

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE FÍSICA
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA AMBIENTAL

**DESENVOLVIMENTO DE MÓDULO DE SOFTWARE
PARA SIMULAÇÃO DO FLUXO DE CALOR NO SOLO**

RAPHAEL PIRES FERREIRA

ORIENTADOR: PROF. DR. DENILTON CARLOS GAIO

Cuiabá, MT
Fevereiro - 2017

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE FÍSICA
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA AMBIENTAL

**DESENVOLVIMENTO DE MÓDULO DE SOFTWARE
PARA SIMULAÇÃO DO FLUXO DE CALOR NO SOLO**

RAPHAEL PIRES FERREIRA

*Dissertação apresentada ao Programa
de Pós-graduação em Física Ambiental
da Universidade Federal de Mato
Grosso, como parte dos requisitos para
obtenção do título de Mestre em Física
Ambiental.*

Orientador: Prof. Dr. Denilton Carlos Gaio

Cuiabá, MT
Fevereiro - 2017

DEDICATÓRIA

A minha esposa Maria
Daniele e a minha família.

AGRADECIMENTOS

- A Deus, por proporcionar mais essa oportunidade em minha vida;
- A minha esposa Maria Daniele, que me incentiva e acredita em meu potencial e me inspira a sempre continuar crescendo;
- Ao meu orientador, professor Dr. Denilton Carlos Gaio, pela paciência e incentivo durante o curso de pós-graduação;
- Ao coordenador do curso, professor Dr. José de Souza Nogueira (Paraná), pelo trabalho que realiza a frente do programa de Pós-Graduação em Física Ambiental e pela parceria durante esses anos de estudo;
- Aos professores Dr. Carlo Ralph, Dra. Carmem, Dr. Campelo, Dr. Francisco Lobo, Dr. Geraldo Neves, Dra. Iramaia, Dr. Paulo Arruda que ministraram as disciplinas no curso de Mestrado do Programa de Pós Graduação em Física Ambiental com imensa dedicação;
- Aos membros da banca que contribuíram na melhoria de minha dissertação com valiosas sugestões;
- Aos colegas que colaboraram durante todo o curso e contribuíram de forma direta e indireta a esse trabalho;
- A Soilce e ao Cesário que estão sempre dispostos a colaborar e me auxiliaram durante esses anos com os assuntos administrativos do programa;
- Ao Coordenador da área de Engenharia de Software da STI/UFMT, Fábio Alves, que compreendeu e me incentivou a estudar e concluir esse curso de Mestrado;

SUMÁRIO

LISTA DE FIGURAS.....	II
LISTA DE TABELAS.....	III
LISTA DE ABREVIATURAS E SÍMBOLOS	IV
RESUMO	VI
ABSTRACT.....	VII
1. INTRODUÇÃO	1
2. REVISÃO DA LITERATURA	5
2.1 PROPRIEDADES FÍSICAS DO SOLO	6
2.2 SISTEMA SOLO-PLANTA-ATMOSFERA (SSPA).....	9
2.3 FLUXO DE CALOR NO SOLO.....	12
2.4 HARDWARE DESCRIPTION LANGUAGE (HDL)	15
2.4.1 Verilog.....	19
3. MATERIAL E MÉTODOS	22
3.1 EQUIPAMENTOS E DISPOSITIVOS.....	22
3.2 SIMULADOR DE CIRCUITO ELÉTRICO.....	22
3.3 MODELO DE SIMULAÇÃO DO FLUXO DE CALOR NO SOLO	23
3.4 DESENVOLVIMENTO DO MÓDULO SOLO.....	26
4. RESULTADOS	30
4.1 MÓDULO SOLO	30
4.2 UTILIZAÇÃO DO MÓDULO DESENVOLVIDO	33
4.3 SIMULAÇÃO DE UM PERFIL DE SOLO	34
5. CONCLUSÃO.....	38
6. REFERÊNCIAS BIBLIOGRÁFICAS	40
APÊNDICE A	43

LISTA DE FIGURAS

Figura 1 - Camadas do solo.....	5
Figura 2 - Triângulo textural do solo com as 13 classes texturais.	7
Figura 3 - Classificação e representação esquemática da estrutura do solo.....	9
Figura 4 – Esquema do Circuito Hídrico de uma Planta.....	10
Figura 5 - Curvas de temperatura por horário e profundidade.....	14
Figura 6 - Fluxo de concepção de um circuito.....	18
Figura 7 - Porta lógica E	20
Figura 8 - Código Fonte 1 - porta lógica E	20
Figura 9 - Interface gráfica do QUCS	23
Figura 10 - Esquema de simulação do fluxo de calor no solo	24
Figura 11 – Circuito básico	25
Figura 12 - Circuito elétrico que representa o solo	26
Figura 13 - Modelo para Desenvolvimento de Nova Versão do Software QUCS	27
Figura 14 - Modelo para Desenvolvimento do Módulo por HDL	28
Figura 15 - Arquivos em C++ criados automaticamente	29
Figura 16 - Módulo solo integrado ao QUCS	30
Figura 17 - Código Fonte 2 - Utilização das Bibliotecas em Verilog-A.....	31
Figura 18 - Código Fonte 3 - Codificação das Resistências e dos Capacitores	31
Figura 19 - Propriedade de edição do Solo	31
Figura 20 - Código Fonte 4 - Declaração das Variáveis Editáveis	32
Figura 21 - Código Fonte 5 - Declaração do Nome do Módulo, Entradas e Saídas ..	32
Figura 22 – Módulo Solo desenvolvido em Verilog-A.....	32
Figura 23 - Código Fonte 6 – Módulo Solo em Verilog-A.....	33
Figura 24 - Esquemático de simulação	34
Figura 25 - Comportamento da simulação	35
Figura 26 - Gráfico da Radiação Solar.....	36
Figura 27 - Temperaturas por profundidade	37
Figura 28 - Temperaturas por profundidade 2	37

LISTA DE TABELAS

Tabela 1 Classificação da textura do solo pelo tamanho das partículas.	7
Tabela 2 - Relação da textura do solo com algumas propriedades	8
Tabela 3 - Grandezas Análogas entre o fluxo elétrico e o fluxo hídrico.....	11
Tabela 4 - Tabela Verdade da Porta Lógica E	20

LISTA DE ABREVIATURAS E SÍMBOLOS

QUCS – *Quite Universal Circuit Simulator*

PPGFA – Programa de Pós-Graduação em Física Ambiental

SSPA – Sistema Solo Planta Atmosfera

SPA – Solo Planta Atmosfera

HDL – *Hardware Description Language*

LP – Linguagem de Programação

VLSI – Integração em Larga Escala

IEEE - Instituto de Engenheiros Elétricos e Eletrônicos

GUI – *Guide User Interface*

GPL – *General Public License*

SSD – *Solid State Drive*

GB – *Gigabyte*

RAM – *Random Access Memory*

DDR – *Double Data Rate*

SPA – Solo Planta Atmosfera

O – Camada orgânica superficial

A – Rocha alterada e húmus

EUA – Estados Unidos da América

E – Camada mineral com pouca matéria orgânica

C – Camada mineral pouco ou parcialmente alterada

F - Fluxo da água em equilíbrio por todo o sistema

T - Fluxo de vapor perdido pela transpiração

Ψ_{sr} - Potencial do sistema radicular

Ψ_s - Potencial de água no solo

Ψ_{ix} - Potencial do ponto inicial do xilema

Ψ_{fx} - Potencial do ponto final do xilema

Ψ_{se} - Potencial do sítio de evaporação

Ψ_{ar} - Potencial do sítio de ar

r_s - Resistência do solo

r_r - Resistência da raiz

r_x - Resistência do xilema

r_f - Resistência da folha

r_g - Resistência da fase gasosa

ddp – Diferença de Potencial

RESUMO

FERREIRA, R.P. **Desenvolvimento de Módulo De Software Para Simulação do Fluxo de Calor no Solo** Cuiabá, 2017. 66p. Dissertação (Mestrado) – Programa de Pós-Graduação em Física Ambiental, Universidade Federal de Mato Grosso. Cuiabá 2017

Tem se utilizado, no Programa de Pós Graduação em Física Ambiental, um simulador de circuitos elétricos para análise do balanço de variáveis ambientais (hídrico e de energia, por exemplo). O software de simulação, QUCS, apresenta de forma pouco prática esse trabalho com variáveis ambientais, pois é voltado para o segmento elétrico. O objetivo desse trabalho é a modelagem do solo em formato nativo para o QUCS, em que é possível realizar a simulação de fluxo de calor no solo. Com este modelo se trabalha de forma mais transparente na ferramenta e mais prática, uma vez que não é mais necessário se preocupar com características elétricas do componente. Foram observados resultados satisfatórios, uma vez que foi possível eliminar algumas características elétricas que não estão presentes no ambiente natural, como o ruído branco. Os testes conduzidos demonstraram a adesão dos resultados aos encontrados na literatura de referência e nos demonstra a capacidade de adaptação do software de simulação, que se mostrou muito interessante para hospedar, de forma completa, um conjunto de módulos ambientais.

Palavras-chave: Análogo Elétrico de Fluxos de variáveis ambientais, Fluxo de calor no solo, QUCS.

ABSTRACT

FERREIRA, R.P. **Development of Software Module for Simulation of Soil Heat Flux**. Cuiabá. 2017. 66p. Dissertation (MSc in Environmental Physics) – Postgraduate Program in Environmental Physics, Federal University of Mato Grosso. Cuiabá 2017

A simulator of electrical circuits has been used in the Graduate Program in Environmental Physics to analyze the balance of environmental variables (water and energy, for example). The simulation software, QUCS, presents in an unpractical way this work with environmental variables, since it is geared towards the electric segment. The objective of this dissertation is to present the soil modelling in native format for Quite Universal Circuit Simulator, where it is possible to simulate heat flux on the ground. With this model working in a more transparent manner in the tool and more practical, since it is no longer necessary to worry about electrical characteristics of the component, very satisfactory results were obtained since it was possible to eliminate some electrical characteristics that are not present in the natural environment, such as white noise. Tests conducted showed adherence of results to those found in the reference literature and demonstrates the adaptability of the simulation software, which proved to be very interesting to host, a complete set of environmental modules.

Key words: Electrical Analog Flows of Environmental Variables, Soil Heat Flow, QUCS.

1. INTRODUÇÃO

Pesquisas ambientais têm adquirido, cada vez mais, maior importância, devido à necessidade de se compreender e dar respostas à ação antrópica sobre o globo terrestre, razão, por exemplo, das mudanças climáticas globais.

A Física Ambiental é uma ciência multidisciplinar, que envolve áreas da física, meteorologia, química, ecofisiologia, biofísica, dentre outras, desenvolvendo técnicas próprias de medidas e de modelagem de variáveis ambientais, edafoclimáticas e de fisiologia de plantas.

Muitas pesquisas em Física Ambiental utilizam o conceito do contínuo Solo-Planta-Atmosfera (SPA) para explicar a dinâmica hídrica. Com esse conceito, os pesquisadores descrevem os caminhos seguidos pela água, desde o solo, passando pela planta e chegando à atmosfera, sumidouro final, e, se alguma alteração ocorrer nesse intervalo, podem observar seus efeitos. O SPA foi proposto por Huber (1928) e Van den Honert (1948), que introduziram essa metodologia com a aplicação da Lei de Ohm Análoga para o fluxo hídrico das plantas.

Ao simular as variáveis ambientais de forma análoga aos circuitos elétricos, vários autores utilizam-se de um rigor formal baseado em séries de mais de um século de experimentos. A analogia eletro-hídrica já foi utilizada, por exemplo, por Lang et al (1969), Cowan (1972) e Johnsson & Prytz (2002).

Soluções numéricas das equações diferenciais advindas dos métodos utilizados em Física Ambiental podem exigir alto custo computacional e, por conseguinte, financeiro. Gaio (2011) introduziu a utilização do software de simulação de circuitos elétricos chamado QUCS para a resolução desta analogia. A vantagem da utilização desse software é a possibilidade de se criar, com facilidade, cenários com diferentes atributos, estudar diferentes regimes, introduzir dados experimentais, de modo a calibrar e validar o modelo.

Nessa mesma linha de pesquisa, de utilização de software de simulação para a modelagem da dinâmica hídrica análoga à elétrica, Pereira (2013) utilizou um conjunto de divisores de corrente para modelar a partição da radiação solar incidente

sobre o estômato. Com base neste circuito, e com a introdução de um resistor de transferência, esse autor modelou a abertura estomática.

Siqueira (2014) obteve resultados satisfatórios, a partir de simulações com o QUCS, da construção do modelo das vias de transporte hídrico da raiz em plantas vasculares.

Essa metodologia não se restringe à modelagem análoga hídrico-elétrica. Com o QUCS, pode-se modelar o fluxo de calor no solo. Desta forma, o calor também é análogo à corrente elétrica; o gradiente de temperatura é análogo à diferença de potencial elétrico (ddp) e a resistência ao fluxo de calor, bem como a propriedade de armazenamento de energia interna são análogas à resistência e a capacitância elétrica, respectivamente.

As pesquisas com o uso do QUCS no Programa de Pós-Graduação em Física Ambiental desenvolvem-se hoje em duas novas frentes. Até o presente momento, as modelagens foram realizadas com séries de dados descritas na literatura, não necessariamente em um único arranjo experimental e nem em um único artigo. Isso implica em não se poder garantir consistência no conjunto de valores. Assim, uma das frentes é a montagem de arranjo experimental para coleta de dados. Precisa-se, para introdução do modelo no QUCS, de informações sobre fluxos e armazenamentos da propriedade em estudo simultaneamente em todos os pontos de referência.

