro lados do símbolo, apenas girando-se a figura do pino o número de graus correspondente para que se respeite a convenção de ter pinos de entrada com sua ponta virada para o interior do símbolo e pinos de saída com a ponta virada para fora.

Tabela 2: Elementos usados para compor o símbolo e funções responsáveis por suas criações.

Elemento	Função	Descrição
Cabeçalho	ifxIecmHeaderFig()	Cria os três campos que compõem o cabeçalho no canto superior esquerdo do símbolo. Os três campos indicam o nome do componente, a biblioteca e a célula de destino da vista do símbolo.
Pino do lado esquerdo	ifxIecmPinFigL()	Cria um pino no lado esquerdo do símbolo.
Pino do lado direito	ifxIecmPinFigR()	Cria um pino no lado direito do símbolo.
Pino de alimentação superior	ifxIecmSupplyFigUp()	Cria um retângulo com um pino de alimentação na parte superior do símbolo.
Pino de alimentação inferior	ifxIecmSupplyFigBottom()	Cria um retângulo com um pino de alimentação na parte inferior do símbolo.
Propriedade de Estruturação	ifxIecmStructPropertyFig()	Cria um retângulo para agrupar junto os pinos do lado esquerdo e direito que possuem a mesma propriedade de estruturação.
Grupo	ifxIecmGroupFig()	Cria um retângulo ao lado dos pinos do lado esquerdo ou direito do símbolo para indicar quais pinos são analógicos e quais são digitais.
Retângulo de preenchimento	ifxIecmDummyFig()	Cria um retângulo para preencher espaços vazios no símbolo.

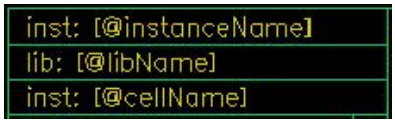
Para definir as dimensões e proporções dos diferentes elementos do símbolo, a variável global *ifxIecmGlobalVar* é definida no arquivo *ifxIecmDataEntryFormatSpecification.il*. Esta variável contém os campos mostrados na Tabela 3.

Tabela 3: Campos da variável *ifxIecmGlobalVar*, usada para definir as dimensões do símbolo.

Campo	Descrição
xWidth	Base de unidade para distâncias na direção **x**.
yWidth	Base de unidade para distâncias na direção **y**.
dim->wHeader	Dimensão **x** para os campos do cabeçalho.
dim->lHeader	Dimensão **y** para os campos do cabeçalho.
dim->wPin	Dimensão **x** dos campos dos pinos dos lados esquerdo e direito do símbolo.
dim->lPin	Dimensão **y** dos campos dos pinos dos lados esquerdo e direito do símbolo.
dim->wSupply	Dimensão **x** dos campos dos pinos de alimentação nos lados superior e inferior do símbolo.
dim->wGroup	Dimensão **x** do retângulo que indica os grupos analógico e digital de pinos.

As figuras criadas para cada um dos elementos que compõem o símbolo podem ser vistas na Tabela 4.

Tabela 4: Figuras criadas pelo programa para cada tipo de elemento do símbolo.

Elemento	Figura
Cabeçalho	
Propriedade de estruturação	
Grupo	
Pino	
Retângulo de preenchimento	

Após receber a *placementList* e o endereço onde a vista do símbolo deve ser criada, a função *ifxIecmAddElements()* chama a função responsável pela criação de cada tipo de elemento armazenado na *placementList*, para criar na vista do símbolo, um por um, os elementos contidos na *placementList.*

Criação dos Elementos do Símbolo

A criação dos elementos do símbolo segue a sequência: cabeçalho, área dos pinos, retângulos de preenchimento, área para texto de comentários.

Cabeçalho

O primeiro elemento a ser posicionado na vista do símbolo é o cabeçalho com seus três campos: nome do componente, biblioteca e célula de destino da vista do símbolo. Ele serve ao propósito de indicar ao projetista a qual componente o símbolo pertence e onde exatamente a vista do símbolo do componente está armazenada. O canto superior esquerdo do cabeçalho é posicionado na coordenada (0,0).

Área dos pinos

Cada pino criado no símbolo consiste na instância do pino, o rótulo com o nome do pino e um rótulo adicional, que serve ao propósito de possibilitar a conectividade dos pinos do símbolo em simulações do componente no ambiente Cadence. Este último rótulo apresenta escrito apenas ***cdsTerm("nome_do_pino")***, sendo que "***nome_do_pino"*** é substituído pelo programa pelo nome do pino em específico, e é ele que permite que o ambiente de simulação de circuitos do Cadence reconheça este pino do componente e permita assim conectá-lo aos pinos de outros componentes durante simulações.

O ponto denominado **Origem Esquerda** é usado para o controle das coordenadas de posicionamente dos pinos do lado esquerdo do símbolo e fica localizado no canto inferior esquerdo do cabeçalho. Partindo deste ponto e deslocando-se de cima para baixo, o programa aloca um por um os pinos do lado esquerdo, começando pelos pinos pertencentes ao grupo analógico e terminando com os pinos pertencentes ao grupo digital. Um retângulo é criado para cada grupo de pinos que possuírem a mesma propriedade de estruturação, juntamente com um rótulo indicando esta propriedade.

A largura dos retângulos onde os pinos são alocados dos lados esquerdo e direito do símbolo é a mesma para efeito de simetria e é estabelecida pelo programa proporcionalmente ao comprimento do nome mais longo dos pinos destes lados do símbolo.

O ponto denominado **Origem Direita**, usado para o controle das coordenadas de posicionamento dos pinos do lado direito do símbolo, é calculado com base no número de pinos nos

lados superior e inferior do símbolo, já que estes determinam a distância entre os pinos do lado esquerdo e os pinos do lado direito. Partindo da **Origem Direita** e deslocando-se de cima para baixo, o posicionamento dos pinos do lado direito do símbolo segue o mesmo princípio usado para os pinos do lado esquerdo.

Com base na menor coordenada **y** dos pinos dos lados esquerdo e direito do símbolo e no tamanho do nome mais comprido dos pinos dos lados superior e inferior, o programa determina o comprimento dos retângulos dos pinos superiores e inferiores do símbolo, os quais tem sempre o mesmo comprimento por motivos de simetria do símbolo.

A **Origem Superior e Inferior**, usada para o controle das coordenadas de posicionamento dos pinos superiores e inferiores, é estabelecida à direita da região dos pinos do lado esquerdo, exatamente na metade da altura do símbolo.

Partindo da **Origem Superior e Inferior,** cada pino do lado superior do símbolo é posicionado seguindo da esquerda para a direita.

Também partindo da **Origem Superior e Inferior,** cada pino do lado inferior do símbolo é posicionado seguindo da esquerda para a direita.

Retângulos de preenchimento

Terminada a alocação do cabeçalho e de todos os pinos, o programa verifica em quais lados se faz necessária a adição de retângulos de preenchimento. Estes retângulos são utilizados para preencher espaços vazios em um ou mais lados do símbolo, fazendo com que o símbolo tenha um formato retangular e com isso um visual melhor. No caso do componente não possuir pinos em um de seus lados, um retângulo de preenchimento é criado ocupando este lado inteiro do símbolo.

Área para texto de comentários

O último elemento a ser criado é uma área no canto superior direito do símbolo para a inserção posterior de um texto descritivo ou de comentários sobre o componente pelo projetista ou qualquer outro usuário do símbolo gerado pela IECM.

Finalmente, terminada a criação de todos os elementos, o símbolo criado pela ferramenta *Import of External Component Models* encontra-se completo e possui uma aparência final semelhante à da Figura 23.

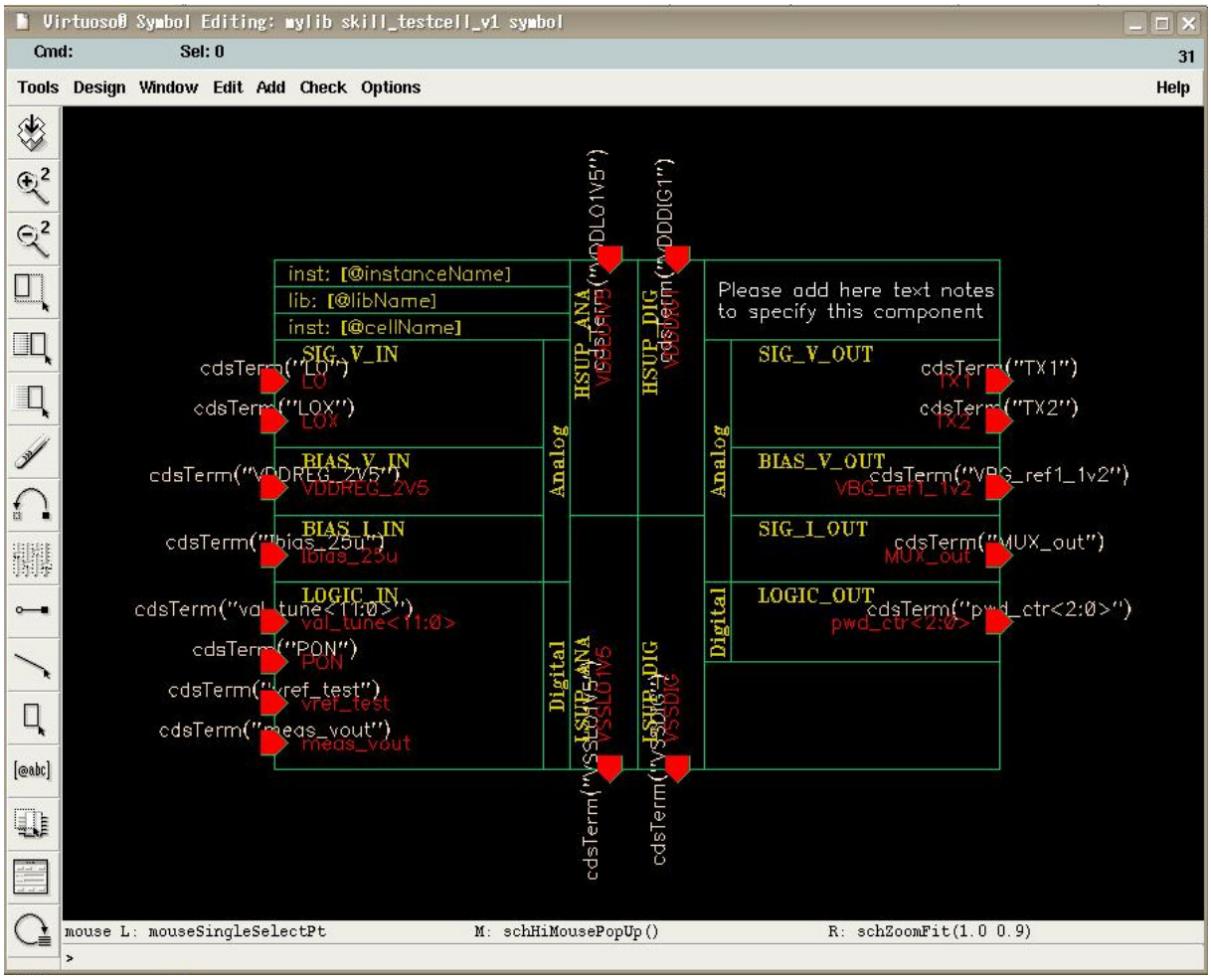


Figura 23 - Exemplo de um símbolo gerado pela ferramenta IECM.