Outra frente é o desenvolvimento de módulos de dispositivos característicos dos fluxos hídricos, de calor, de CO₂, ou outra propriedade própria na área de Física Ambiental, que funcionem de forma nativa no QUCS. Nesta frente, este trabalho é pioneiro, uma vez que não existem softwares capazes de realizar essas simulações com variáveis ambientais ou trabalhar a analogia hídrico-elétrica adequadamente como essa ferramenta possibilita.

A vantagem de se trabalhar, sem a necessidade de serem chamados de análogos na interface do usuário, com os módulos desenvolvidos, é a possibilidade de estabelecer intervalos de valores adequados. Atualmente, o software QUCS utiliza componentes elétricos, que podem ser encontrados no comércio. A analogia estava restrita aos intervalos de valores desses dispositivos comerciais. O trabalho de estabelecer essas analogias com a área de Física Ambiental nem sempre é fácil e

requer muitas tentativas, em alguns casos se tornando frustrante. O estabelecimento de módulos próprios montados diretamente no QUCS estabelece naturalmente as relações próprias para o fluxo hídrico, fluxo de CO₂, de calor etc.

No desenvolvimento de um componente QUCS para o estudo do fluxo de calor no solo apresentam-se diversas possibilidades de abordagem. Consideramos duas como principais: a primeira seria abordar a codificação diretamente nos arquivos fonte do software, uma vez que ele está disponível sobre uma licença GNU, ou seja, é um software livre e a linguagem utilizada seria o C++. A segunda alternativa seria o desenvolvimento dentro da própria ferramenta, porém utilizando uma linguagem menos conhecida, Verilog-A, e que opera sobre as premissas de ser uma linguagem de descrição de hardware, ou seja, uma linguagem de mais baixo nível.

Dado ser o foco da pesquisa disponibilizar um componente de fácil uso a outros pesquisadores da área de Física Ambiental, a segunda abordagem foi à escolhida. Na primeira opção, o processo de desenvolvimento e os testes do novo módulo implicariam na dependência de saber em que parte colocar esses novos arquivos fontes, como compilar uma nova versão do software completo, como criar um novo pacote de instalação, entre outras coisas, e, com isso, o processo não seria algo simples e poderia desestimular a criação de novos módulos por parte de pesquisadores não habituados com os processos de desenvolvimento de software e que ficariam na dependência de pessoal especializado para realizar todo esse processo de novas versões.

Para tanto, fica, para esses pesquisadores, apenas a necessidade de conhecimentos básicos das Linguagens de Descrição de Hardware (HDL). Lipovski (1977), Chu (2006), Weste e Harris (2011) e Palnitkar (2013) fornecem bom material de pesquisa na área de desenvolvimento de mais baixo nível, que traz consigo a teoria necessária para tratar desse trabalho de criação de circuitos e componentes elétricos em nível de programação.

O resultado foi à possibilidade de realizar a criação do módulo solo de forma satisfatória; submete-lo a testes de aderência ao modelo proposto por Gaio (2011) e verificar os resultados de uma simulação de fluxo de calor. Com estes testes, notou-se que o módulo funcionou da forma esperada e acompanhou as curvas

de temperatura encontradas na literatura para um perfil de solo em diferentes profundidades.

Dessa forma, o OBJETIVO GERAL deste trabalho é:

Desenvolver um componente solo integrado a qualquer projeto do software de simulação de circuitos elétricos, QUCS, que demonstre o comportamento do fluxo de calor no solo, facilitando a utilização em simulações ambientais no Programa de Pós-Graduação em Física Ambiental (PPGFA).

Definiu-se como OBJETIVOS ESPECÍFICOS:

- (i) Desenvolver, por meio de uma linguagem de descrição de hardware, o componente solo para simulações análogas às elétricas, balanço de variáveis ambientais no software QUCS;
- (ii) Validar o comportamento do componente desenvolvido por meio de um modelo de associação de módulos básicos, com propriedades resistivas e capacitivas para variáveis ambientais, que encapsulados geram “dispositivos ambientais”;
- (iii) Elaborar documentação, bem como os arquivos fonte necessário, roteiro de como operacionalizar o QUCS para compilar e utilizar o módulo, trazendo mais praticidade nas futuras simulações que outros pesquisadores irão realizar (a qual ficará livremente disponível a pesquisadores do Programa PGFA).

No próximo capítulo, será desenvolvida a revisão da bibliografia básica dessa dissertação. Basicamente serão descritas as propriedades térmicas do solo e as características da Linguagem HDL. No capítulo seguinte tem-se a descrição da metodologia seguida, em especial, versa sobre os passos seguidos na criação do componente no simulador QUCS e sua utilização. Segue o capítulo de resultado e análises e as conclusões do trabalho.

2. REVISÃO DA LITERATURA

O solo, substrato das plantas, tem sua importância como suporte físico e no fornecimento de água e nutrientes, desta forma, é de fundamental importância seu estudo para se compreender o meio ambiente.

Ao se retirar uma porção de solo para análise, pode-se notar, em um corte vertical dessa amostra, diferentes camadas, ou horizontes, de solo, conforme Araguaia (2005) apresenta na Figura 1.

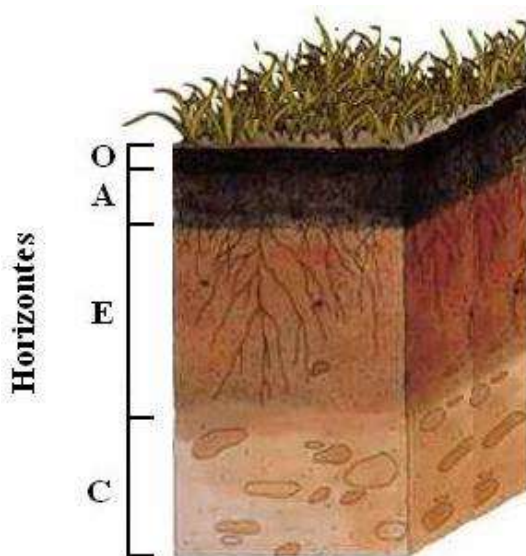


Figura 1 - Camadas do solo

Horizontes do solo: O – Camada orgânica superficial, A – Rocha alterada e húmus, E – Camada mineral com pouca matéria orgânica, C – Camada mineral pouco ou parcialmente alterada.

Fonte: Araguaia (2005).

Para conseguir uma boa compreensão do solo e de sua importância para os estudos ambientais é necessário estudar outros aspectos que serão abordados nessa seção do trabalho, como o Sistema Solo-Planta-Atmosfera, as Propriedades Físicas do Solo, o Fluxo de Calor no Solo. Por fim, para a construção do componente análogo elétrico do solo é preciso revisar sobre linguagens de descrição de hardware (HDL) e Verilog que é a linguagem de programação (LP) utilizada para o desenvolvimento do componente.

2.1 PROPRIEDADES FÍSICAS DO SOLO

Conforme Reinert & Reichert (2006) a física de solos estuda e define, qualitativa e quantitativamente, as propriedades físicas, bem como sua medição, predição e controle, com o objetivo principal de entender os mecanismos que governam a funcionalidade dos solos e seu papel na biosfera.

O solo tem origem da rocha, que lhe dá suporte, a partir de processos físicos, químicos e biológicos sofridos. Reconhecem-se cinco fatores da formação do solo: material original (rocha), M; tempo (idade), I; clima, C; topografia, T; e organismos vivos, O. Utilizando a linguagem matemática pode-se dizer que (REICHARDT & TIMM, 2004):

$$Solo = f(M, I, C, T, O) \quad (1)$$

Diferentes resultados podem surgir a partir de valores diferentes para os quatro últimos fatores da formação do solo, ou seja, com o mesmo material original, M, surgem diferentes tipos de solos. Pode-se estabelecer um sistema trifásico, sólido, líquido e gasoso. Reinert & Reichert (2006) relatam que o solo considerado fisicamente ideal para o crescimento de plantas é aquele que apresenta boa retenção de água, bom arejamento, bom suprimento de calor e pouca resistência ao crescimento radicular, e junto a isso uma boa estabilidade dos agregados e uma boa infiltração de água no solo, que representam condições físicas importantes para qualidade ambiental dos ecossistemas.

A porosidade do solo é uma das características físicas mais importantes do solo, pois é por meio dela que vários mecanismos importantes ocorrem, como a retenção e o fluxo de água e ar. Também é possível analisar a porosidade em conjunto com a matriz do solo para estabelecer importantes propriedades como a relação de massa e volume das fases do sistema solo (LACERDA, 2007) e (REICHARDT & TIMM, 2004).

O modo como às partículas sólidas se arranjam na formação dos diferentes tipos de solo é um fator que influencia diretamente a porosidade

A definição da textura do solo é algo que deve ser amplamente considerado. A Sociedade Brasileira de Ciência do Solo definiu quatro classes de tamanho de

partículas do solo. Por meio do tamanho dessas partículas pode-se definir a classe dos materiais que formam o solo e também definir a sua textura.

Define-se como textura de um solo, a relação quantitativa das diferentes classes de tamanho de suas partículas sólidas: areia, silte e argila.

Tabela 1 Classificação da textura do solo pelo tamanho das partículas.

Textura	Tamanho da partícula (diâmetro)
Areia grossa	2 a 0,2 mm
Areia fina	0,2 a 0,05 mm
Silte	0,05 a 0,002 mm
Argila	menor que 0,002 mm

O total de partículas no solo, de acordo com a Tabela 1, será o somatório da quantidade dessas partículas, desconsiderando todas as partículas que forem maiores que 2 mm. Um solo pode ser completamente formado de argila, de areia ou de silte, como pode ocorrer grande heterogeneidade na interação entre os tipos de partículas. O diagrama da Figura 2 possibilita uma forma prática de caracterizar um solo a partir de sua textura.

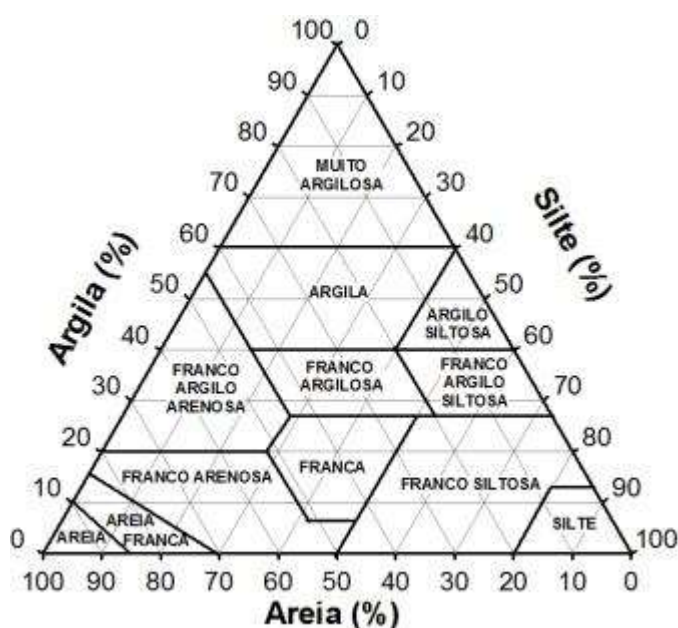


Figura 2 - Triângulo textural do solo com as 13 classes texturais.

Representação para definição da textura do solo e suas características físicas.

Fonte: Lier (2010).

Este diagrama textural, que consta da sobreposição de três triângulos, que representam as quantidades percentuais de argila, silte e areia do solo, baseia-se na classificação norte americana (USDA, United States Department of Agriculture). “A Sociedade Brasileira de Ciência do Solo acrescentou a classe textural muito argilosa ao triângulo do USDA” (FERREIRA, 2010). A leitura se faz encontrando-se a porcentagem de cada classe. Para a areia, encontra-se a concentração na base do triângulo. Segue-se que, a reta, a partir desse ponto, paralela ao lado do triângulo da escala de silte (à direita), tem-se a mesma concentração de areia. Para o silte, segue-se a linha paralela ao lado da escala de argila e, para a argila, a paralela ao lado do triângulo da escala de areia (retas horizontais).

Além da classe textural, o uso do solo é também fator determinante de sua densidade. Em solos de mata nativa, por exemplo, que não possuem interferência humana, os valores de densidade são menores e os de porosidade são maiores (CALONEGO et al., 2012). A partir da densidade de um solo, pode-se inferir outras propriedades, tais como o nível de compactação do solo, níveis de porosidade, resistência à penetração radicular.

A Tabela 2 estabelece uma relação entre propriedades do solo e a textura.

Tabela 2 - Relação da textura do solo com algumas propriedades

Solos arenosos	Solos argilosos
Menor porosidade do solo	Maior porosidade do solo
Menor micro e maior macroporosidade	Maior micro e menor macroporosidade
Baixa retenção de água	Alta retenção de água
Boa drenagem e aeração	Drenagem lenta e pouco arejado
Menor densidade do solo	Maior densidade do solo
Aquece rápido	Aquece lentamente
Resiste a compactação	Mais suscetível a compactação
Baixa CTC	Maior CTC
Mais lixiviável	Menos lixiviável
Mais suscetível à erosão	Mais resistente à erosão
Coesão baixa, friável	Coesão elevada, firme
Fácil preparo mecânico	Mais resistente ao preparo
Consistência friável ⁽¹⁾	Consistência plástica e pegajosa ⁽²⁾
MO baixa e rápida decomposição	MO de média a alta e lenta decomposição

Legenda: MO - matéria orgânica; solo (1) úmido, (2) molhado.

Fonte: Reinert e Reichert (2006).

Outro atributo que caracteriza o solo é sua estrutura, ela é concebida a partir do agrupamento e organização das partículas do solo (argila, silte e areia) – as quais

podem ser chamadas de partículas primárias – com outras, tanto orgânicas quanto minerais. Essa agregação de partículas origina unidades estruturais compostas, chamadas de micro e macroagregados do solo (CAPECHE, 2008).

A estrutura do solo possui uma classificação própria que é referida a forma de agregação e ao tipo dessas estruturas compostas geradas, conforme a Figura 3.

<u>TIPOS</u>	<u>SUBTIPOS</u>	<u>EIXOS</u>	<u>REPRESENTAÇÃO</u>	<u>CARACTERÍSTICA</u>
ESFEROIDAL	GRANULAR GRUMOSA			HORIZONTES SUPERFICIAIS
BLOCO (cúbica)	ANGULARES SUBANGULARES			HORIZONTE B
PRISMÁTICA	PRISMÁTICA COLUNAR			HORIZONTE B HORIZONTES SALINOS
LAMINAR	(não tem subtipo)			HORIZONTE C

Figura 3 - Classificação e representação esquemática da estrutura do solo

Fonte: Heinrichs (2010).

Gaio et al. (2015) dizem que as propriedades físicas de resistência à penetração e a umidade estão diretamente relacionadas com o crescimento e desenvolvimento radicular das plantas. Pode-se dizer então que a interferência humana no solo pode alterar as suas propriedades físicas.

2.2 SISTEMA SOLO-PLANTA-ATMOSFERA (SSPA)

O estudo do fluxo contínuo de água no Sistema Solo-Planta-Atmosfera deve considerar desde as forças matriciais que ligam a água ao solo até a demanda evaporativa da atmosfera no topo da copa de uma árvore. Entre um e outro ponto, o

que irá caracterizar o caminho seguido pela água serão as resistências e as capacitâncias dos diversos componentes da planta. Huber (1928) e Van der Honert (1948) aplicaram uma lei análoga à de Ohm ao fluxo de água, utilizando técnicas derivadas da área de circuitos elétricos, conforme se pode notar na Figura 4, como resistências, capacitores, entre outros.

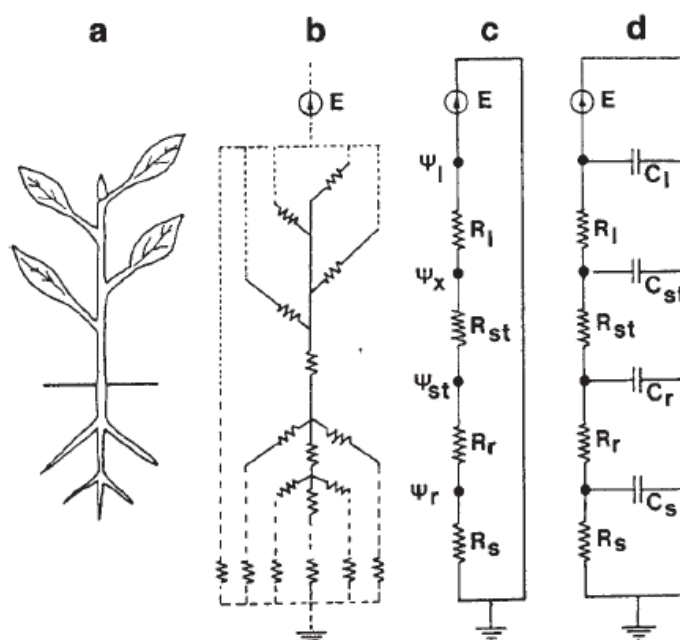


Figura 4 – Esquema do Circuito Hídrico de uma Planta

(a) Representação simplificada da Planta; (b) Rede correspondente das resistências ao fluxo, incluindo as resistências do solo, raiz, caule e folhas; (c) Modelo da catenária simplificado com os caminhos de galhos de (b) representados pelas resistências R_s (solo), R_r (raiz), R_{st} (caule) e R_l (folha) e (d) Mesmo modelo de (c), porém incluindo as capacitâncias correspondentes dos tecidos. “E” representa a demanda e a direção da transpiração. A partir de Jones (1983), adaptado de Pallardy (1989).
Fonte: Pallardy (2008).

Conforme Gil-Peigrín *et al* (2005), interrupções no contínuo SSPA, em qualquer parte, por qualquer que seja o motivo, afeta o desempenho da planta, podendo inclusive comprometer sua sobrevivência.

A analogia entre o fluxo hídrico e o fluxo elétrico já vem sendo alvo de vários estudos e pesquisas.

Em um modelo simplificado (EMBRAPA, 2011), o fluxo de água ocorre em equilíbrio dinâmico desde o solo até a atmosfera, sendo originado por diferença de potenciais da água, entre os diversos pontos do fluxo, e dependente de resistências ao transporte em cada parte do sistema. Pode-se então escrever em forma de equação a dinâmica deste fluxo:

$$F = T = \frac{\Psi_{sr} - \Psi_s}{r_s} = \frac{\Psi_{ix} - \Psi_{sr}}{r_r} = \frac{\Psi_{fx} - \Psi_{ix}}{r_x} = \frac{\Psi_{se} - \Psi_{fx}}{r_f} = \frac{\Psi_{ar} - \Psi_{se}}{r_g} \quad (2)$$

F: Fluxo da água em equilíbrio por todo o sistema;

T: Fluxo de vapor perdido pela transpiração;

Ψ_{sr} : Potencial do sistema radicular;

Ψ_s : Potencial de água no solo;

Ψ_{ix} : Potencial do ponto inicial do xilema;

Ψ_{fx} : Potencial do ponto final do xilema;

Ψ_{se} : Potencial do sítio de evaporação;

Ψ_{ar} : Potencial do sítio de ar;

r_s : Resistência do solo;

r_r : Resistência da raiz;

r_x : Resistência do xilema;

r_f : Resistência da folha;

r_g : Resistência da fase gasosa;

“Todos os componentes do contínuo SPA armazenam água. No solo, por exemplo, nos espaços entre os grãos de argila. No ar, o conteúdo de água, dependendo da temperatura, pode representar até 4% do valor da massa de ar seco” (GAIO, 2011, pg. 20).

Pereira (2013) utiliza, além de resistores e capacitores, transistores para modelar a influência da radiação solar sobre o fluxo hídrico em estômatos.

Um avanço nas pesquisas foi à definição de grandezas análogas entre fluxos elétrico e hídrico estabelecidos por Gaio (2011), conforme Tabela 3.

Tabela 3 - Grandezas Análogas entre o fluxo elétrico e o fluxo hídrico.

Fluxo Elétrico			Fluxo hídrico		
Grandeza	Símbolo	Relação	Grandeza	Símbolo	Relação
Carga (Q)	coulomb (C)		Volume (W)		m ³
Potencial (V)	volt (V)	J.C ⁻¹	Potencial (ψ)	Pascal (Pa)	J.m ⁻³
Corrente (i)	ampere (A)	C.s ⁻¹	Fluxo (f)		m ³ .s ⁻¹
Resistência (R)	ohm (Ω)	V.A ⁻¹	Resistência (R)		Pa.s.m ⁻³
Capacitância (C)	farad (F)	C.V ⁻¹	Capacitância (C)		m ³ .Pa ⁻¹

A partir das analogias: carga elétrica e volume de água (linha 1) e Potencial Elétrico e o potencial hídrico (linha 2), as analogias com as outras grandezas se estabelecem automaticamente.

Fonte: Gaio (2011)

2.3 FLUXO DE CALOR NO SOLO

Dos processos que permeiam o comportamento do solo, o conhecimento do fluxo de calor no solo torna-se necessário para a devida compreensão do comportamento de seus componentes, como a difusão dos solutos e dos gases, o desenvolvimento das plantas, como germinação de sementes (ZUCARELI, et al., 2010), dentre outros.

A energia disponível na superfície terrestre é particionada da seguinte forma: (i) uma parte é utilizada para aquecer o solo por condução e (ii) outra parte para aquecer a atmosfera, dando origem ao processo de convecção; (iii) uma parte para evaporação da água; (iv) e outra é usada pelos organismos em processos fotoquímicos (LOPES et al 2011).

De importância no estudo termodinâmico e agrônômico do solo são suas propriedades térmicas, sobretudo o calor específico do solo e a condutividade térmica do solo. A primeira se refere ao solo como reservatório de energia e a segunda, como transmissor de calor (REICHARDT & TIMM, 2004). Energia térmica pode ser transferida em meios porosos, como o solo, por meio de condução molecular e convecção (BASTIAANSEN, 1995), mas a principal forma é a condução (HILLEL, 1998).

Quando os raios solares atingem o solo, uma parte dessa energia é absorvida, esquentando o solo. Este calor é transferido da superfície por condução para camadas mais profundas do solo; utilizado para o aquecimento da atmosfera e na evaporação da água (CAMPBELL E NORMAN, 1998).

O valor, a qualquer momento, da temperatura do solo (bem como a forma de sua variação no tempo e no espaço) é um fator de importância primária para determinar as taxas e a direção dos processos físicos do solo e da sua troca de energia e massa com a atmosfera (HILLEL, 1998).

O fluxo de calor deve-se à tendência homogeneizadora da natureza. Desta forma, flui calor em um material de um ponto de temperatura maior para outro de menor temperatura até que se atinja o equilíbrio térmico. (HILLEL, 1998).

No trabalho realizado por Oliveira et al. (2009) pode-se verificar que o fluxo de calor no solo é dependente da incidência dos raios solares sobre a superfície e ao

tipo de solo. Solos argilosos possuem maior condutividade térmica que solos arenosos (LOPES et al 2011). A teoria clássica de transporte de calor no solo considera que a condutividade do fluxo de calor no solo depende do fluxo através dos diversos constituintes: água contida no solo (umidade do solo), ar, minerais e material orgânico (BASTIAANSEN, 1995).

Conforme Campbell e Norman (1998), no solo deve-se considerar os reservatórios de calor e também o seu fluxo. Particiona-se o solo em camadas para seu estudo. O fluxo de calor de uma camada para a outra é calculado com base na lei de Fourier:

$$H = -K \frac{dT}{dz} \quad (3)$$

Na qual, H é a densidade do fluxo de calor em W/m²; K é a condutividade térmica em Wm⁻¹K⁻¹ e dT/dz é o gradiente vertical de temperatura em °C/m. Entende-se gradiente vertical como o módulo da componente vertical do gradiente.

A equação de continuidade deve ser resolvida de modo simultâneo a lei de Fourier:

$$\rho_s c_s \frac{\partial T}{\partial t} = -\frac{\partial G}{\partial z} \quad (4)$$

Em que ρ_s é a densidade do solo; c_s é seu calor específico. O produto $\rho_s c_s$ é a capacidade volumétrica de calor e G é a densidade do fluxo de calor no solo.

Tem-se a densidade do fluxo de calor no solo em ambas às equações, de modo que que é possível combina-las em uma só:

$$\rho_s c_s \frac{\partial T}{\partial t} = \frac{\partial}{\partial z} \left(K \frac{\partial T}{\partial z} \right) \quad (5)$$

A condutividade térmica independente explicitamente da profundidade. Dividindo-se os dois lados da Equação 5 por $\rho_s c_s$, obtem-se a Equação de Calor:

$$\frac{\partial T}{\partial t} = D \frac{\partial^2 T}{\partial z^2} \quad (6)$$

Nesta equação, $D = K/\rho_s c_s$ é a difusividade térmica do solo.

Conforme Campbell e Norman (1998), é possível notar que quando a variação com a altura do gradiente de temperatura é maior, também mais rapidamente está variando a temperatura com o tempo.

O valor da temperatura (Equação 7), solução para a equação de calor do solo (OMETTO, 1981) em função de t e z ($T_{(z,t)}$) deve-se ao fato de não ser a fonte de energia disponível (Sol) um ponto fixo.

$$T_{(z,t)} = \bar{T} \pm T_0 \cdot e^{\left(-z\sqrt{\frac{w}{2D}}\right)} \cdot \text{sen}\left(\omega t - z\sqrt{\frac{w}{2D}}\right) \quad (7)$$

A temperatura do solo segue a variação da radiação solar que atinge a superfície, a qual depende da velocidade de rotação da Terra ($\omega = 7,27 \cdot 10^{-3} \text{rd.s}^{-1}$). A curva da temperatura superficial é uma senóide, acompanhando a forma da curva da intensidade de radiação.

Na Equação 7, $\bar{T}$ é a temperatura média do perfil; T_0 é a amplitude térmica e D é a difusividade térmica.

Medidas experimentais em Azevedo e Galvani (2003) mostram a variação de temperatura em z , para diferentes instantes de tempo, conforme a Figura 5. Esse acentuado decréscimo da amplitude da temperatura, em apenas alguns decímetros de profundidade, evidencia a reduzida capacidade do solo em conduzir energia (VIANELLO & ALVES, 1991).

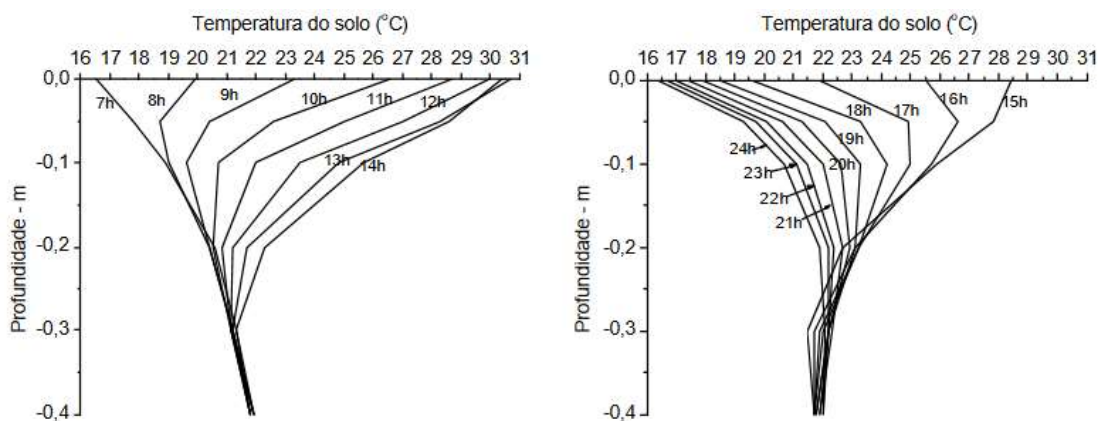


Figura 5 - Curvas de temperatura por horário e profundidade.