Através deste exemplo, pode-se distinguir claramente cada elemento do símbolo. No canto superior esquerdo tem-se o cabeçalho com seus três campos, enquanto no canto superior direito tem-se o retângulo reservado à inserção de notas sobre o componente. Os quatro lados do símbolo são reservados aos pinos do componente, que juntamente com suas instâncias, apresentam também o rótulo com o nome do pino e o rótulo ***cdsTerm***, que permite a conectividade do pino em simulações do componente.

Analisando-se as áreas dos pinos nos quatro lados do símbolo, pode-se fazer algumas observações. Em primeiro lugar, nota-se que para este componente em específico, os pinos dos lados esquerdo e superior do componente são todos de entrada, já que todos possuem sua ponta virada para o centro do símbolo. Da mesma maneira, tem-se que todos os pinos dos lados direito e inferior são de saída, já que suas pontas apontam para fora do símbolo. Este componente não possuí nenhum pino de entrada/saída, o que nota-se pela ausência de figuras de pinos com duas pontas.

Este símbolo, em específico, usou como propriedade de estruturação a classe de sinal dos pinos (coluna de propriedade *SigClass*), como pode-se perceber ao se observar os rótulos dos retângulos onde os pinos estão alocados, que mostram SIG_V_IN, BIAS_V_IN, LOGIC_OUT e outras entradas pertencentes exclusivamente à coluna de propriedade de pinos *SigClass*. É fácil

vizualizar o agrupamento feito com os pinos que possuem mesma classe de sinal, como acontece com os pinos **LO** e **LOX**, que por possuírem ambos a classe de sinal **SIG_V_IN**, foram postos no mesmo retângulo. À direita dos retângulos dos pinos da esquerda e à esquerda dos retângulos dos pinos da direita, tem-se os dois retângulos verticais que servem ao propósito de indicar quais pinos são analógicos e quais são digitais.

Apenas um retângulo de preenchimento foi necessário para este símbolo em específico, o qual foi colocado no canto direito inferior para preencher o espaço vazio ali presente, devido ao fato do lado direito do componente possuir três pinos a menos que o lado esquerdo.

Mais explicações e exemplos de símbolos gerados pela IECM são dados no capítulo 4, onde os resultados dos testes realizados com a ferramenta são apresentados.

3.3.2 – Geração do Esquemático de Pinos

Arquivo:

ifxIecmCreatePinSchematic.il

Função:

ifxIecmCreatePinSchematic()

Caso tenha sido selecionado na GUI pelo usuário, a IECM irá gerar também um simples esquemático de pinos do componente, usando para isso as informações extraídas da *Component Specification*, de maneira semelhante ao que foi feito para se criar o símbolo. Este esquemático de pinos consiste basicamente numa vista com todos os pinos enfileirados, um abaixo do outro. A Figura 24 mostra a aparência final de uma vista de esquemático de pinos típica gerada pela ferramenta IECM.

Pela pequena variedade de elementos usados em sua construção e por não haver a necessidade da implementação de uma lógica complexa de posicionamento dos pinos (como a necessária na geração do símbolo), o processo de geração da vista do esquemático de pinos do componente é muito mais simples que o processo de geração da vista do símbolo. De fato, o programa simplesmente posiciona os pinos, um abaixo do outro, criando para cada pino a instância do pino, um rótulo com o nome do mesmo e um fio que serve de terminal para permitir a conectividade deste pino com outros componentes em simulações feitas no ambiente Cadence. Este terminal é uma alternativa ao rótulo ***cdsTerm("nome_do_pino")***, usado para permitir a conectividade dos pinos da vista do símbolo, na qual não era conveniente se ter um monte de fios saindo dos pinos para fora do símbolo gerado.

Para permitir que o projetista diferencie o tipo de cada pino no esquemático, o programa posiciona pinos de entrada e pinos de entrada/saída no lado esquerdo e pinos de saída no lado direito, lembrando que a diferenciação entre pinos de entrada e pinos de entrada/saída é feita através da diferença entre a figura usada para a instância deles, sendo que a figura dos pinos de entrada possuem apenas uma ponta e a dos pinos de entrada/saída possui duas.

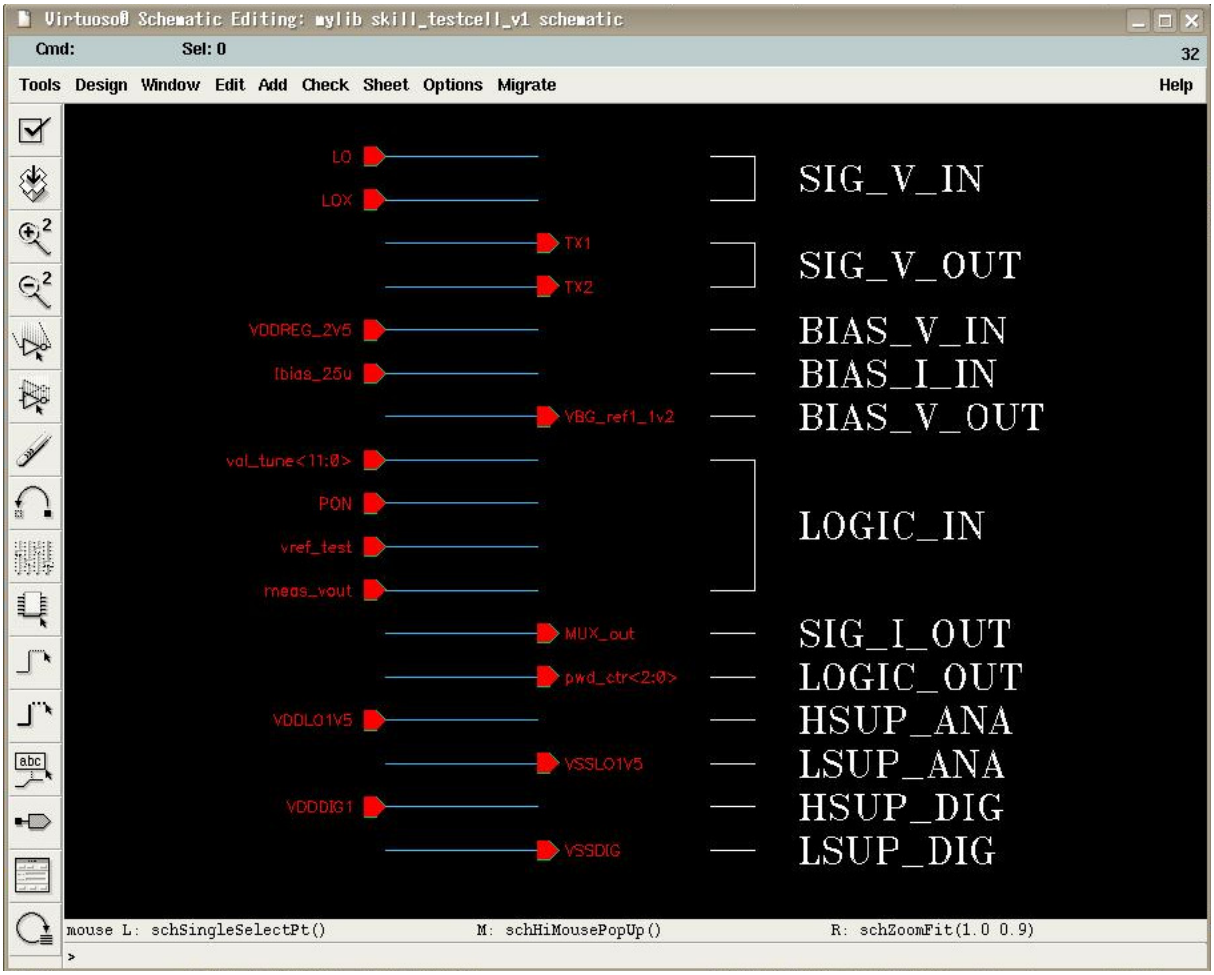


Figura 24 – Exemplo de um esquemático de pinos gerado pela ferramenta IECM.

Da mesma maneira que foi feito para a vista do símbolo, os pinos são agrupados de acordo com a propriedade de estruturação selecionada pelo usuário na GUI, com a diferença de que neste caso até mesmo os pinos que pertenceriam a lados diferentes do símbolo são agrupados juntos caso possuam a mesma propriedade. Como na vista do esquemático de pinos não existem retângulos para agrupar pinos com a mesma propriedade de estruturação, são criados colchetes à direita dos pinos, juntamente com rótulos que indicam a propriedade dos pinos sendo agrupados por cada colchete.

Mais explicações e exemplos de esquemáticos de pinos gerados pela IECM são dados a seguir no capítulo 4.

4. RESULTADOS DOS TESTES COM A IECM

No que segue, serão apresentados os resultados dos testes realizados com a ferramenta *Import of External Component Models*, visando comprovar sua capacidade de gerar as vistas de símbolos e esquemáticos de pinos de componentes através da importação de seus modelos de *Component Specification*. Para a execução dos testes foram utilizadas aproximadamente 10 *Component Specifications* de componentes reais, as quais foram especialmente fornecidas por uma das empresas participantes do projeto, para que as funcionalidades da IECM pudessem ser testadas diante de tarefas semelhantes às para as quais ela se destinará quando estiver sendo usada em ambiente industrial. No entanto, embora a IECM tenha importado os modelos reais e gerado suas vistas com sucesso, por apresentarem conteúdos confidencias, os resultados dos testes realizados com tais modelos de componentes reais não serão apresentados neste trabalho.

Os resultados mostrados neste documento foram obtidos através de testes utilizando uma *Component Specification* criada especialmente para testar a eficiência da IECM e a qual representa um componente fictício denominado *Test_Component*. O fato dessa *Component Specification* não representar um componente real não tira no entanto a validade dos resultados obtidos, já que ela foi completamente baseada em *Component Specifications* de componentes reais. A *Component Specification* fictícia, criada para a realização dos testes, é mostrada na Figura 25.

É importante salientar que as entradas das colunas de propriedades da seção DEFINIÇÃO DE PROPRIEDADES DOS PINOS foram escolhidas especificamente visando testar a capacidade da IECM de gerar vistas com as características mais diversas possíveis. Portanto, quaisquer inconsistências físicas nas entradas da *Component Specification* devem ser desconsideradas, já que as entradas foram escolhidas buscando-se apenas estabelecer vários tipos de combinações entre elas, sem se preocupar se estas representariam características físicas reais. Um exemplo disso é o pino **Ix**, cuja classe de sinal está definida na coluna *SigClass* como sendo **CLK_IN** (uma classe de sinal típica de pinos de entrada) e no entanto é definido na coluna *PinType* como um pino de saída (**output**). Essa combinação entre **CLK_IN** e **output** do pino **Ix** foi feita para testar como a ferramenta lidaria com a tarefa de criar dois pinos de saída do grupo digital no mesmo lado do símbolo (pinos **Ix** e **CLAMP_ON**). Da mesma maneira, outras combinações foram feitas entre as propriedades dos pinos para que se pudesse testar a eficiência da ferramenta, embora tais combinações possam ser conceitualmente erradas.

```
* BEGIN IMPORT HEADER
* ==========================================
* BEGIN COMMENTS
* This is a Component Specification Example.
*
* END COMMENTS
* ==========================================
* BEGIN COMPONENT DESCRIPTION
* DESIGNATION          , Test_Component
* CDBA PATH            , U:\mprado\IECM\test
* TARGET LIB           , test_library
* TARGET CELL          , test_cell
* NUMBER OF PINS       , 18
* COMPONENT TYPE       , wiring_chip
* LOWER RIGHT COORDINATE , 20:-15
* END COMPONENT DESCRIPTION
* ==========================================
* BEGIN DEFINE PIN PROPERTY COLUMNES
* SubcktPin, PortNr, PinName, PinType, BusWidth, IntNetName, SigClass,
XtSevClass, SymPinPos, HsupDomain, LsupDomain, LoadModel, NomDcBias,
Xpos, Ypos, PinDescription
* END DEFINE PIN PROPERTY COLUMNES
* ==========================================
* BEGIN PIN PROPERTY DEFINITION
* I , 1 , I , input , 1 , I_IntName , SIG_V_IN , --- , left , VDD1 , VSS1
, (V_DC (Vval=2.5)) , --- , --- , --- , Pin Description String
* Ix , 2 , Ix , output , 1 , Ix_IntName , CLK_IN , --- , left , VDD1 ,
VSS1 , (V_DC (Vval=2.5)) , --- , --- , ---, ---
* PON , 3 , PON , inputOutput , 1 , PON_IntName , LOGIC_IN , --- , left ,
VDD1 , VSS1 , (V_DC (Vval=2.5)) , --- , --- , ---, ---
* CLAMP_ON , 4 , CLAMP_ON , output , 1 , CLAMP_ON_IntName , HSUP_DIG , --
- , left , VDD1 , VSS1 , (V_DC (Vval=2.5)) , --- , --- , ---, ---
* ibias , 5 , ibias , input , 1 , ibias_IntName , BIAS_I_IN , --- ,
bottom , VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , --- , ---
* bottom , 6 , bottom , output , 1 , bottom_IntName , BIAS_I_IN , --- ,
bottom , VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , --- , ---
* bottom2 , 7 , bottom2 , inputOutput , 1 , bottom2_IntName , BIAS_I_IN ,
--- , bottom , VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , --- , ---
* top  , 8 , top , inputOutput , 1 , top_IntName , BIAS_I_IN , --- , top
, VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , --- , ---
* top2 , 9 , top2 , input , 1 , top2_IntName , BIAS_I_IN , --- , top ,
VDD1 , VSS1 , (I_DC (Ival=400u)) , --- , --- , --- , ---
* O , 10 , O , output , 1 , O_IntName , CRT_OUT , --- , right , VDD1 ,
VSS1 , (R (Rval=1k)) , --- , --- , --- , ---
* Ox , 11 , Ox , inputOutput , 1 , Ox_IntName , SIG_V_OUT , --- , right ,
VDD1 , VSS1 , (R (Rval=1k)) , --- , --- , --- , ---
* R_trim<4:0> , (12;13;14;15;16) , R_trim<4:0> , input , 5 ,
R_trim_IntName<4:0> , SIG_V_IN , --- , right , VDD1 , VSS1 , (V_DC
(Vval=2.5)) , --- , --- , --- , ---
* GAIN , 17 , GAIN , input , 1 , GAIN_IntName , SIG_V_IN , --- , right ,
VDD1 , VSS1 , (V_DC (Vval=2.5)) , --- , --- , --- , ---
* GAIN2 , 18 , GAIN2 , input , 1 , GAIN2_IntName , SIG_V_OUT , --- ,
right , VDD1 , VSS1 , (V_DC (Vval=2.5)) , --- , --- , --- , ---
* END PIN PROPERTY DEFINITION
* ==========================================
* END IMPORT HEADER
```

Figura 25 - *Component Specification* usada nos testes.

Seguindo o processo de importação de modelos para a IECM já explicado anteriormente, as informações da *Component Specification* são separadas, organizadas e armazenadas na variável *CompSpec*, para que se possa usá-las posteriormente na geração das vistas. Para uma melhor vizualização de como as informações do componente encontram-se organizadas após a importação da *Component Specification*, mostra-se abaixo o conteúdo dos vetores que compõem a variável *CompSpec* e guardam as informações contidas na *Component Specification* prontas para serem usadas pelo programa.

SubcktPin[] = [I, Ix, PON, CLAMP_ON, ibias, bottom, bottom2, top, top2, O, Ox, R_trim<4:0>, GAIN, GAIN2]

PortNr[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, [12, 13, 14, 15, 16], 17, 18]

PinName[] = [I, Ix, PON, CLAMP_ON, ibias, bottom, bottom2, top, top2, O, Ox, R_trim<4:0>, GAIN, GAIN2]

PinType[] = [input, output, inputOutput, output, input, output, inputOutput, inputOutput, input, output, inputOutput, input, input, input]

BusWidth[] = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1]

IntNetName[] = [I_IntName, Ix_IntName, PON_IntName, CLAMP_ON_IntName, ibias_IntName, bottom_IntName, bottom2_IntName, top_IntName, top2_IntName, O_IntName, Ox_IntName, R_trim_IntName <4:0>, GAIN_IntName, GAIN2_IntName]

SigClass[] = [SIG_V_IN, CLK_IN, LOGIC_IN, HSUP_DIG, BIAS_I_IN, BIAS_I_IN, BIAS_I_IN, BIAS_I_IN, BIAS_I_IN, CRT_OUT, SIG_V_OUT, SIG_V_IN, SIG_V_IN, SIG_V_OUT]

XtSevClass[] = [---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---]

SymPinPos[] = [left, left, left, left, bottom, bottom, bottom, top, top, right, right, right, right, right]

HSupDomain[] = [VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1, VDD1]

LSupDomain[] = [VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1, VSS1]

LoadModel[] = [[V_DC, Vval, 2.5], [V_DC, Vval, 2.5], [V_DC, Vval, 2.5], [V_DC, Vval, 2.5], [I_DC, Ival, 400u], [I_DC, Ival, 400u], [I_DC, Ival, 400u], [I_DC, Ival, 400u], [I_DC, Ival, 400u], [R, Rval, 1k], [R, Rval, 1k], [V_DC, Vval, 2.5], [V_DC, Vval, 2.5], [V_DC, Vval, 2.5]]

NomDcBias[] = [---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---]

Xpos[] = [---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---]

Ypos[] = [---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---]

PinDescription[] = [Pin Description String , ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---, ---]

Ao se pressionar o botão ***Run Import*** na GUI, o programa prossegue gerando as vistas do símbolo e do esquemático de pinos do componente *Test_Component* através dos passos já descritos neste documento.

4.1 – A Geração do Símbolo.

Usando as informações contidas em *CompSpec*, o programa cria a *placementList* dos elementos a serem criados para compor o símbolo e, em seguida, chama uma à uma as funções responsáveis pela criação destes. Terminado esse processo, o símbolo de *Test_Component* encontra-se finalmente completo.

Para se mostrar as diferenças entre os símbolos gerados para cada propriedade de estruturação escolhida no botão ***Structured by*** da GUI, foram gerados três símbolos diferentes de *Test_Component*, o primeiro utilizando a coluna *SigClass* como propriedade de estruturação, o segundo com *PinType* e o último com *PinPos*.

A Figura 26 mostra a vista do símbolo do componente fictício gerada pela IECM utilizando *SigClass* como propriedade de estruturação.

No símbolo gerado nota-se claramente a divisão dos pinos de cada lado do componente entre os grupos analógico e digital, de acordo com a classe de sinal dos pinos. Assim, analisando-se o lado esquerdo, temos que o pino **I** de classe de sinal **SIG_V_IN** foi posto no grupo analógico, já que, segundo o estabelecido em *ifxIecmDataEntryFormatSpecification.il*, temos que **SIG_V_IN** é uma classe de sinal pertencente ao grupo analógico. Da mesma maneira, os pinos **Ix**, **PON** e **CLAMP_ON**, que possuem as classes **CLK_IN**, **LOGIC_IN** e **HSUP_DIG** respectivamente, foram postos juntos no grupo digital, já que suas classes foram declaradas como pertencentes deste grupo em *ifxIecmDataEntryFormatSpecification.il*.