No lado esquerdo da Figura são os horários das 7hs da manhã até às 14hs e no lado direito é a continuidade das horas até às 00hs.

Fonte: Azevedo e Galvani (2003)

2.4 HARDWARE DESCRIPTION LANGUAGE (HDL)

Nos últimos 25 anos, o design de circuitos digitais evoluiu rapidamente. Os primeiros circuitos digitais eram desenvolvidos em tubos a vácuo e transistores. Logo depois os circuitos integrados foram criados, em que portas lógicas foram inseridas em um único chip. (PALNITKAR, 2013). O mesmo autor afirma que, com a introdução da tecnologia VLSI (Integração em larga escala), os designers puderam desenvolver chips com mais de 100 mil transistores. Essa veloz evolução, tanto em hardware, quanto em software, estabeleceu um consenso da necessidade de assistência de computadores para realizar esses designs mais sofisticados, assim como os testes. Não era mais possível realizar os testes e a produção diretamente das placas de circuitos.

Inicialmente, recorreu-se às linguagens de programação tradicionais na busca da solução desse problema. Nessas linguagens, entretanto, os programas, ou rotinas, são construídos para obedecer a uma sequência de processos, ou seja, trabalham de forma sequencial de passos, o que torna muito mais prático para qualquer ser humano estabelecer um algoritmo para desenvolver as atividades necessárias. Conforme Chu (2006), para se entender a necessidade de uma linguagem específica para desenvolvimento de hardware, é preciso buscar, nas origens das linguagens tradicionais, as suas bases de construção, que estão na sintaxe e na semântica. Enquanto a sintaxe compreende as regras gramaticais, a semântica confere o significado associado à construção da linguagem.

Chu (2006) também destaca características dos hardwares digitais: (i) esses sistemas são um agregado de sistemas menores; (ii) cada um possui tempo de operação característico e sequência lógica interna e (iii) as partes podem ser executadas simultaneamente, sem necessariamente depender de outro processo sequencial como nas linguagens de programação (LPs) tradicionais.

Cada operação ter um tempo de execução característico representa a propagação da espera de uma parte específica. Depois de completada cada uma das partes, que estava em operação, atualiza-se o valor de cada uma de suas portas de saída (CHU, 2006).

Tratando-se de programação para o desenvolvimento de dispositivos eletrônicos, circuito integrado, entre outros, foi necessário à criação de uma linguagem específica a estes tipos de dispositivos, pois era preciso uma lógica com características de hardware digital. Uma HDL (*Hardware Description Language*, termo em inglês para linguagem de descrição de *hardware*) pode ser utilizada para descrever a lógica de portas de máquinas sequenciais e os módulos funcionais, juntamente com suas conexões e o controle, em um dispositivo digital (LIPOVSKI, 1977).

Ainda em comparação com as linguagens tradicionais, pode-se estabelecer um fluxo de trabalho comparativo. As LPs tradicionais sempre são utilizadas para a resolução de algum problema específico, no qual tem-se uma entrada de dados, uma transformação desses em alguma informação e a sua saída esperada. Já com as linguagens de descrição de hardware é diferente, conforme Chu (2006) relata, é necessário um papel mais formal de 3 passos para estabelecer a utilização de uma HDL:

DOCUMENTAÇÃO FORMAL.

Mesmo que em alguns casos as pessoas utilizem a própria HDL para esse fim;

ENTRADA EM UM SIMULADOR.

Com a utilização de uma linguagem de descrição de hardware é possível realizar testes do comportamento do circuito descrito;

ENTRADA EM UM SINTETIZADOR.

Após os testes de simulação, o sintetizador trabalha o código HDL para realizar um processo de refinamento e geração de um novo código em HDL mais próximo do circuito real.

Muitos livros tratam a HDL como uma linguagem de programação, porém as linguagens de descrição de hardware são melhores compreendidas se tratadas como uma escrita rápida, ou abreviada, para descrever equipamentos digitais. O ideal é sempre pensar no hardware a ser modelado iniciando com um projeto no papel e depois levado para uma HDL, realizando os testes necessários antes da construção física do equipamento (WESTE & HARRIS, 2011).

Na Figura 6, nota-se o fluxo normalmente seguido para o desenvolvimento de um circuito digital de larga escala.

Para os estudos do fluxo hídrico no contínuo SSPA realizados por Gaio (2011), Pereira (2013) e Siqueira (2014) foi utilizado o simulador QUCS.

Pode-se considerar que há dois grupos de trabalho no desenvolvimento do QUCS, apesar de todo o software ser desenvolvido na linguagem de programação C. O primeiro grupo é responsável pela interface gráfica baseado no framework Qt da Digia. O Qt é um framework que estende a linguagem C++, ou seja, adiciona muitas funcionalidades que facilitam o desenvolvimento das aplicações. O segundo grupo de desenvolvimento é baseado na camada modelo do software, que seria o responsável por implementar as regras de negócio, e este é chamado de QUCSator, ele é responsável por trabalhar toda a parte de simulação, é um simulador de circuitos de linhas de comando. A entrada é em rede de lista em um determinado formato e gera uma saída no formato de um conjunto de dados do QUCS. Com este tipo de desenvolvimento, o QUCSator pode ser portado para utilização em outras aplicações quando necessário.

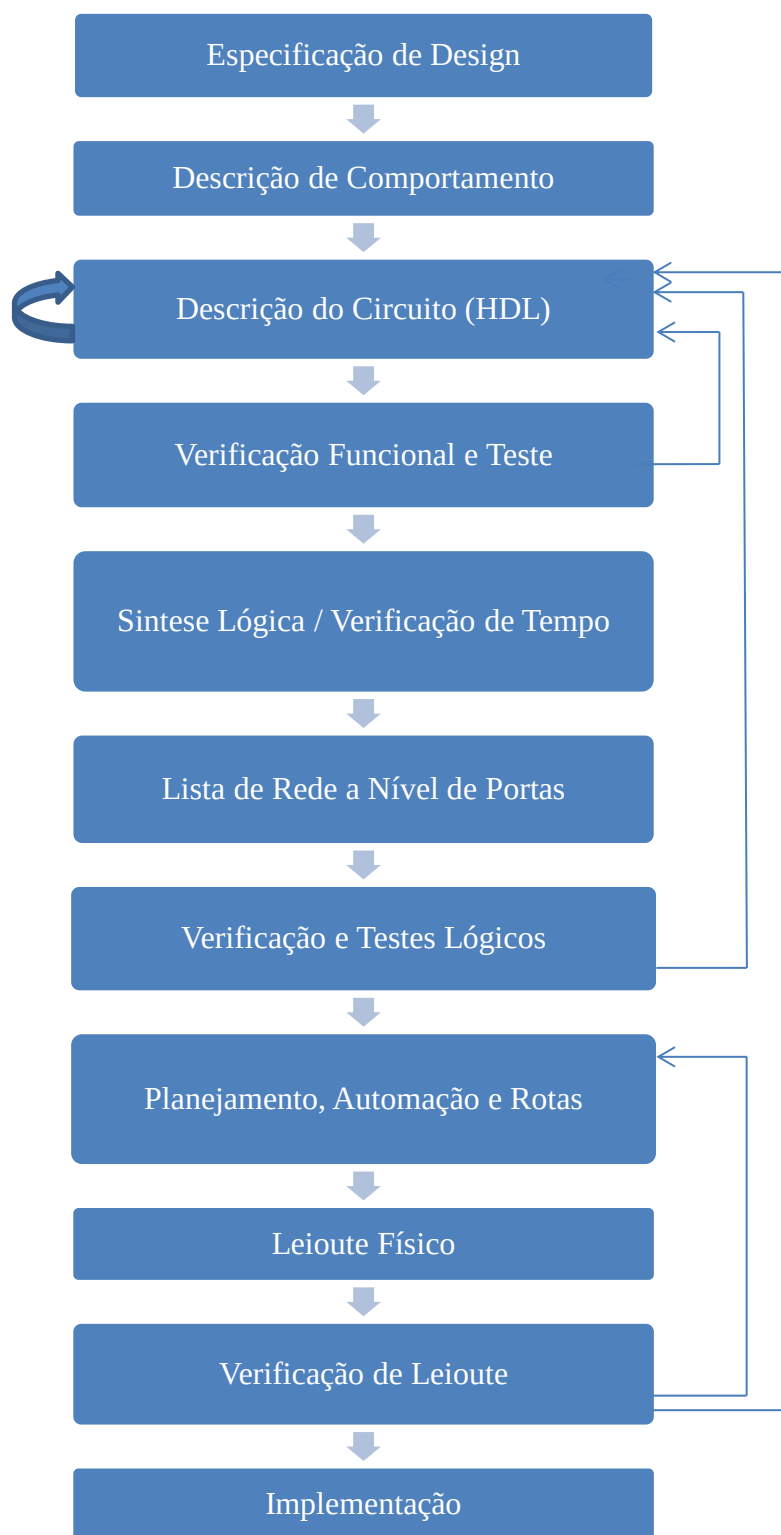


Figura 6 - Fluxo de concepção de um circuito

Demonstra o passo a passo da construção de um circuito de larga escala conforme os padrões formais de construção e design.

Fonte: Adaptado de Palnitkar (2013)

Atualmente existem duas variações de HDL que são largamente utilizadas, a Verilog e a VHDL. Essas duas variações são padrões estabelecidos pela IEEE (Instituto de Engenheiros Elétricos e Eletrônicos) e são linguagens livres. O QUCS apresenta apenas suporte ao Verilog, motivo pelo qual será apenas essa linguagem a seguir descrita.

2.4.1 Verilog

Verilog é uma linguagem que surgiu em 1983 pela Gateway Design Automation. Com isso, muitos designers adotaram as linguagens de descrição de hardware para a concepção e simulação dos circuitos digitais de larga escala (PALNITKAR, 2013). Devido a uma pressão de mercado na época, as linguagens proprietárias como Verilog estavam perdendo sua força, pois uma corrente de software livre estava se formando e fortalecendo essa ideologia de códigos abertos. Em 1990, com o aumento da força de atuação do Verilog na área, devido a decisão da Gateway Design Automation em abrir a linguagem para a comunidade, muitas empresas passaram a desenvolver hardware que utilizavam o Verilog como a linguagem principal.

Em 1993, um comitê foi estabelecido pela IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) para criar um padrão da linguagem, que agora é aberta, para toda a comunidade. Em 1995, obteve-se o resultado final desse comitê que estabeleceu o padrão 1364. O Verilog agora é um padrão estabelecido com normas internacionais, o que resulta em uma força muito maior para todos que utilizam essa linguagem HDL.

Um bom exemplo para entender o funcionamento desta linguagem de descrição de hardware seria a criação de um módulo para representar o funcionamento de uma porta E, conforme explícito na Figura 7.

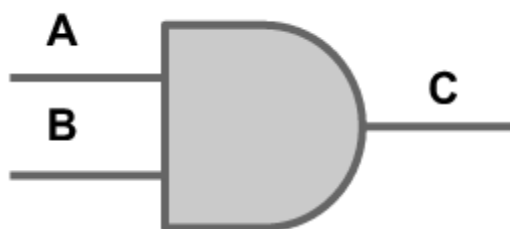


Figura 7 - Porta lógica E

Na Figura 7 é possível observar a porta lógica E com duas portas de entrada, A e B, e uma porta de saída, C. É possível verificar o seu funcionamento de acordo com a tabela verdade, conforme mostra a Tabela 4.

Tabela 4 - Tabela Verdade da Porta Lógica E

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Na Tabela A e B são portas de entrada, 0 e 1 são sinais, em que o zero representa que não há estímulo na porta e 1, há. A porta C somente é estimulada caso as portas A e B sejam estimuladas.

Para representar esse módulo na linguagem Verilog será necessário escrever algumas linhas de código.

```
//Código Verilog para o módulo de porta AND
module AND (input A, B output C);
assign C = (A & B);
endmodule
```

Figura 8 - Código Fonte 1 - porta lógica E

No Código Fonte 1, é possível notar uma simples implementação da porta lógica AND em linguagem Verilog. As linhas do código podem ser descritas:

- Module AND – é o nome do módulo que está sendo construído;
- (input A, B) – são as portas com os sinais de entrada do módulo;
- (output C) – é a porta com o sinal de saída do módulo;

- `assign C = (A&B)` – é o comando para dizer que a porta de saída C irá receber o resultado da operação A & B;
- `endomodule` – é a palavra reservada para a finalização do bloco de código que descreve o módulo AND.

3. MATERIAL E MÉTODOS

A partir de um modelo análogo-elétrico de simulação do fluxo de calor no solo desenvolvido no software QUCS (Quite Universal Circuit Simulator), foi desenvolvido um módulo próprio do solo para simulação do fluxo de calor em linguagem Verilog-A. Esse desenvolvimento insere-se em um projeto de construir um Simulador de Circuitos do Sistema Solo-Planta-Atmosfera, para fluxos hídrico, calor e CO₂, tendo como parâmetros as variáveis ambientais.

3.1 EQUIPAMENTOS E DISPOSITIVOS

O trabalho foi desenvolvido utilizando um computador desktop com a seguinte configuração:

- Processador AMD Phenom II 955 com 4 núcleos a 3.2 Ghz;
- 12 Gb de memória RAM DDR 3 à 1333 Mhz;
- Placa de vídeo Nvidia Geforce GTX 660 com 2 Gb de memória;
- SSD de 128 Gb.

Todo desenvolvimento aconteceu em ambiente Linux com a distribuição Ubuntu Desktop versão 14.04.3, com a versão do QUCS 0.0.18, que é a versão mais recente do software de simulação de circuitos elétricos.