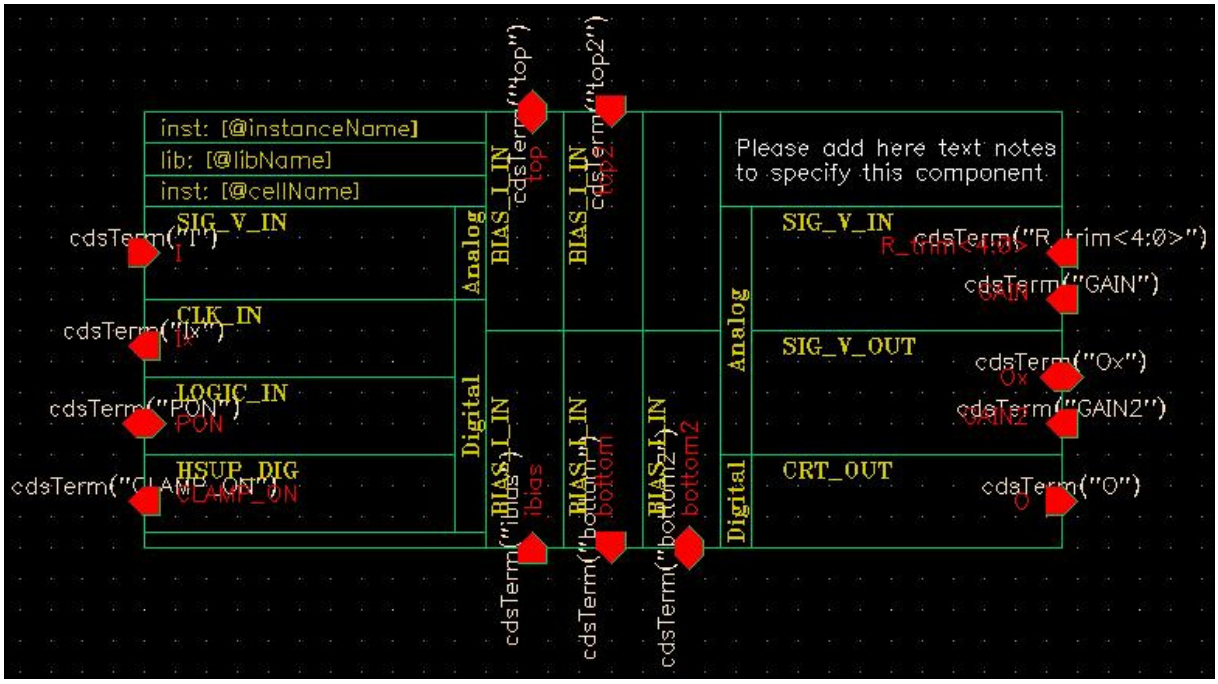


Figura 26 - Símbolo do componente *Test_Component* estruturado por *SigClass.*

Além disso, ao se comparar os lados esquerdo e direito do símbolo, percebe-se claramente o agrupamento feito com os pinos de mesma classe de sinal, já que nesse caso *SigClass* foi escolhida como a propriedade de estruturação do símbolo. Assim, tem-se do lado direito os pinos **R_trim<4:0>** e **GAIN** agrupados juntos por possuírem ambos a classe de sinal **SIG_V_IN** e os pinos **Ox** e **GAIN2** juntos por terem a classe de sinal **SIG_V_OUT**.

Cabe lembrar que tanto o agrupamento dos pinos entre os grupos analógico e digital quanto a organização deles de acordo com a propriedade de estruturação não são realizados nos pinos nas partes superior e inferior do símbolo por motivos de estética e visando uma melhor vizualização dos pinos pertencentes a esses lados, os quais são geralmente reservados para os pinos de alimentação do componente.

Variando-se a propriedade de estruturação das vistas escolhida na GUI, obtém-se diferentes disposições para os pinos e layouts para o símbolo. Na Figura 27 pode-se ver o símbolo criado pela IECM quando a coluna *PinType* é escolhida como propriedade de estruturação, e sua diferença em relação ao símbolo visto na Figura 26.

Nesse caso, embora a divisão de pinos entre os grupos analógico e digital mantenha-se a mesma, nota-se que os pinos foram organizados pelo programa de maneira a agrupar os pinos de mesmo tipo: entrada, saída ou entrada/saída. Com isso tem-se que **Gain2** e **Ox**, que no símbolo estruturado de acordo com *SigClass* haviam sido postos juntos, agora pertencem a retângulos distintos já que **GAIN2** é do tipo **entrada** e **Ox** do tipo **entrada/saída**. Nota-se também que a mudança da propriedade de estruturação escolhida não apenas alterou a disposição dos pinos, como também os

rótulos dos retângulos responsáveis por mostrar a propriedade dos pinos agrupados, que agora mostram o tipo dos pinos pertencentes a cada retângulo.

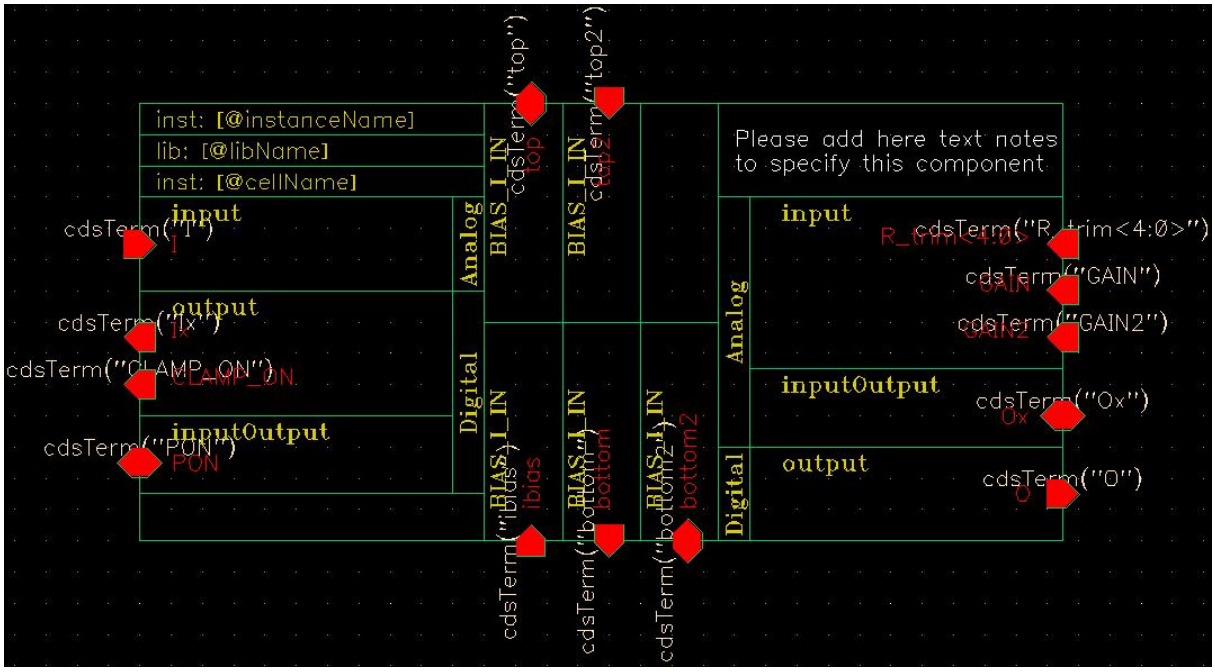


Figura 27 - Símbolo do componente *Test_Component* estruturado por *PinType*.

A vantagem dessa configuração faz-se evidente, já que através dela o projetista pode ver instantaneamente quais pinos atuam como entrada de sinais no componente, quais atuam como saída e quais permitem a entrada e a saída de sinais. Cabe lembrar que, como já dito, embora os pinos dos lados esquerdo e direito sejam organizados de acordo com a propriedade *PinType*, os pinos dos lados superior e inferior do componente mantém sua configuração de acordo com *SigClass*, além de serem colocados em retângulos individuais no símbolo.

Como último exemplo dos efeitos que se têm na estrutura do símbolo ao se alterar a propriedade de estruturação das vistas, tem-se na Figura 28 o símbolo criado pela IECM quando se escolhe a coluna *SymPinPos* como propriedade de estruturação.

Esse é um caso especial de configuração do símbolo pois aqui o único critério de separação dos pinos é o grupo ao qual eles pertencem: **analógico** ou **digital**. Isso porque o programa usa a coluna *SymPinPos* como propriedade de estruturação, agrupando os pinos pertencentes ao lado esquerdo do símbolo juntos e fazendo o mesmo com os do lado direito. Assim tem-se que, com efeito, os pinos acabam por serem divididos apenas entre os retângulos que definem os grupos analógico e digital de pinos, o que pode ser vantajoso ao projetista quando este desejar conectar os pinos do símbolo do componente aos pinos de outros componentes para a construção de circuitos maiores.

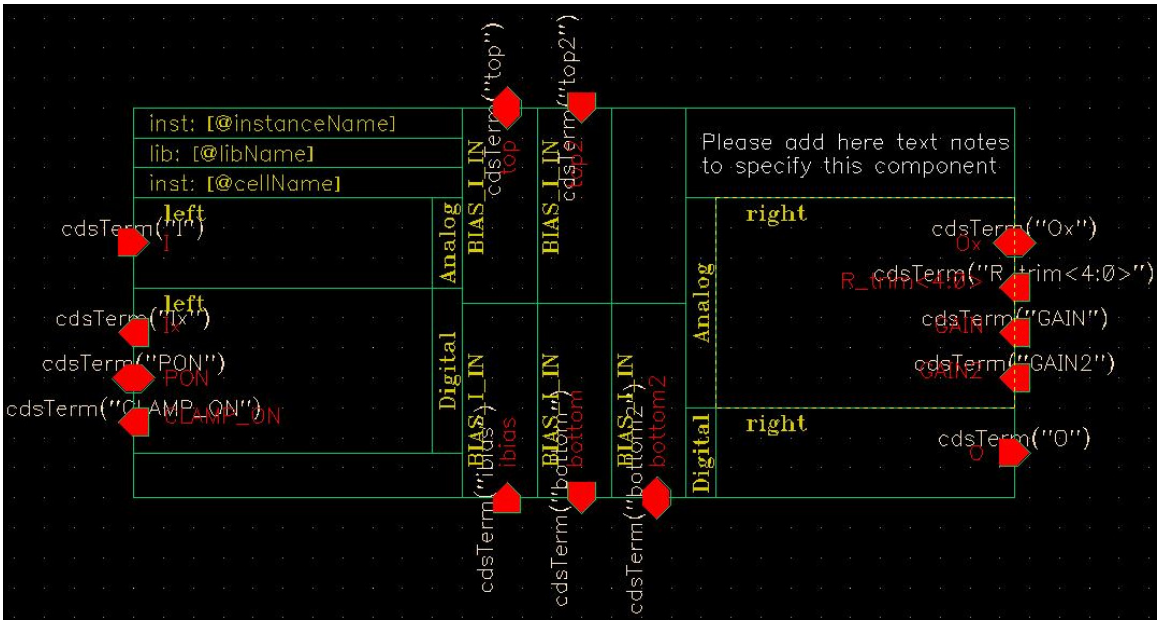


Figura 28 - Símbolo do componente *Test_Component* estruturado por *SymPinPos.*

4.2 – A Geração do Esquemático de Pinos.

Terminada a geração do símbolo, a ferramenta IECM prossegue gerando uma vista do tipo esquemático contendo apenas os pinos do componente. Para o primeiro teste de geração da vista do esquemático de pinos de *Test_Component*, foi escolhida a propriedade de estruturação *SigClass* na GUI e a vista correpondente gerada pela ferramenta IECM é mostrada na Figura 29.

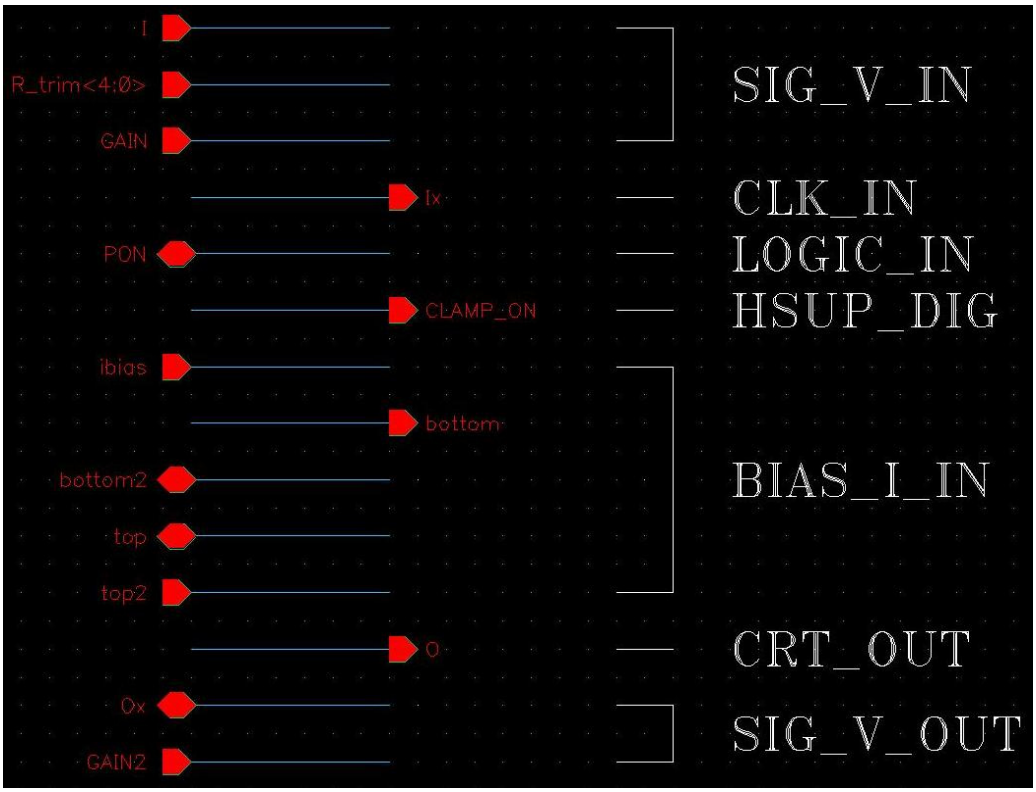


Figura 29 - Esquemático de pinos do componente *Test_Component* estruturado por *SigClass.*

Como pode-se ver, para cada pino definido na *Component Specification*, o programa criou a instância do pino e um terminal para que seja possível conectá-lo ao terminal de outros componentes em simulações. Os pinos definidos como dos tipos entrada ou entrada/saída foram posicionados pelo programa à esquerda no esquemático, enquanto os pinos que são do tipo saída foram posicionados à direita, possibilitando dessa maneira ao projetista diferenciar visualmente o tipo dos pinos presentes no esquemático. Assim, ao se examinar o esquemático, nota-se facilmente que os pinos **I**, **R_trim<4:0>**, **GAIN**, **ibias**, **top2** e **GAIN2** são pinos de **entrada**, os pinos **PON, bottom2, top** e **Ox** são pinos de **entrada/saída** e os pinos **Ix**, **CLAMP_ON**, **bottom** e **O** são pinos de **saída**.

Semelhantemente ao que foi feito na estruturação do símbolo, o programa agrupou todos os pinos que possuem a mesma propriedade de estruturação (neste caso *SigClass*), com a diferença de que para realizar esse agrupamento, ao invés de criar retângulos para agrupar os pinos como foi feito no símbolo, criou-se colchetes à direita dos pinos seguidos de rótulos indicando a propriedade de cada grupo de pinos em específico. Vale lembrar que no agrupamento de pinos feito para o esquemático de pinos nenhuma distinção é feita entre os pinos definidos como pertencentes a lados diferentes do símbolo. Exemplos disso são os pinos **I**, **R_trim<4:0>** e **GAIN**, que apesar de possuírem mesma classe de sinal (SIG_V_IN) ficaram separados na vista do símbolo, já que **I** pertence ao lado esquerdo do símbolo enquanto **R_trim<4:0>** e **GAIN** pertencem ao lado direito, mas contudo foram posicionados juntos na vista do esquemático de pinos.

Mais uma vez, para demonstrar o efeito obtido com a mudança da propriedade de estruturação escolhida, as Figuras 30 e 31 mostram os esquemáticos de pinos de *Test_Component* estruturados de acordo com as colunas *PinType* e *SymPinPos* respectivamente. O esquemático de pinos estruturado de acordo com *PinType* permite vizualizar claramente a distinção que o programa faz entre os tipos de pinos, posicionando os pinos de entrada e de entrada/saída do lado esquerdo do esquemático e os pinos de saída do lado direito. Dessa maneira, tem-se os pinos de entrada **I**, **ibias**, **top2**, **R_trim<4:0>**, **GAIN** e **GAIN2** dispostos do lado esquerdo, assim como os pinos de entrada/saída **PON**, **bottom2**, **top** e **Ox**, enquanto os pinos **Ix**, **CLAMP_ON**, **bottom** e **O** foram postos do lado direito do esquemático. Esquemáticos de pinos estruturados segundo *PinType* possuem a clara vantagem de apresentar organizadamente ao projetista quais pinos recebem sinais de entrada, quais fornecem sinais de saída e quais aceitam tanto a entrada quanto a saída de sinais.

Por sua vez, esquemáticos de pinos estruturados segundo *SymPinPos* mostram-se interessantes por indicarem a que lado do componente cada pino pertence. Assim, tem-se que **I**, **Ix**, **PON** e **CLAMP_ON** pertencem ao **lado esquerdo**, **ibias**, **bottom** e **bottom2** pertencem ao **lado inferior**, **top** e **top2** pertencem ao **lado superior** e **O**, **Ox**, **R_trim<4:0>**, **GAIN** e **GAIN2** pertencem ao **lado direito**.

Terminada a geração das três variações de vistas do símbolo e das três variações de vistas do esquemático de pinos, foram encerrados os testes da IECM com *Test_Component*.

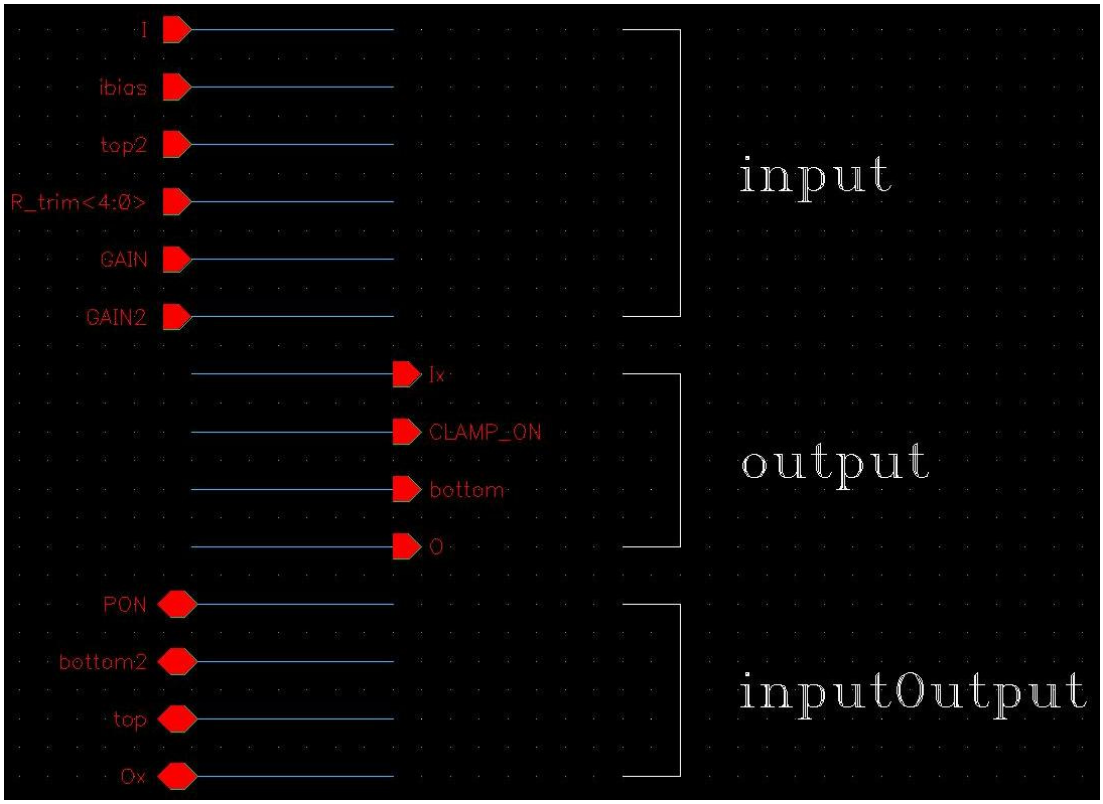


Figura 30 - Esquemático de pinos do componente *Test_Component* estruturado por *PinType.*

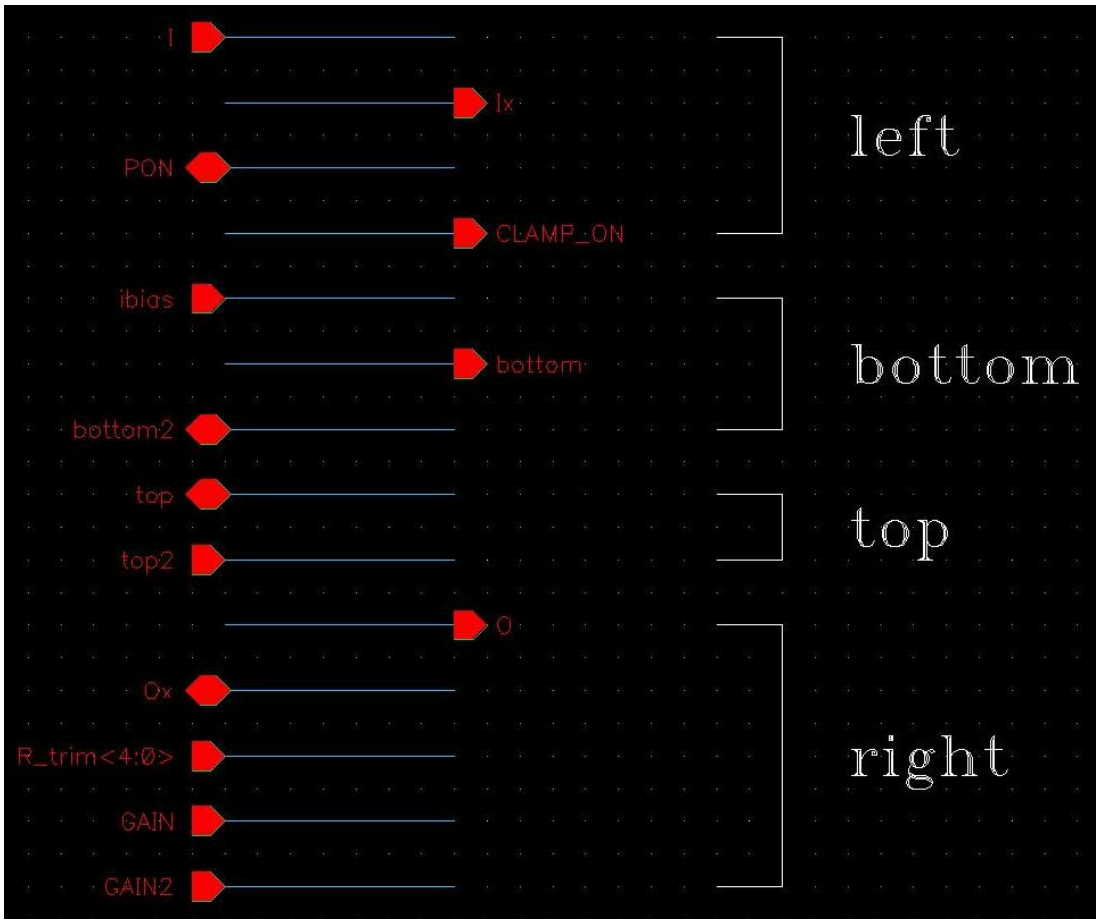


Figura 31 - Esquemático de pinos do componente *Test_Component* estruturado por *SymPinPos.*