3.2 SIMULADOR DE CIRCUITO ELÉTRICO

O software *Quite Universal Circuit Simulator* foi utilizado durante todo o trabalho. Apesar de ainda estar em versões beta, sem previsão divulgada de versão final, apresenta recursos úteis para o trabalho na área de simulação de circuitos elétricos. Ele é uma ferramenta completa como um simulador de circuitos integrados, sendo possível, por exemplo, realizar simulações de grandes e pequenos sinais, e verificar o comportamento dos ruídos nos circuitos.

A escolha deste software para o trabalho de pesquisa foi também fundamentada no tipo de licenciamento que ele utiliza, o QUCS é licenciado sob a

GPL, significando que ele é totalmente livre, programadores podem ter acesso a seu código fonte original e contribuir para o desenvolvimento dele, ou então, adequá-lo a sua necessidade, sem nenhum tipo de custo. Isso permitiu a implementação de módulos para circuitos do SSPA, o que torna esse software adequado em nível acadêmico na área das ciências ambientais.

Todo o circuito é montado em uma interface gráfica, tornando o ambiente amigável para o usuário final. Como é possível verificar na Figura 9, a interface gráfica (GUI) do QUCS é muito intuitiva, a grande área amarelada é o local em que os circuitos são montados. Os componentes para criar esses circuitos e esquemas estão todos dispostos ao lado esquerdo.

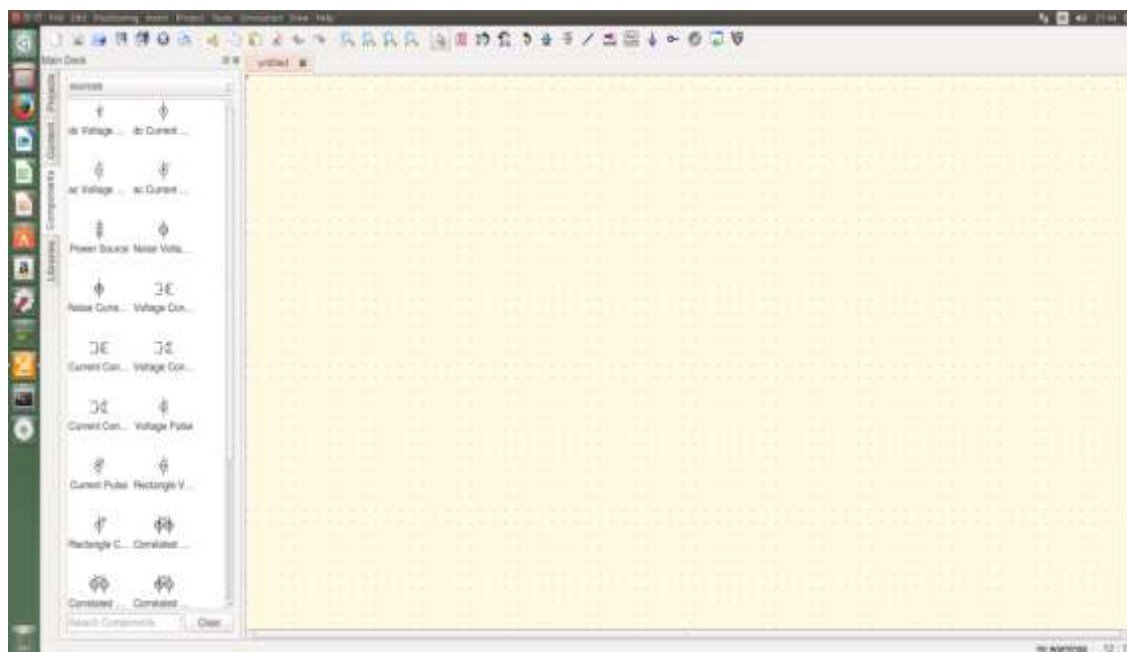


Figura 9 - Interface gráfica do QUCS

Figura que apresenta a interface de montagem de circuitos no software.

3.3 MODELO DE SIMULAÇÃO DO FLUXO DE CALOR NO SOLO

Todo trabalho de desenvolvimento originou-se a partir de um modelo construído na própria ferramenta de simulação. Esse modelo é um análogo elétrico

que representa o processo do fluxo de calor no solo e também das demais variáveis ambientais envolvidas no processo, como o Sol.

Na Figura 10, é possível visualizar o esquema completo de simulação do fluxo de calor no solo, transiente. Cada uma das faixas marrons no esquema está representando uma camada de solo, pela qual a energia irá entrar e sair para a camada subsequente. Cada uma dessas camadas está interconectada e as medições do fluxo ocorrem em cada uma das sondas U, de U1 à U6.

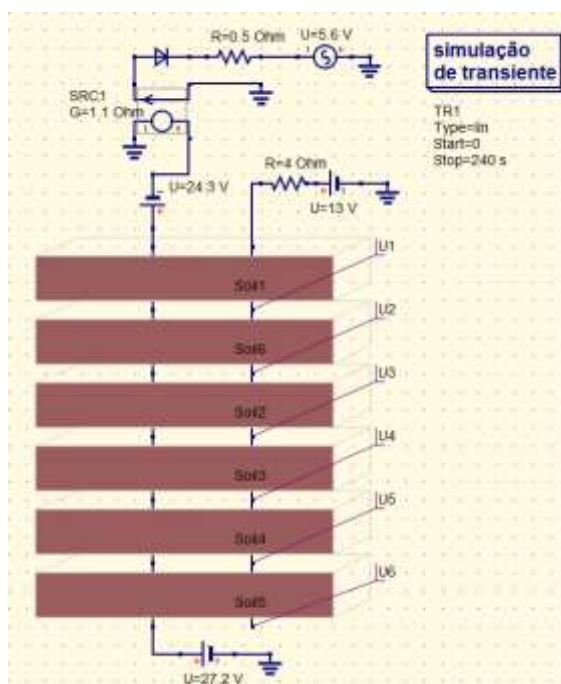


Figura 10 - Esquema de simulação do fluxo de calor no solo

Representação do esquema inicial, nota-se 6 camadas de solo e 6 sondas (U1 à U6) para medição da temperatura nas camadas.

Os circuitos criados para simulação do contínuo SSPA são modificações de um circuito base, exposto na Figura 11. Esses circuitos base modificados em arranjos de ligações em série ou em paralelo descrevem o fluxo contínuo da variável ambiental (GAIO, 2011). O circuito base é formado por resistores e capacitores que cumprem as funções, no caso do calor, de condução e armazenamento de energia, respectivamente.

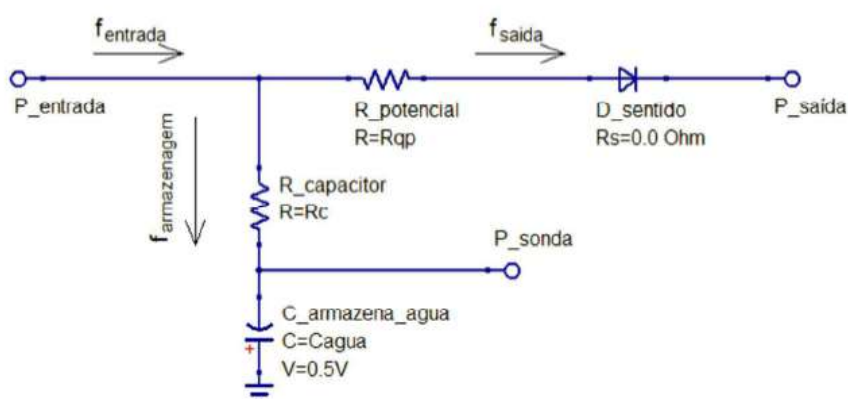


Figura 11 – Circuito básico

O fluxo de entrada (f_{entrada}) separa-se, de acordo com a lei dos nós, nos fluxos de saída ($f_{\text{saída}}$) e armazenagem ($f_{\text{armazenagem}}$), os quais dependem das resistências R_c e R_{qp} relativos aos resistores nomeados por $R_{\text{potencial}}$ e $R_{\text{capacitor}}$, respectivamente.

Fonte: Gaio (2011)

No próximo esquema, que está na Figura 12, há um detalhamento do que ocorre no interior das faixas de solo, ou seja, o circuito construído para representar o solo na simulação de fluxo de calor.

Na literatura (por exemplo, NOBEL, 2009), encontram-se valores característicos dos volumes de água armazenada em função da diferença de potencial hídrico para os diferentes componentes do sistema solo-planta-atmosfera. A partir da relação análoga da capacitância, $V = C\Delta\psi$, obteve-se as capacitâncias. De modo semelhante, a diferença de potencial hídrico e os valores de fluxo, permitiu determinar as resistências a partir da lei de Ohm análoga.

Deste modo, foi possível construir o circuito da Figura 12, composto por três resistências, sendo duas de 0.40 Ohms e outra de 2 Ohms, e uma capacitância de 5F. Semelhantemente foram construídos os circuitos dos outros componentes.

Cada uma das camadas “Soil”, apresentadas na Figura 10, são compostas pelo esquema elétrico da Figura 12, com esse modelo inicial a simulação acontece de forma satisfatória, porém sempre que é necessário alterar algum tipo de parâmetro nas faixas de solo, para representar alguma alteração do solo, é necessário editar por completo o esquema elétrico e alterando um esquema a ferramenta replica essas alterações para todas as camadas de solo, pois estão todos baseados no mesmo sub esquema elétrico.

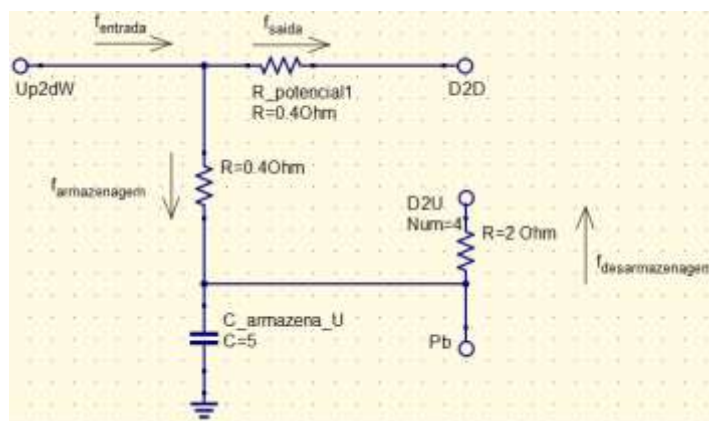


Figura 12 - Circuito elétrico que representa o solo

Circuito para representação do solo, que é derivado do circuito básico, com a adição do fluxo de desarmazenagem ($f_{desarmazenagem}$).

Essa falta de flexibilidade da ferramenta na criação de sub esquemas implica em ter que se replicar os sub esquemas elétricos em diferentes tipos de solos. Isso se aplica para qualquer outro tipo de variável ambiental que se queira simular, o que reduz a manutenibilidade com uso de um número grande sub esquemas. Como consequência, tem-se perda de controle, afetando os resultados esperados de saída.

3.4 DESENVOLVIMENTO DO MÓDULO SOLO

No estudo realizado para o processo de desenvolvimento do módulo, apresentaram-se duas alternativas. A primeira seria partir para uma nova versão do QUCS, em que uma nova compilação do programa seria necessária, desenvolvendo o módulo na linguagem C++. O início do fluxograma do modelo de desenvolvimento desta alternativa, apresentado na Figura 13, dá-se no “Estudo da construção do Software (SW)”.

A segunda alternativa seria o desenvolvimento a partir de uma linguagem de descrição de hardware, que a princípio se mostrou um pouco mais complicado devido a sua curva de aprendizagem ser um pouco mais longa. As linguagens HDL podem ser mais difíceis de compreender. Porém a linguagem Verilog-A, utilizada para o desenvolvimento desse módulo, provou ser bem acessível e com boas aproximações a linguagens de mais alto nível, como o próprio C++, conforme é possível notar na Figura 8 (página 20).

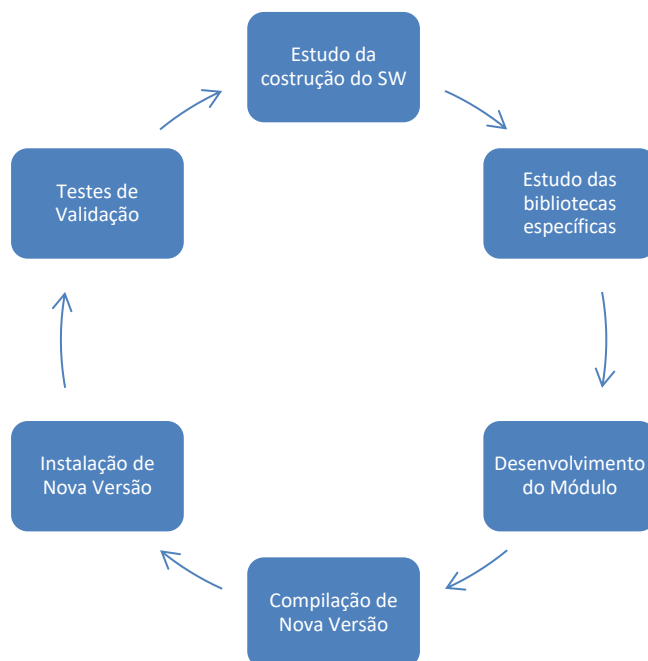


Figura 13 - Modelo para Desenvolvimento de Nova Versão do Software QUCS

Modelo de desenvolvimento mais longo, que representa os passos, de forma macro, para alteração e nova versão do QUCS.

A opção pela segunda alternativa se provou mais acertada a partir do momento que um dos principais objetivos seria a disponibilização do módulo para utilização das equipes do programa de pós-graduação em Física Ambiental. A complexidade e o conhecimento necessário para se compilar e gerar uma nova versão do software completo seria um impedimento para a usabilidade, praticidade e transparência para os usuários finais utilizarem não só o novo módulo desenvolvido, mas também, a possibilidade de explorar a criação de novos módulos sem a necessidade de alguém com alto conhecimento em linguagens de programação.

O ciclo de desenvolvimento da alternativa de se trabalhar dentro do próprio ambiente QUCS é menor, conforme a Figura 14. Portanto, esta alternativa mostrou-se ser mais vantajosa, uma vez que a possibilidade de se obter um produto final de forma mais ágil e prática é apresentada. Uma vantagem dessa opção é a oportunidade de testar de forma mais fácil os códigos desenvolvidos, já que se pode escrever, compilar e integrar o novo módulo ao projeto atual de forma dinâmica.

Outra vantagem da opção de se trabalhar com HDL dentro do QUCS é que a partir do momento que o novo módulo está pronto em Verilog-A, o programa

durante o processo de compilação transforma o arquivo fonte, que está em HDL, em um novo código fonte traduzido em C++, já realizando as ligações de bibliotecas matemáticas e físicas, que são necessárias para o funcionamento correto do módulo, ao passo que para o desenvolvimento direto em C++ seria preciso um domínio desses códigos já existentes.

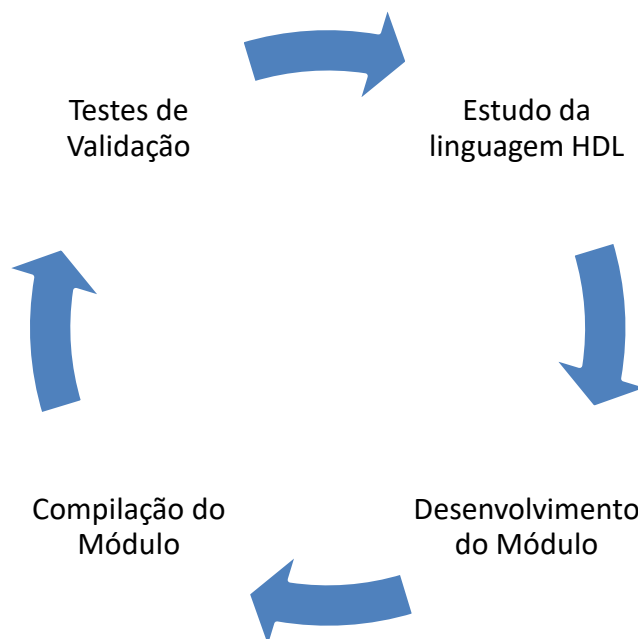


Figura 14 - Modelo para Desenvolvimento do Módulo por HDL

Modelo de desenvolvimento que representa apenas o desenvolvimento do módulo dentro do QUCS.

Esta estratégia implica apenas em domínio sobre o Verilog-A. Tem a vantagem de a própria ferramenta gerar automaticamente o código fonte em C++. Na Figura 15 é possível verificar os arquivos criados de forma automática e no Apêndice A tem-se o arquivo fonte do módulo Solo em linguagem C++. Futuramente, após o desenvolvimento de um número adequado de módulos em Verilog-A, poder-se-á aproveitar esses arquivos fontes criados para uma nova versão do QUCS. Para tanto, será necessário compilar todos os módulos, que se integrarão de forma nativa ao QUCS. Definir o número adequado de módulos será um problema futuro, que dependerá da experiência do grupo de trabalho.

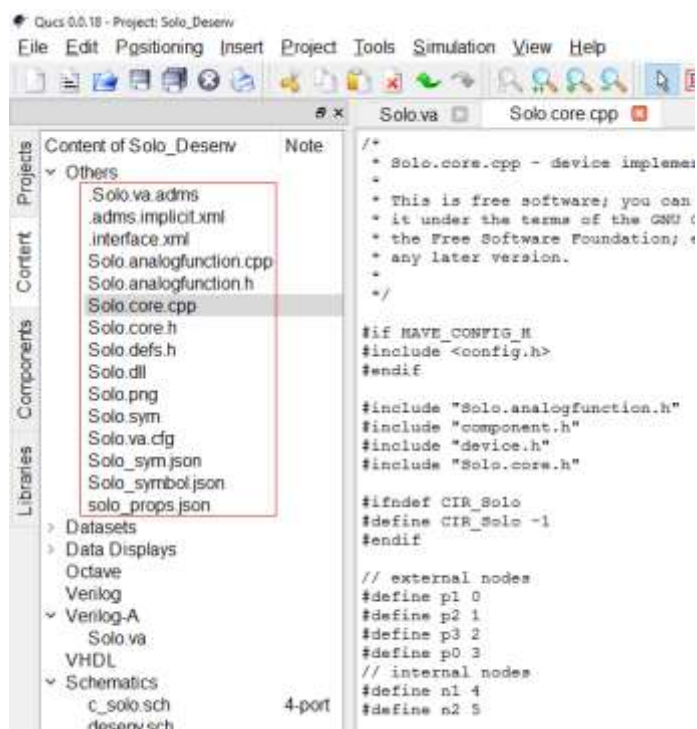


Figura 15 - Arquivos em C++ criados automaticamente

É possível identificar os arquivos que foram criados automaticamente, após a compilação do código em Verilog-A.

4. RESULTADOS

Os resultados obtidos da pesquisa e do desenvolvimento foram muito promissores, uma vez que a camada de desenvolvimento para criação do módulo foi toda feita no próprio ambiente de simulação o que a torna muito prática para qualquer alteração ou melhoria que o pesquisador queira fazer no módulo Solo. Nota-se, na Figura 16, o módulo solo está totalmente integrado ao ambiente de simulação do QUCS, sendo possível sua utilização de forma nativa no programa.

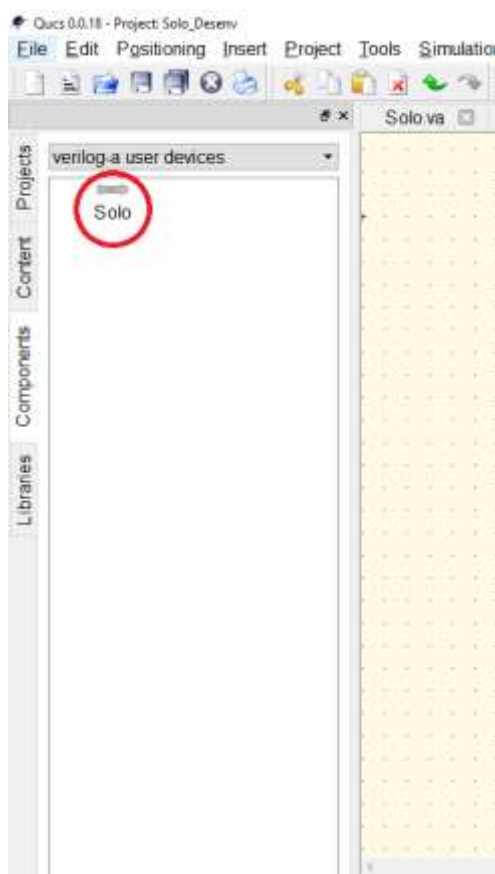


Figura 16 - Módulo solo integrado ao QUCS

O círculo vermelho enfatiza o modelo desenvolvido, compilado e já integrado ao QUCS.

4.1 MÓDULO SOLO

O desenvolvimento do módulo ocorreu de acordo com o fluxograma apresentado na Figura 14 e utiliza as bibliotecas “disciplines.vams” e “constants.vams”, que servem para declarar nós analógicos, portas, constantes

matemáticas e físicas, as quais devem ser incluídas na codificação. A palavra reservada include iniciada por apóstrofe indica para o compilador Verilog-A a inclusão de uma biblioteca conforme o Código Fonte 2.

```
`include "disciplines.vams"
`include "constants.vams"
```

Figura 17 - Código Fonte 2 - Utilização das Bibliotecas em Verilog-A

Analizando o modelo da Figura 12, o qual representa o solo em um esquemático de analogia a circuitos elétricos, se pode notar que nada mais é que um conjunto de resistores simples e um capacitor. A implementação parte então para desenvolver esses dois componentes, claro que obedecendo à sequência proposta por Gaio (2011) em seu esquema Solo.

```
I(p1,p2)<+V(p1,p2)/(resOhm);
I(p2,p1)<+V(p2,p1)/(resOhm);
I(p1,n1)<+V(p1,n1)/(resOhm);
I(n1,p0)<+V(n1,p0)/(resCap);
I(p0,n1)<+V(p0,n1)/(resCap);
I(n2)<+ddt(V(n2)*(capF));
```

Figura 18 - Código Fonte 3 - Codificação das Resistências e dos Capacitores

Na Figura 19, tem-se a interface de edição das variáveis e o trecho de Código Fonte 4 traz a implementação em Verilog-A:

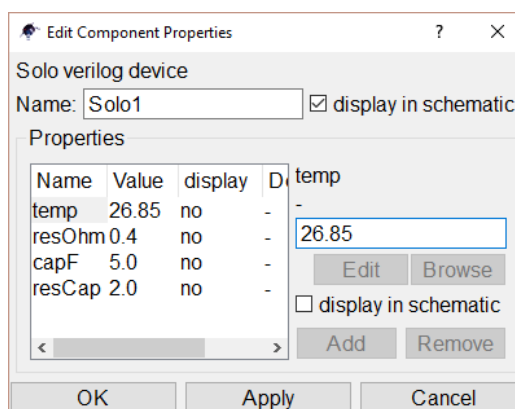


Figura 19 - Propriedade de edição do Solo

Outro ponto a ser explorado pelo desenvolvimento deste módulo é a possibilidade de edição em tempo real das variáveis que compõem essas resistências e a capacitância, do mesmo modo como é feito em um componente nativo do QUCS, porém agora em um módulo desenvolvido para as pesquisas ambientais.

```
parameter real temp = 26.85;
parameter real resOhm = 0.4;
parameter real capF = 5.0;
parameter real resCap = 2.0;
```

Figura 20 - Código Fonte 4 - Declaração das Variáveis Editáveis

O módulo desenvolvido, mostrado na Figura 22, possui também portas de entrada e saída. Faz-se necessário codificar essas conexões na ferramenta. Para isso, deve-se identifica-las como tipo de sinal elétrico e como entrada e saída, conforme o Código Fonte 5:

```
module Solo(p1, p2, p3, p0);
inout p1, p2, p3, p0;
electrical p0, p1, p2, p3, n1,
```

Figura 21 - Código Fonte 5 - Declaração do Nome do Módulo, Entradas e Saídas

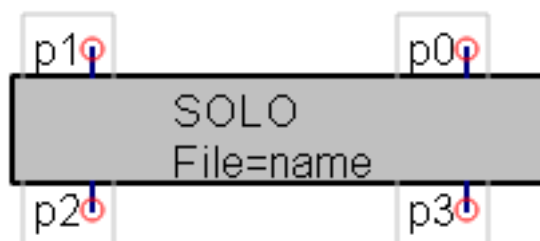


Figura 22 – Módulo Solo desenvolvido em Verilog-A

Representação gráfica do módulo desenvolvido em Verilog-A, é possível identificar as portas de entrada/saída pelos símbolos p_n .

Após todos esses passos é possível finalizar o código do módulo Solo e compilar o mesmo para que ele passe a fazer parte da biblioteca de componentes do projeto que está aberto no QUCS, conforme o Código Fonte 6.

```

`include "disciplines.vams"
`include "constants.vams"
module Solo(p1, p2, p3, p0);
inout p1, p2, p3, p0;
electrical p0, p1, p2, p3, n1, n2;
parameter real temp = 26.85;
parameter real resOhm = 0.4;
parameter real capF = 5.0;
parameter real resCap = 2.0;
analog begin
I(p1,p2)<+V(p1,p2)/(resOhm);
I(p2,p1)<+V(p2,p1)/(resOhm);
I(p1,n1)<+V(p1,n1)/(resOhm);
I(n1,p3)<+V(n1,p3);
I(n1,p0)<+V(n1,p0)/(resCap);
I(p0,n1)<+V(p0,n1)/(resCap);
I(n2)<+ddt(V(n2)*(capF));
I(n2,p0)<+V(n2,p0)/(resCap);
I(n2,p3)<+V(n2,p3);
I(n2,p2)<+V(n2,p2)/(resCap*2);
end
endmodule

```

Figura 23 - Código Fonte 6 – Módulo Solo em Verilog-A

4.2 UTILIZAÇÃO DO MÓDULO DESENVOLVIDO

Conforme apresentado na Figura 16, o módulo desenvolvido, agora integrado de forma nativa à ferramenta de simulação, está pronto para o uso. A partir disso, foi construído o modelo de simulação conforme a Figura 11, pois foi deste modelo que se deu a necessidade de desenvolvimento do módulo Solo.

A Figura 24 mostra como ficou o modelo já com a utilização do novo módulo desenvolvido, nota-se que ficou muito semelhante ao modelo inicial, e isso foi proposital, pois se buscou durante a criação a manutenção da fidelidade ao original para fins de teste de validação de seus resultados e com isto foi possível construir o módulo da maneira mais fiel possível.

É possível agora realizar a edição das propriedades dos componentes Solo, apenas clicando duas vezes sobre cada um deles e seu comportamento é independente entre eles. No modelo da Figura 10 toda a vez que era necessária alteração de alguma propriedade tinha que se realizar a edição de cada esquemático

separadamente para cada uma das camadas *Soil*, ou então, se todas as referências forem para o mesmo esquemático seria impossível trabalhar de forma única com cada camada.

Agora, qualquer simulação realizada com esse componente desenvolvido será nativo ao software de simulação.