5. CONCLUSÕES

A elaboração deste projeto concentrou-se no desenvolvimento de uma ferramenta que possuísse, dentre outras funcionalidades, a capacidade de importar arquivos de modelos descritivos de componentes eletrônicos e usar as informações contidas nestes modelos para gerar automaticamente símbolos e esquemáticos do componente dentro do Ambiente de Desenvolvimento Cadence. Essa ferramenta será no futuro a responsável pela geração prática e rápida de grandes bibliotecas contendo símbolos e esquemáticos de componentes, que possam ser usados em simulações e associados entre si para formarem sistemas maiores e complexos, e assim facilitar e otimizar consideravelmente o trabalho dos projetistas de circuitos, que de outra maneira teriam que desenvolver manualmente todos os símbolos e esquemáticos dos componentes num processo longo e laborioso.

Para que este projeto resultasse na ferramenta *Import of External Component Models* (IECM), produto final deste projeto, foi primeiro necessário todo um processo de aprendizagem por parte do aluno de como trabalhar dentro do Ambiente de Desenvolvimento Cadence e de como criar programas dentro deste, utilizando sua linguagem de programação SKILL. A seguir, sob orientação de seu supervisor e de outro parceiro de projeto, coube ao aluno desenvolver conceitualmente de que maneira cada módulo componente da ferramenta poderia ser implementado, para que se pudesse obter todas as funcionalidades desejadas para a ferramenta final. Terminada a etapa de planejamento, iniciou-se então a implementação em código SKILL de cada um dos módulos da IECM, processo que ocupou a maior parte do tempo dedicado ao projeto.

O processo de importação dos modelos de componentes batizados de *Component Specifications* envolve um conjunto complexo de módulos responsáveis por cada uma das etapas, desde a leitura e checagem de erros das informações contidas nos modelos até a organização destas informações de forma estruturada, para que estas possam enfim serem usadas na geração das vistas do componente. Da mesma maneira, outro conjunto de módulos é responsável especificamente por usar essas informações para a criação do símbolo e do esquemático de pinos do componente. No corpo deste documento, tentou-se dar uma explicação sucinta e clara do funcionamento básico de cada um desses módulos, deixando-se de lado os pormenores do código do programa onde os módulos foram implementados.

Cabe comentar que boa parte do código implementado teve de ser revisto e alterado diversas vezes durante a criação da ferramenta, devido a mudanças nas especificações da ferramenta e na estrutura das *Component Specifications* por parte dos líderes de projeto, até que se chegasse à versão atual da IECM, apresentada neste documento.

Uma vez que as funcionalidades da IECM já encontravam-se implementadas, iniciou-se o período de testes da ferramenta, para os quais o aluno dispôs de modelos de componentes reais fornecidos por uma das empresas parceiras do projeto, para que a ferramenta pudesse assim ser testada

em aplicações exatamente iguais às para as quais a mesma será utilizada quando for liberada para uso em ambiente industrial. Durante essa fase de testes foram utilizados aproximadamente 10 *Component Specifications* de componentes desenvolvidos pela empresa parceira de projeto, e para todos estes a ferramenta foi capaz de gerar corretamente as vistas do símbolo e do esquemático de pinos dos componentes. Apesar disso, os resultados de tais testes não puderam ser incluídos no corpo deste documento, já que as *Component Specifications* utilizadas apresentam informações confidenciais sobre componentes que ainda não foram divulgados pela empresa. Por este motivo, o aluno criou uma nova *Component Specification* com base nas *Component Specifications* fornecidas pela empresa, e os resultados mostrados neste documento referem-se justamente aos testes realizados com esta. Enfatizou-se nos testes realizados, entre outras coisas, analisar as diferenças entre as vistas geradas para diferentes propriedades de estruturação escolhidas na GUI da IECM pelo usuário, ressaltando-se eventuais vantagens de cada configuração em relação às outras.

Cabe frisar que a ferramenta criada pelo aluno durante este trabalho ainda continuará sendo desenvolvida pelos próximos meses para que sejam incluídas novas funcionalidades, como a possibilidade de importar modelos Titan, criar esquemáticos mais detalhados do componente, e outros recursos que aperfeiçoem ainda mais a versão atual da IECM. Contudo, conforme comprovado pelos testes feitos pelo aluno e posteriormente corroborado pelos parceiros do projeto, a versão atual da ferramenta IECM já é capaz de realizar todas as funções pré-estabelecidas pelos líderes do projeto para que esta possa ser incorporada no ambiente industrial.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] TUTORIAL CADENCE DESIGN ENVIRONMENT. Disponível em <<http://www.ece.nmsu.edu/vlsi/cadence/CADENCE%20Manual.pdf >>. Acessado em março de 2010.

[2] Cadence Design Systems, Inc. SKILL Language User Guide. United States of America, 2005.

[3] A Quick Tour of SKILL Programming. Disponível em <<www.cadence.com/Community/CSSharedFiles/forums/storage/28/10475/CadScriptingLanguages_skill.pdf>>. Acessado em março de 2010.

[4] SPICE User's Manual. Disponível em <<http://newton.ex.ac.uk/teaching/cdhw/Electronics2/userguide/sec2.html>>. Acessado em maio de 2010.

[5] Creating a SPICE Subcircuit. Disponível em <<http://www.5spice.com/Subckts.htm>>. Acessado em maio de 2010.

[6] Touchstone File Format Specification. Disponível em <<www.vhdl.org/pub/ibis/adhoc/interconnect/touchstone_spec2_draft2.pdf>>. Acessado em novembro de 2009.

Apêndice A – Lista dos arquivos e funções da ferramenta IECM

Arquivos da IECM	Funções
ifxIecmGUI.il	ifxIecmImportExternalModelsGUI() ifxIecmImportSpecFileButtonCB()
ifxIecmBrowse.il	ifxIecmBrowse() ifxIecmFileBoxSingleClickCB() ifxIecmDirBoxSingleClickCB() ifxIecmDirBoxDoubleClickCB()
ifxIecmCallModules.il	ifxIecmMain() ifxIecmInitWorkSpace()
ifxIecmDataEntryFormatSpecification.il	ifxIecmInitializeVar()
ifxIecmCreateComponentDB.il	ifxIecmReadComponentSpec() ifxIecmParseComponentSpec() ifxIecmCheckColumnsSection() ifxIecmCheckLength() ifxIecmCheckUnique() ifxIecmCheckBus() ifxIecmCheckNumbers() ifxIecmCheckOptions() ifxIecmCheckStrings() ifxIecmCheckBinary() ifxIecmCreateModelStructure() ifxIecmSpecialSyntax()
ifxIecmCreateSymbol.il	ifxIecmCreateSymbol() ifxIecmCheckCV() ifxIecmAddElements()
ifxIecmFigs.il	ifxIecmHeaderFig() ifxIecmStructPropertyFig() ifxIecmGroupFig() ifxIecmPinFigL() ifxIecmPinFigR() ifxIecmSupplyFigUp() ifxIecmSupplyFigBottom() ifxIecmDummyFig() ifxIecmCheckbBox()
ifxIecmCreatePinSchematic.il	ifxIecmCreatePinSchematic()
ifxIecmCheckModelValidity.il	ifxIecmCheckSpectre() ifxIecmCheckTouchstone()