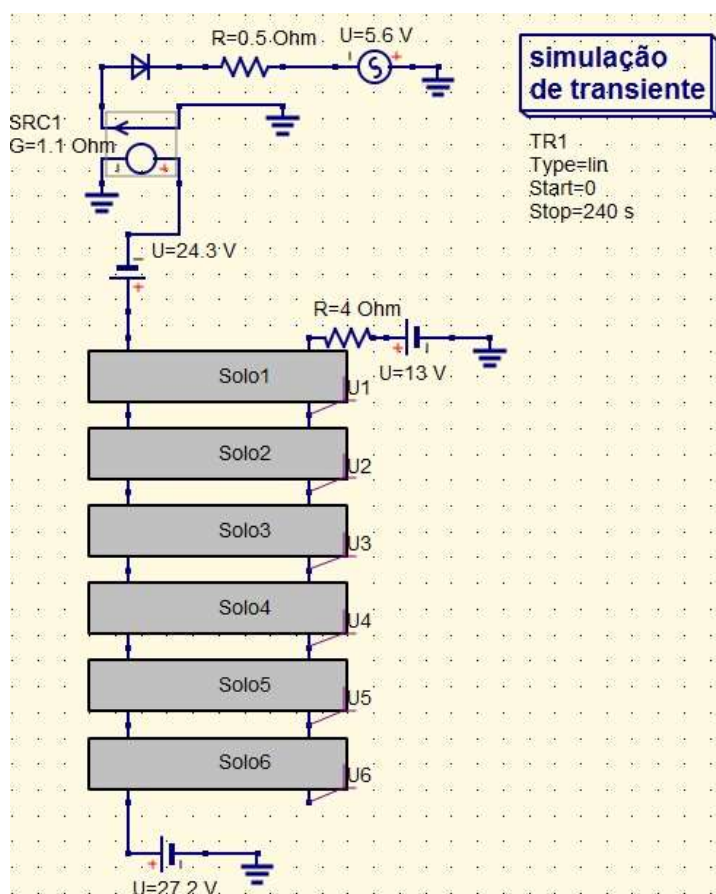


Figura 24 - Esquemático de simulação

Esquemático de simulação do fluxo de calor no solo já com a utilização do módulo solo desenvolvido em Verilog-A. Contém 6 camadas de solo e 6 sondas que atuam como termômetros para medição das temperaturas nas camadas.

4.3 SIMULAÇÃO DE UM PERFIL DE SOLO

Após o desenvolvimento e construção do novo componente, realizou-se uma simulação para o fluxo de calor no solo com o objetivo de se obter curvas de temperatura no perfil do solo, em diferentes instantes, comparáveis aos metadados.

O componente nativo “Solo”, agora, substitui o módulo no esquema inicial da Figura 10. A simulação obedece aos mesmos parâmetros. Tem-se um fluxo de calor transiente entre as diferentes camadas.

A temperatura de cada camada é uma variável de estado, proporcional a sua energia interna, que é análoga ao potencial elétrico. Deste modo, fixar o potencial elétrico mínimo em 24.3 V é equivalente a fixar a temperatura mínima do solo em 24.3 °C. De forma semelhante na última camada, a temperatura será fixa em 27.2 °C.

As sondas U_n medem a temperatura de cada uma das camadas e, portanto, comparam-se aos termômetros com seus bulbos nos respectivos pontos de medida.

O gráfico da Figura 25 mostra o comportamento térmico dos diferentes pontos do perfil do solo ao longo de um dia. A forma senoidal das curvas, deve-se, logicamente, à fonte de energia (radiação solar) ser uma fonte senoidal, como é visto na Figura 26.

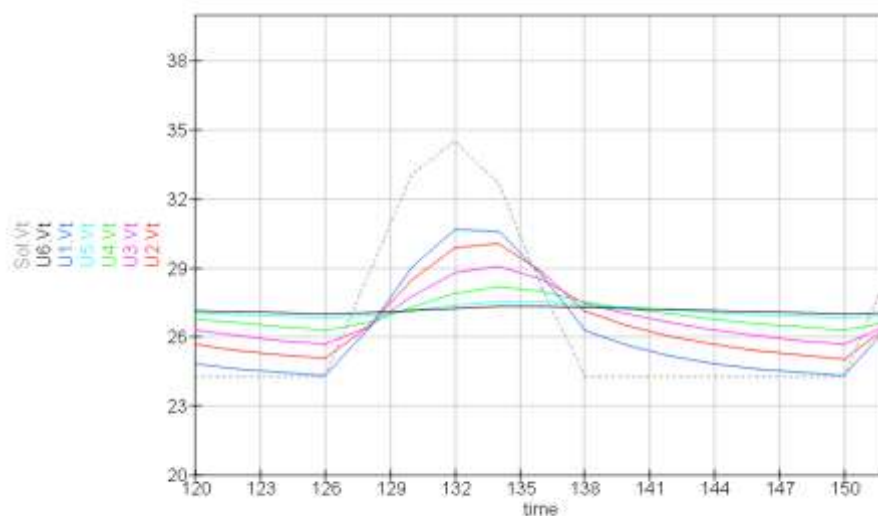


Figura 25 - Comportamento da simulação

A escala à esquerda refere-se as linhas coloridas, as quais apresentam as temperaturas das sondas (termômetros) nas diferentes profundidades. A linha pontilhada (gráfico da temperatura do ar a superfície) demonstra a variação da intensidade solar no tempo.

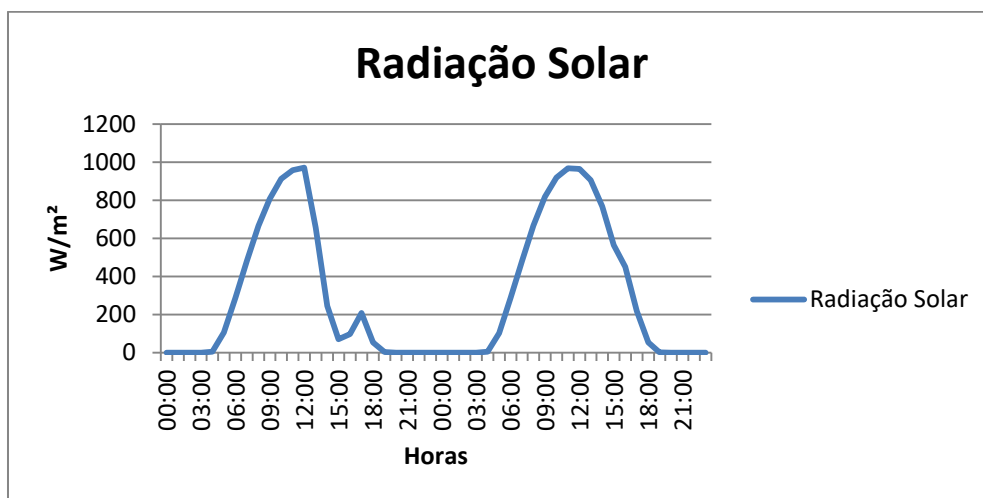


Figura 26 - Gráfico da Radiação Solar

Esta figura representa a variação da radiação solar que está atingindo o solo em um intervalo de 2 dias consecutivos.

Fonte: Estação de Crossroads, Novo México – EUA.

A variação encontrada é realmente a esperada. A sonda U1 apresentou a maior variabilidade térmica, pois ela está mais próxima à camada do solo onde incidem diretamente os raios solares e, com isso, a variação de temperatura é grande durante todo o dia. A sonda U6 apresentou a menor variabilidade térmica, essa sonda está localizada na camada mais profunda. O fluxo de calor reduz sua intensidade para o interior do solo, o que está de acordo com os dados experimentais. O modelo proposto, neste sentido, consegue simular com simplicidade uma característica fundamental no perfil de temperaturas do solo que é a assimetria das curvas, isto é, o valor médio na superfície não é coincidente com o valor fixo no ponto mais profundo. A Figura 21 demonstra os resultados de temperatura a cada uma das profundidades e as diferenças dessas temperaturas de acordo com a variação do tempo.

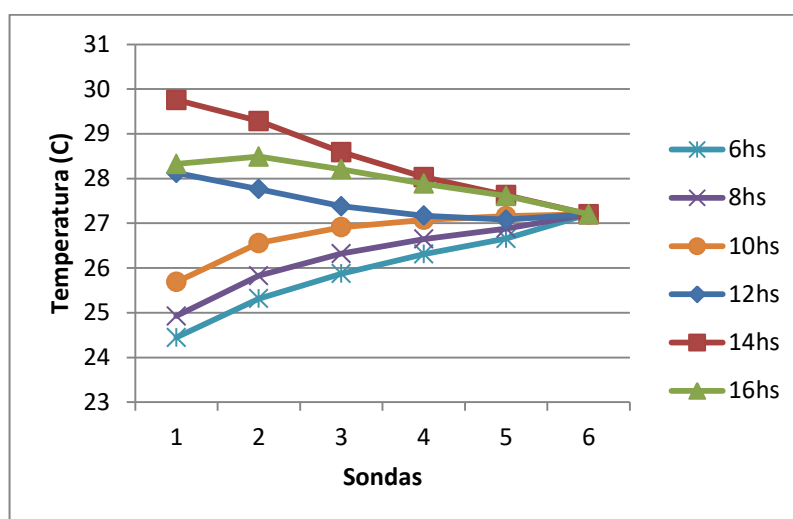


Figura 27 - Temperaturas por profundidade

As curvas de temperatura do modelo de simulação de acordo com a profundidade de cada sonda ao longo de um dia.

É possível, ainda, realizar a comparação desses resultados com alguns resultados encontrados na Estação de Crossroads, Novo México – EUA (Estados Unidos da América), latitude 33.54° e longitude -103.24° , conforme a Figura 21 mostra, a simulação apresenta o mesmo comportamento dos dados coletados em campo.

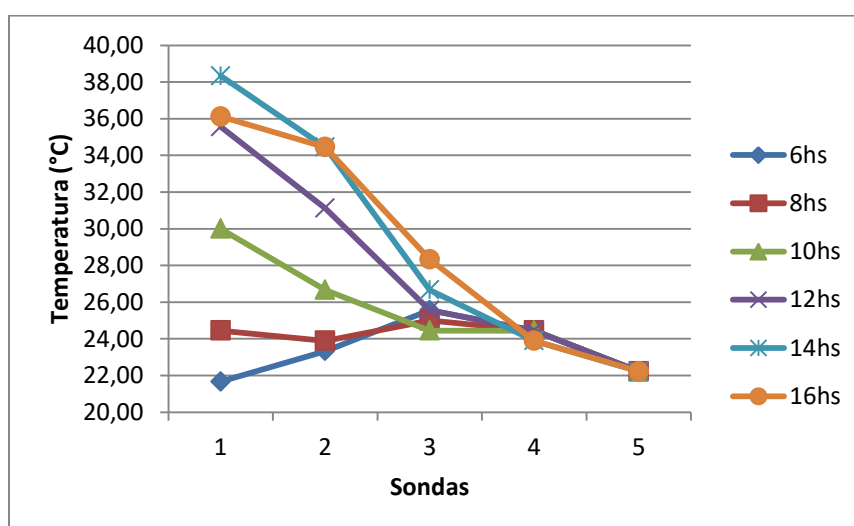


Figura 28 - Temperaturas por profundidade 2

Curvas de temperatura de acordo com os termômetros em diferentes profundidades.

Fonte: Estação de Crossroads, Novo México – EUA.

5. CONCLUSÃO

Com este trabalho inicial a alteração do software livre se mostrou muito promissora para sua completa adequação para utilização em pesquisas na área ambiental.

O módulo proposto para ser desenvolvido e testado foi concluído com sucesso e seus testes mostraram os resultados esperados, com as vantagens desejadas, isto é, o módulo Solo desenvolvido foi altamente aderente ao modelo original, com o qual a modelagem era muito mais trabalhosa e a manutenção muito difícil.

A construção do modelo atendeu as leis de transferência de calor. A alta variabilidade na superfície e a não variabilidade na sonda de profundidade, bem como as curvas horárias de temperatura no perfil, que demonstraram comportamento semelhante ao de dados experimentais. O modelo proposto, nesse sentido, consegue simular com simplicidade uma característica fundamental no perfil de temperaturas do solo que é a assimetria das curvas, isto é, o valor médio na superfície não é coincidente com o valor fixo no ponto mais profundo. Essa característica deve-se à inércia térmica do material do solo e não é bem tratada pela Equação 7. Foi possível ainda eliminar algumas interferências externas do meio elétrico, como o ruído branco, que não fazem sentido quando utilizados com variáveis ambientais.

O trabalho feito de forma experimental abordou o desenvolvimento de um novo componente para trabalhar de forma nativa no ambiente do QUCS, porém verificou que é possível também trabalhar com o desenvolvimento em uma linguagem de mais alto nível, como o C++, assim sendo possível gerar uma distribuição nova que poderia ser nomeada diretamente como Simulações de Variáveis Ambientais, não sendo mais necessário trabalhar de forma análoga a circuitos elétricos.

Em um trabalho futuro deseja-se dar continuidade no desenvolvimento das variáveis ambientais em linguagem de descrição de hardware, e tentar trabalhar com todo o contínuo solo-planta-atmosfera, conforme Gaio (2011) propôs, acoplando, inclusive, as transferências de energia causadas pelos diferentes fluxos (por exemplo, água, calor e CO₂).

Para perspectivas futuras é interessante dar continuidade ao desenvolvimento da ferramenta de forma a atender as demandas ambientais não somente do Programa de Pós Graduação em Física Ambiental, mas também toda a comunidade acadêmica e profissional que se dedica a testes e experimentações que envolvam variáveis ambientais.