Apêndice B – Arquivo *ifxIecmBrowser.il*

Esse apêndice contém o código de programa no qual foi implementado o Browser utilizado para importar arquivos de modelos para a ferramenta IECM.

```
;#########################################################################
;
; SKILL functions for Import of External Component Models
;                          - File Browser -
;
; Author: Marcelo Prado, Fraunhofer IIS
; uwe.knoechel@eas.iis.fraunhofer.de
;#########################################################################

;#######################################################
; This functions creates a Browser Form and displays it.
; The Browser has one Box Field to select the Directory
; and another to select the File inside the directory.
; A Filter button allows the user to choose
; the initial Directory of the Browser.
;#######################################################
procedure( ifxIecmBrowser()
	let( ( fileDir dirContent elementsToDisplay listOfDirs fileBoxField dirBoxField dirField fileField filterLabel
dirLabel fileLabel openDirButton filterButton fields result )

	;*** If the initial directory of the browser is " ~ " adds " / " to its path
	fileDir = pwd()
	if( fileDir == "~"
	   fileDir = "~/"
	) ;if

	;*** Gets the folders and files in the directory
	dirContent = ls(fileDir)

	;#######################################################
	; Creates the list of folders and the list of files which
	; will be displayed in the two Box Fields of the browser.
	;#######################################################
	foreach( elementOfDir dirContent
		if( isFile(sprintf(nil "%s/%s" fileDir elementOfDir) ) then
			elementsToDisplay = cons(elementOfDir elementsToDisplay)
		else
			listOfDirs = cons(sprintf(nil "%s/" elementOfDir) listOfDirs)
		) ;if
	) ;foreach
	;*** The elements are sorted by alphabetic order
	elementsToDisplay = sort(elementsToDisplay 'alphalessp)
	listOfDirs = sort(listOfDirs 'alphalessp)

	;*** Files Box Field ***
	fileBoxField = hiCreateListBoxField(
		?name `fileBoxField
		?choices reverse(elementsToDisplay)
		?changeCB "ifxIecmFileBoxSingleClickCB()"
		?doubleClickCB "hiFormDone(browserWindow)"
	)

	;*** Directories Box Field ***
	dirBoxField = hiCreateListBoxField(
			?name `dirBoxField
			?choices listOfDirs
			?changeCB "ifxIecmDirBoxSingleClickCB()"
```

```
                        ?doubleClickCB "ifxIecmDirBoxDoubleClickCB()"
)

;*** String field for the Filter
dirField = hiCreateStringField(
            ?name `dirField
            ?value fileDir
)
;*** String field to show the full path of the Selected File
fileField = hiCreateStringField(
            ?name `fileField
)

;*** Label of the Filter
filterLabel = hiCreateLabel(
                        ?name 'filterLabel
                        ?labelText "Filter"
                        )
;*** Label of the Directories Box Field
dirLabel = hiCreateLabel(
                        ?name 'dirLabel
                        ?labelText "Directories"
                        )
;*** Label of the Files Box Field
fileLabel = hiCreateLabel(
                        ?name 'fileLabel
                        ?labelText "Files"
                        )
;*** Open Directory Button - Alternative to the Double-Click in the Directories Box Field
openDirButton = hiCreateButton(
                        ?name 'openDir
                        ?buttonText "Open Dir"
                        ?callback "ifxIecmDirBoxDoubleClickCB( )"
                        )
;*** Filter Button
filterButton = hiCreateButton(
                        ?name 'filterButton
                        ?buttonText "Filter"
                        ?callback "ifxIecmDirBoxDoubleClickCB( t )"
                        )

;*** Fields of the Browser ***
fields = list(
            list( filterLabel      10:0       50:30                )
            list( dirField                    10:20     480:0     0          )
            list( dirLabel                    13:50     80:30                )
            list( fileLabel                   163:50    50:30                )
            list( dirBoxField    10:70      150:240 )
            list( fileBoxField   160:70   350:240  )
            list( fileField      10:310    480:0     0          )
            list( openDirButton               45:345    70:20                )
            list( filterButton   220:345 50:20                  )
            );list

;###############
; Browser window
;###############
browserWindow = hiCreateAppForm(
            ?name 'browserWindow
            ?formTitle "Open File"
            ?fields    fields
            ?buttonLayout 'OKCancel
            ?minSize 500:410
);hiCreateAppForm

result = hiDisplayForm( browserWindow )
```

```
            ;*** The Full Path of the Selected File is returned ***
            if( result
               browserWindow->fileField->value
            );if

            );let
);procedure
;##########################################################################


;#############################################################
; Callback called when a file is selected in the File Box Field
; It changes the value in the File Path string field.
;#############################################################
procedure( ifxIecmFileBoxSingleClickCB()
   let( ( currentDir selectedDir subDirs numOfSubDirs fileDir it selectedFile )

   currentDir = browserWindow->dirField->value
   selectedDir = car( browserWindow->dirBoxField->value )
   if( selectedDir
      then
               case( selectedDir
                        ( "../"
                                    ;*** Gets the path from the directory one level above from the current directory
                                    subDirs = parseString( currentDir "/" )
                                    numOfSubDirs = length( subDirs )
                                    fileDir = ""
                                    it = 0
                                    while( it < numOfSubDirs-1
                                          fileDir = sprintf( nil "%s%s/" fileDir nth( it subDirs ) )
                                          it++
                                    ) ;while
                        )
                        ( "./"
                                    ;*** Gets the path from the current directory
                                    fileDir = currentDir
                        )
                        ( t
                                    ;*** Gets the path from the directory selected in the Directories Box Field
                                    fileDir = sprintf( nil "%s%s" currentDir selectedDir )
                        )
               ) ;case
      else
               ;*** If no directory is selected gets the current directory
               fileDir = currentDir
   ) ;if

   ;*** Updates the File string field
   selectedFile = car( browserWindow->fileBoxField->value )
   browserWindow->fileField->value = sprintf( nil "%s%s" fileDir selectedFile )

   ) ;let
) ;procedure
;#############################################################


;#################################################################################
; Callback called when a directory is clicked one time in the Directories Box Field
; Shows all the files in this directory in the File Box Field.
;#################################################################################
procedure( ifxIecmDirBoxSingleClickCB()
   let( ( fileDir dirContent elementsToDisplay )

   ;*** Adds the selected subfolder name to the current directory path
   fileDir = sprintf( nil "%s%s" browserWindow->dirField->value car( browserWindow->dirBoxField->value ) )

   ;*** Gets the folders and files in the directory
```

```
  dirContent = ls( fileDir )
  ;*** Creates the list of files which will be displayed in the File Box Field ***
  foreach( elementOfDir dirContent
        if( isFile( sprintf(nil "%s/%s" fileDir elementOfDir) )
              elementsToDisplay = cons(elementOfDir elementsToDisplay)
           ) ;if
  ) ;foreach

  ;*** Updating the choices in the Box Fields
  elementsToDisplay = sort( elementsToDisplay 'alphalessp )
  browserWindow->fileBoxField->choices = elementsToDisplay

  ) ;let
) ;procedure
;##########################################################################################


;##########################################################################################
; Callback called when a directory is clicked two times in the Directories Box Field
; or if the Filter Button is pressed.
; Changes the folder displayed in the Directories Box Field.
;##########################################################################################
procedure( ifxIecmDirBoxDoubleClickCB( @optional filter )
  let(( fileDir selectedDir currentDir subDirs numOfSubDirs it dirContent elementsToDisplay listOfDirs )

  ;*** Gets the current directory
  fileDir = browserWindow->dirField->value

  ;#########################################################
  ; If the callback was not called through the Filter Button
  ;#########################################################
  if( !filter
     then
           ;*** Gets the selected directory
           selectedDir = car( browserWindow->dirBoxField->value )


;##########################################################################################
           ; When choosing "." ".." it is necessary to control how the folder field is updated
           ; since "." and ".." should not appear in the folder path

;##########################################################################################
           case( selectedDir
                 ( "./"
                        fileDir =  browserWindow->dirField->value
                 )
                 ( "../"
                        currentDir = browserWindow->dirField->value
                        subDirs = parseString( currentDir "/" )
                        numOfSubDirs = length(subDirs)
                        fileDir = ""
                        it = 0
                        ;*** Rebuilds the folder path, excluding the last subdirectory
                        while( it < numOfSubDirs-1
                              fileDir = sprintf( nil "%s%s/" fileDir nth( it subDirs ) )
                              it++
                        ) ;while
                 )
                 ( t
                        ;*** Adds the new selected folder to the path of the previous folder path
                        fileDir = sprintf( nil "%s%s" fileDir selectedDir )
                 )
           ) ;case
  ) ;if

  if( !isFile( fileDir )
     then
```

```
            ;*** Gets the folders and files in the directory
            dirContent = ls( fileDir )

            ;#######################################################
            ; Creates the list of folders and the list of files which
            ; will be displayed in the two Box Fields of the browser.
            ;#######################################################
            foreach( elementOfDir dirContent
                        if( isFile( sprintf(nil "%s/%s" fileDir elementOfDir) )
                           then
                                 elementsToDisplay = cons(elementOfDir elementsToDisplay)
                              else
                                 listOfDirs = cons(sprintf(nil "%s/" elementOfDir) listOfDirs)
                        ) ;if
            ) ;foreach

            ;*** Updating the choices in the two Box Fields
            elementsToDisplay = sort( elementsToDisplay 'alphalessp )
            listOfDirs = sort( listOfDirs 'alphalessp )
            browserWindow->fileBoxField->choices = elementsToDisplay
            browserWindow->dirBoxField->choices = listOfDirs

            ;*** Final value of the Folder String Field
            browserWindow->dirField->value = fileDir
   );if

   );let
);procedure
;##########################################################################
```

Apêndice C – Arquivo *ifxIecmDataEntryFormatSpecification.il*

Esse apêndice contém o código de programa no qual o formato e sintaxe exigidos para a *Component Specification* são definidos. É também nesse arquivo que se encontram definidas as diversas dimensões dos elementos que compõem o símbolo.

```
;#################################################################################
##
;
;                    SKILL functions for Import of External Component Models
; - Defines the required Syntax and Format for the entries from the Component Specification -
;                    - Defines also the dimensions of the elements of the symbol -
;
; Author: Marcelo Prado, Fraunhofer IIS
; uwe.knoechel@eas.iis.fraunhofer.de
;#################################################################################
##
;; ---------------------------------------------------------------
;;  HINTS:
;;  -- Each pin has one row.
;;  -- Each row can have beneath the already defined standard columnes arbitrary additional columnes
;      for further informations like position coordinates, text tags, etc.
;;
;;  Extract the Values by: ifxIecmDataEntryFormatSpec->PinNr->??
;;                    ifxIecmDataEntryFormatSpec->PinNr->number
;;                    ifxIecmDataEntryFormatSpec->LoadModel->RCser->??
;;                    ifxIecmDataEntryFormatSpec->LoadModel->RCser->Cval
;;                    ifxIecmDataEntryFormatSpec->LoadModel->R_LC_ser->Lval
;; Liste Löschen:
;; ifxIecmDataEntryFormatSpec  ='unbound
;; ifxIecmDataEntryFormatSpec->LoadModel = 'unbound
;;
;; load( strcat( ifxIecmWORK "/ifxIecmDataEntryFormatSpecification.il"))
;; ---------------------------------------------------------------
;;
;; This section defines the required list structure to read in the
;; standardised pin property specification, which is part of the
;; Component Specification. This list structure has the name
;; ifxIecmTemplate_PinPropSpec.
;;
;; ==============================================
;; BEGIN DEFINE PIN SPECIFICATION LIST STRUCTURE
ifxIecmCompSpec =
list(nil
    'SubcktPin      list("p_1")
    'PortNr         list("1")
    'PinName        list("IO_pin1")
    'PinType        list("inputOutput")
    'BusWidth       list("1")
    'IntNetName     list("IO_IntNetPin1")
    'SigClass       list("SIG_V_IN")
    'XtSevClass     list("RX_high_MHz")
    'SymPinPos      list("left")
    'HsupDomain     list("VDD1")
    'LsupDomain     list("VSS1")
    'LoadModel      list(nil)
    'NomDcBias      list(nil)
    'Xpos           list(nil)
    'Ypos           list(nil)
    )

ifxIecmTemplate_PinPropSpec =
list(nil
```

```
    'SubcktPin      list("p_1")
    'PortNr         list("1")
    'PinName        list("IO_pin1")
    'PinType        list("inputOutput")
    'BusWidth       list("1")
    'IntNetName     list("IO_IntNetPin1")
    'SigClass       list("SIG_V_IN")
    'XtSevClass     list("RX_high_MHz")
    'SymPinPos      list("left")
    'HsupDomain     list("VDD1")
    'LsupDomain     list("VSS1")
    'LoadModel      list(nil 'RCpar list(nil 'Rval "100k" 'Cval "15f"))
    'NomDcBias      list(nil 'V_DC  list(nil 'Vval "1.5"))
    'Xpos           list(nil)
    'Ypos           list(nil)
    )
;; END DEFINE PIN SPECIFICATION LIST STRUCTURE
;; ==========================================
;;
;; This section defines the allowed entries for all columnes in the
;; Component's pin property specification. This is a list structure
;; called ifxIecmCompPinPropSpec
;;
;; ==========================================
;; BEGIN DATA ENTRY FORMAT SPECIFICATION
ifxIecmDataEntryFormatSpec =
list(nil
    'SubcktPin  list(nil 'string "[A-Za-z0-9_!<:>]")
    'PortNr     list(nil 'number "integer" 'low 1 'high 999)
    'PinName    list(nil 'string "[A-Za-z0-9_!<:>]")
    'PinType    list("input" "output" "inputOutput")
    'BusWidth   list(nil 'number "integer" 'low 1 'high 99)
    'IntNetName list("[- ]+" 'string "[A-Za-z0-9_!<:>]")
    'SigClass   list(
                        "[- ]+"
                        "SIG_V_IN"
                        "SIG_I_IN"
                        "SIG_V_OUT"
                        "SIG_I_OUT"
                        "BIAS_V_IN"
                        "BIAS_I_IN"
                        "BIAS_V_OUT"
                        "BIAS_I_OUT"
                        "LOGIC_IN"
                        "LOGIC_OUT"
                        "CLK_IN"
                        "CLK_OUT"
                        "CRT_IN"
                        "CRT_OUT"
                        "HSUP_ANA"
                        "LSUP_ANA"
                        "HSUP_DIG"
                        "LSUP_DIG"
                        )
    'XtSevClass list(
                "[- ]+"
                        "RX_high_kHz"
                        "RX_med_kHz"
                        "RX_low_kHz"
                        "RX_high_MHz"
                        "RX_med_MHz"
                        "RX_low_MHz"
                        "RX_high_GHz"
                        "RX_med_GHz"
                        "RX_low_GHz"
                        "TX_high_kHz"
                        "TX_med_kHz"
```

```
                        "TX_low_kHz"
                        "TX_high_MHz"
                        "TX_med_MHz"
                        "TX_low_MHz"
                        "TX_high_GHz"
                        "TX_med_GHz"
                        "TX_low_GHz"
                        "TRX_high_MHz"
                        "TRX_med_MHz"
                        "TRX_low_MHz"
                        "TRX_high_GHz"
                        "TRX_med_GHz"
                        "TRX_low_GHz"
                        )
    'SymPinPos  list("left" "right" "top" "bottom")
    'HsupDomain list( "[- ]+" "VDD1" "VDD2" "VDD3" "VDD4")
    'LsupDomain list( "[- ]+" "VSS1" "VSS2" "VSS3" "VSS4")
    'LoadModel  list("[- ]+"
                        'I_DC      list(nil 'Ival "real")
                        'V_DC      list(nil 'Vval "real")
                        'R         list(nil 'Rval "real")
                        'C         list(nil 'Cval "real")
                        'L         list(nil 'Lval "real")
                        'RCser     list(nil 'Rval "real" 'Cval "real")
                        'RCpar     list(nil 'Rval "real" 'Cval "real")
                        'RLser     list(nil 'Rval "real" 'Lval "real")
                        'RLpar     list(nil 'Rval "real" 'Lval "real")
                        'RLCser    list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'RLCpar    list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'C_RL_par  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'L_RC_par  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'R_LC_par  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'C_RL_ser  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'L_RC_ser  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        'R_LC_ser  list(nil 'Rval "real" 'Cval "real" 'Lval "real")
                        )
    'NomDcBias  list("[- ]+"
                'I_DC        list(nil 'Ival "real")
                'V_DC        list(nil 'Vval "real")
                'AC_out      list(nil 'Cval "real")
                'DIG_out     list(nil 'Pattern "binary")
                        )
    'Xpos list("[- ]+"
                         'number "real"
                        )
    'Ypos list("[- ]+"
                        'number "real"
                       )

;; END DATA ENTRY FORMAT SPECIFICATION
;; ==========================================
;;
;; This section sets the class groups
;;
;; ==========================================
;; BEGIN CLASS GROUP SPECIFICATION
    'Analog          list(
                       "SIG_V_IN"
                       "SIG_I_IN"
                       "SIG_V_OUT"
                       "SIG_I_OUT"
                       "BIAS_V_IN"
                       "BIAS_I_IN"
                       "BIAS_V_OUT"
                       "BIAS_I_OUT"
                       "HSUP_ANA"
                       "LSUP_ANA"
```

```
                        )
    'Digital            list(
                        "LOGIC_IN"
                        "LOGIC_OUT"
                        "CLK_IN"
                        "CLK_OUT"
                        "CRT_IN"
                        "CRT_OUT"
                        "HSUP_DIG"
                        "LSUP_DIG"
                        )
    'CompType     list(
                        "circuit_rf"
                        "circuit_analog"
                        "circuit_digital"
                        "circuit_block"
                        "circuit_supply"
                        "wiring_chip"
                        "wiring_package"
                        "wiring_pcb"
                        )
    'XT_Agressor list(
                        "TX_med_kHz"
                        "TX_low_kHz"
                        "TX_high_MHz"
                        "TX_med_MHz"
                        "TX_low_MHz"
                        "TX_high_GHz"
                        "TX_med_GHz"
                        "TX_low_GHz"
                        "TRX_high_MHz"
                        "TRX_med_MHz"
                        "TRX_low_MHz"
                        "TRX_high_GHz"
                        "TRX_med_GHz"
                        "TRX_low_GHz"
                        )
    'XT_Victime list(
                        "RX_high_kHz"
                        "RX_med_kHz"
                        "RX_low_kHz"
                        "RX_high_MHz"
                        "RX_med_MHz"
                        "RX_low_MHz"
                        "RX_high_GHz"
                        "RX_med_GHz"
                        "RX_low_GHz"
                        "TRX_high_MHz"
                        "TRX_med_MHz"
                        "TRX_low_MHz"
                        "TRX_high_GHz"
                        "TRX_med_GHz"
                        "TRX_low_GHz"
                        )
;; END CLASS GROUP SPECIFICATION
;; ==========================================
) ; end ifxIecmDataEntryFormatSpec

;#########################################################################################
#####
;It initializes the global var ifxIecmGlobalVar for the Import of External Component Models Tool
;#########################################################################################
#####
procedure( ifxIecmInitializeVar()
;           printf("--> ifxIecmInitializeVar <--\n")

            defstruct(          symbolDim
```

```
                    wHeader
                    lHeader
                    wPin
                    lPin
                    wSupply
                    wGroup
        ) ;defstruct

        defstruct(  GlobalVariable
                    ifxSubcktModelImportForm
        ) ;defstruct

        ifxIecmGlobalVar = make_GlobalVariable()

        ;*** Reference dimensions for the symbol elements
        ifxIecmGlobalVar->xWidth = 0.0625
        ifxIecmGlobalVar->yWidth = 0.0625

        ;*** Structure with all the dimensions used in the symbol
        ifxIecmGlobalVar->dim = make_symbolDim()

        ;########################################
        ;Defines the dimesions for the symbol elements
        ;the prefix w is for width - x dimension
        ;the prefix l is for lenght - y dimension
        ;########################################
        ifxIecmGlobalVar->dim->wHeader = 22*ifxIecmGlobalVar->xWidth
        ifxIecmGlobalVar->dim->lHeader = 2 *ifxIecmGlobalVar->yWidth
        ifxIecmGlobalVar->dim->wPin    = 20*ifxIecmGlobalVar->xWidth
        ifxIecmGlobalVar->dim->lPin    = 3 *ifxIecmGlobalVar->yWidth
        ifxIecmGlobalVar->dim->wSupply = 5 *ifxIecmGlobalVar->xWidth
        ifxIecmGlobalVar->dim->wGroup  = 2 *ifxIecmGlobalVar->xWidth

) ;procedure
```

Apêndice D – Carta do gerente do projeto no Instituto Fraunhofer

Esse apêndice contém a carta escrita por Uwe Knöchel, gerente do Grupo de Circuitos Analógicos e de Rádio-frequência do Instituto Fraunhofer para Circuitos Integrados, a qual não apenas comprova as atividades desenvolvidades durante o o período de trabalho do aluno no Instituto Fraunhofer, como também o sucesso do mesmo e o potencial para futura aplicação dos produtos finais do projeto em ambiente industrial.

Fraunhofer
IIS

Fraunhofer-Institute for Integrated Circuits IIS
Design Automation Division EAS

Director
Prof. Dr.-Ing. Heinz Gerhäuser
(executive)
Prof. Dr.-Ing. Günter Elst

Zeunerstraße 38
01069 Dresden
Germany

Uwe Knöchel
Group Manager AHF
Mixed-Signal Systems
Phone + 49 351 4640-748 | Fax -703
uwe.knoechel@eas.iis.fraunhofer.de
www.eas.iis.fraunhofer.de

Fraunhofer IIS Division EAS | Zeunerstraße 38 | 01069 Dresden

Dresden, December 30, 2009

To whom it may concern

Marcelo Loyo Prado has attended a 6-months traineeship from 01st July 2009 to 30th December 2009 at our institute. He worked on the development of tools which support the designers of integrated circuits. This included:

- Introduction to the Cadence Design Framework for the design of integrated circuits
- Getting experiences in Skill-Programming language
- Conception, programming and test of tools model and symbol generation
- Documentation of the developed tools

Although Marcelo was faced with the hard task to handle the complexity of the Cadence framework as well as the high demands on the developed tools, he learned very fast and with a minimum of guidance how to use the framework and how to write Skill programs. The tools that he developed exceeds the expectations of the project team and will be integrated in an industrial design flow. Beside this he provided a detailed documentation of the programs in excellent English language.

It was my pleasure to coach Marcelo during the last six month, because he consistently met and often exceeded his job requirements. He demonstrated outstanding enthusiasm, team working ability. Marcelo acted on its own initiative and was always cooperative. I'd like to thank him very much for his work and wish him all the best for his life and career.

Sincerely,

Uwe Knöchel
Group Manager
Analog and RF Circuits

Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., München
Executive Board of the Fraunhofer-Gesellschaft
Univ.-Prof. Dr.-Ing. habil. Prof. e.h. mult. Dr. h.c. mult. Hans-Jörg Bullinger, president
Prof. Dr. rer. nat. Ulrich Buller
Dr. rer. pol. Alfred Gossner
Prof. Dr. phil. Marion Schick

Cheques and transfers payable to
Deutsche Bank, München
Account 752193300 BLZ 700 700 10
IBAN DE86 7007 0010 0752 1933 00
BIC (SWIFT-Code) DEUTDEMM
V.A.T.Ident No DE129515865
Tax Number 143/215/20392