6. REFERÊNCIAS BIBLIOGRÁFICAS

- ARAGUAIA, M. (2005). **"Constituição do solo": Brasil Escola**. Acesso em Março 29, 2016, from <http://brasilecola.uol.com.br/biologia/constuicao-solo.htm>
- AZEVEDO, Tarik Rezende., Galvani, Emerson. **Ajuste do ciclo médio mensal horário da temperatura do solo em função da temperatura do ar**. Revista Brasileira de Agrometeorologia, Santa Maria, v. 11, p.123-130. 2003
- BASTIAANSEN, W.G.M. **Regionalization of surface flux densities and moisture indicators in composite terrain**. Doctoral thesis, Wageningen Agricultural University, Wageningen The Netherlands. 273p. 1995.
- CAPECHE, C. L. (2008, Dezembro). **Noções sobre tipos de estrutura do solo e sua**. *Comunicado Técnico 51*, pp. 1-6.
- CAMPBELL, Gaylon S., NORMAN, John M. (1998). **An Introduction to Environmental Biophysics**. Segunda Edição. New York: Springer – Verlag.
- CHU, Pong P. (2006). **RTL Hardware Desing Using VHDL**. New Jersey: John Wiley & Sons, Inc.
- COWAN, I.R. **An Electrical Analogue of Evaporation from, and Flow of Water in Plants**. *Planta*, 106: 221-226. 1972
- E. GIL-PELEGRÍN, I. A.-P. (2005). **El continuo suelo-planta-atmósfera como un modelo integrador**. *Investigación agraria. Sistemas y recursos forestales*, 358-370.
- Embrapa Informação Tecnológica. (2011). **Irrigação e fertirrigação em fruteiras e hortaliças**. Brasília.
- FARIA, M. A. (n.d.). **Manejo da Irrigação**.
- FERREIRA, Mozart M. **Caracterização Física do Solo**. In: Física do Solo. Editor: Quirijn de Jong van Lier. Viçosa – MG: Sociedade Brasileira de Ciência do Solo. 2010.
- GAIO, D.C. **Análogo elétrico da dinâmica hídrica em plantas vasculares**. Cuiabá. 2011. 104 f. Tese (Doutorado em Física Ambiental), Instituto de Física, Universidade Federal de Mato Grosso.
- GAIO, D.C., da Cunha, J. M., Silva, D. M., Campos, M. C., França, S. P., & da Silva, L. I. (2015). **Estoque de carbono e atributos físicos do solo em áreas**

de Terra Preta Arqueológica no Sul do Amazonas. XXXV Congresso Brasileiro de Ciência do Solo - O Solo e Suas Múltiplas Funções. Natal.

HEINRICHS, R. (2010). **Aula 6.** Acessado em Abril 19, 2016, from Unesp Dracena: http://www2.dracena.unesp.br/graduacao/arquivos/solos/aula_6_estrutura_do_solo.pdf

HILLEL, D. (1998). **Environmental soil physics.** San Diego: Academic Press.

JOHANSSON A.; PRYTZ G. **Period Doubling of Rhythmic Water Regulation in Plants. In: Biological Rhythms.** Editor: Vinod Kumar. New Delhi: Narosa Publishing House. 2002.

LACERDA, Claudivan Feitosa. **Relações Solo-Água-Planta em Ambientes Naturais e Agrícolas do Nordeste Brasileiro.** Universidade Federal Rural de Pernambuco. Recife. 2007

LANG, A. et al. **Leaf Water Balance During Oscillation of Stomatal Aperture.** Plant Physiol. 44. 1969. pp 826-830.

LIER, Quirijn de Jong van.; **Física do Solo.** Viçosa – MG: Sociedade Brasileira de Ciência do Solo. 2010.

LIPOVSKI, G. J. (1977) **Hardware Description Languages: Voices from the Tower of Babel.** Texas.

LOPES, H. L., Teixeira, A. H., Accioly, L. d., Sobral, M. d., & Montenegro, S. G. (2011). **Relações do fluxo de calor no solo com o uso e cobertura das terras na bacia hidrográfica do. Anais XV Simpósio Brasileiro de Sensoriamento Remoto - SBSR.** Curitiba.

NOBEL, S. Park (2009) ; **Physicochemical and Environmental Plant Physiology.** Quarta Edição. Canadá: Editora Elsevier.

OMETTO, José Carlos (1981). **Bioclimatologia Vegetal.** São Paulo: Ed. Agronômica Ceres.

PALNITKAR, Samir (2013). **Verilog HDL: A Guide to Digital Design and Synthesis.** Segunda Edição. Nova Jersey: Prentice Hall PTR.

PEREIRA, V.M.R.; **Análogo Elétrico da Pressão de Turgescência em Células Guarda de Vicia Faba: Modelo Científico e Conceitual.** Cuiabá. 2013. 66 f. Dissertação (Mestrado em Física Ambiental), Instituto de Física, Universidade Federal de Mato Grosso.

REICHARDT, K., & TIMM, L. C. (2004). **Solo, Planta e Atmosfera: conceitos, processos e aplicações.** Barueri: Editora Manole Ltda.

- REINERT, D. J., & REICHERT, J. M. (2006, Maio). **Propriedades física do solo**. Santa Maria, Rio Grande do Sul, Brasil.
- SIQUEIRA, W.C.L.; **Modelagem das Vias de Transporte Hídrico da Raiz em Plantas Vasculares – Análogo Elétrico**. Cuiabá. 2014. 64 f. Dissertação (Mestrado em Física Ambiental), Instituto de Física, Universidade Federal de Mato Grosso.
- TULLER, Markus., OR, Dani.; **Vadose Zone Hydrology/Soil Physics**. Editora: Spring. 2004.
- VIANELLO, R.L., ALVES, A.R. **Meteorologia Básica e Aplicações**. Viçosa: Imprensa Universitária, Universidade Federal de Viçosa, 1991. 449 p.
- WESTE, NEIL H. E., & HARRIS, D. M. (2011). **CMOS VLSI Design A Circuits and Systems Perspective**. Quarta Edição. Boston: Addison-Wesley.
- ZUCARELI, V.; AMARO, A. C. E.; SILVÉRIO, E. V.; FERREIRA, G. **Métodos de superação da dormência e temperatura na germinação de sementes de Dioclea violácea**. Semina: Ciências Agrárias, Londrina, v. 31, p. 1305-1312, 2010. Suplemento 1.

APÊNDICE A

```

/*
 * Solo.core.cpp - device implementations for Solo module
 *
 * This is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 */

#if HAVE_CONFIG_H
#include <config.h>
#endif

#include "Solo.analogfunction.h"
#include "component.h"
#include "device.h"
#include "Solo.core.h"

#ifndef CIR_Solo
#define CIR_Solo -1
#endif

// external nodes
#define p1 0
#define p2 1
#define p3 2
#define p0 3
// internal nodes
#define n1 4
#define n2 5

// useful macro definitions
#define NP(node) real (getV (node))
#define BP(pnode,nnode) (NP(pnode) - NP(nnode))
#define _load_static_residual2(pnode,nnode,current)\
    _rhs[pnode] -= current;\
    _rhs[nnode] += current;
#define _load_static_augmented_residual2(pnode,nnode,current)\
    _rhs[pnode] -= current;\
    _rhs[nnode] += current;
#define _load_static_residual1(node,current)\
    _rhs[node] -= current;
#define _load_static_augmented_residual1(node,current)\
    _rhs[node] -= current;
#define _load_static_jacobian4(pnode,nnode,vpnode,vnnode,conductance)\

```

```

        _jstat[pnode][vpnode] += conductance;\
        _jstat[nnode][vnnode] += conductance;\
        _jstat[pnode][vnnode] -= conductance;\
        _jstat[nnode][vpnode] -= conductance;\
        if (doHB) {\
            _ghs[pnode] += conductance * BP(vpnode,vnnode);\
            _ghs[nnode] -= conductance * BP(vpnode,vnnode);\
        } else {\
            _rhs[pnode] += conductance * BP(vpnode,vnnode);\
            _rhs[nnode] -= conductance * BP(vpnode,vnnode);\
        }
    }
#define _load_static_jacobian2p(node,vpnode,vnnode,conductance)\
    _jstat[node][vpnode] += conductance;\
    _jstat[node][vnnode] -= conductance;\
    if (doHB) {\
        _ghs[node] += conductance * BP(vpnode,vnnode);\
    } else {\
        _rhs[node] += conductance * BP(vpnode,vnnode);\
    }
#define _load_static_jacobian2s(pnode,nnode,node,conductance)\
    _jstat[pnode][node] += conductance;\
    _jstat[nnode][node] -= conductance;\
    if (doHB) {\
        _ghs[pnode] += conductance * NP(node);\
        _ghs[nnode] -= conductance * NP(node);\
    } else {\
        _rhs[pnode] += conductance * NP(node);\
        _rhs[nnode] -= conductance * NP(node);\
    }
#define _load_static_jacobian1(node,vnode,conductance)\
    _jstat[node][vnode] += conductance;\
    if (doHB) {\
        _ghs[node] += conductance * NP(vnode);\
    } else {\
        _rhs[node] += conductance * NP(vnode);\
    }
#define _load_dynamic_residual2(pnode,nnode,charge)\
    if (doTR) _charges[pnode][nnode] += charge;\
    if (doHB) {\
        _qhs[pnode] -= charge;\
        _qhs[nnode] += charge;\
    }
#define _load_dynamic_residual1(node,charge)\
    if (doTR) _charges[node][node] += charge;\
    if (doHB) {\
        _qhs[node] -= charge;\
    }
#define _load_dynamic_jacobian4(pnode,nnode,vpnode,vnnode,capacitance)\
    if (doAC) {\
        _jdyna[pnode][vpnode] += capacitance;\

```

```

        _jdyna[nnode][vnnode] += capacitance;\
        _jdyna[pnode][vnnode] -= capacitance;\
        _jdyna[nnode][vpnode] -= capacitance;\
    }\
    if (doTR) {\
        _caps[pnode][nnode][vpnode][vnnode] += capacitance;\
    }\
    if (doHB) {\
        _chs[pnode] += capacitance * BP(vpnode,vnnode);\
        _chs[nnode] -= capacitance * BP(vpnode,vnnode);\
    }\
#define _load_dynamic_jacobian2s(pnode,nnode,vnode,capacitance)\
    if (doAC) {\
        _jdyna[pnode][vnode] += capacitance;\
        _jdyna[nnode][vnode] -= capacitance;\
    }\
    if (doTR) {\
        _caps[pnode][nnode][vnode][vnode] += capacitance;\
    }\
    if (doHB) {\
        _chs[pnode] += capacitance * NP(vnode);\
        _chs[nnode] -= capacitance * NP(vnode);\
    }\
#define _load_dynamic_jacobian2p(node,vpnode,vnnode,capacitance)\
    if (doAC) {\
        _jdyna[node][vpnode] += capacitance;\
        _jdyna[node][vnnode] -= capacitance;\
    }\
    if (doTR) {\
        _caps[node][node][vpnode][vnnode] += capacitance;\
    }\
    if (doHB) {\
        _chs[node] += capacitance * BP(vpnode,vnnode);\
    }\
#define _load_dynamic_jacobian1(node,vnode,capacitance)\
    if (doAC) {\
        _jdyna[node][vnode] += capacitance;\
    }\
    if (doTR) {\
        _caps[node][node][vnode][vnode] += capacitance;\
    }\
    if (doHB) {\
        _chs[node] += capacitance * NP(vnode);\
    }\

#define _save_whitenoise1(n1,pwr,type)\
    _white_pwr[n1][n1] += pwr;
#define _save_whitenoise2(n1,n2,pwr,type)\
    _white_pwr[n1][n2] += pwr;
#define _save_flickernoise1(n1,pwr,exp,type)\

```

```

        _flicker_pwr[n1][n1] += pwr;\
        _flicker_exp[n1][n1] += exp;
#define _save_flickernoise2(n1,n2,pwr,exp,type)\
        _flicker_pwr[n1][n2] += pwr;\
        _flicker_exp[n1][n2] += exp;
#define _load_whitenoise2(n1,n2,pwr)\
        cy (n1,n2) -= pwr/kB/T0; cy (n2,n1) -= pwr/kB/T0;\
        cy (n1,n1) += pwr/kB/T0; cy (n2,n2) += pwr/kB/T0;
#define _load_whitenoise1(n1,pwr)\
        cy (n1,n1) += pwr/kB/T0;
#define _load_flickernoise2(n1,n2,pwr,exp)\
        cy (n1,n2) -= pwr*pow(_freq,-exp)/kB/T0;\
        cy (n2,n1) -= pwr*pow(_freq,-exp)/kB/T0;\
        cy (n1,n1) += pwr*pow(_freq,-exp)/kB/T0;\
        cy (n2,n2) += pwr*pow(_freq,-exp)/kB/T0;
#define _load_flickernoise1(n1,pwr,exp)\
        cy (n1,n1) += pwr*pow(_freq,-exp)/kB/T0;

// derivative helper macros
// transcendental LRM p. 59
#define m00_cos(v00,x)      v00 = cos(x);
#define m10_cos(v10,v00,x)  v10 = (-sin(x));
#define m00_sin(v00,x)      v00 = sin(x);
#define m10_sin(v10,v00,x)  v10 = (cos(x));
#define m00_tan(v00,x)      v00 = tan(x);
#define m10_tan(v10,v00,x)  v10 = (1.0/cos(x)/cos(x));
#define m00_cosh(v00,x)     v00 = cosh(x);
#define m10_cosh(v10,v00,x) v10 = (sinh(x));
#define m00_sinh(v00,x)     v00 = sinh(x);
#define m10_sinh(v10,v00,x) v10 = (cosh(x));
#define m00_tanh(v00,x)     v00 = tanh(x);
#define m10_tanh(v10,v00,x) v10 = (1.0/cosh(x)/cosh(x));
#define m00_acos(v00,x)     v00 = acos(x);
#define m10_acos(v10,v00,x) v10 = (-1.0/sqrt(1-x*x));
#define m00_asin(v00,x)     v00 = asin(x);
#define m10_asin(v10,v00,x) v10 = (+1.0/sqrt(1-x*x));
#define m00_atan(v00,x)     v00 = atan(x);
#define m10_atan(v10,v00,x) v10 = (+1.0/(1+x*x));
#define m00_hypot(v00,x,y)  v00 = sqrt((x)*(x)+(y)*(y));
#define m10_hypot(v10,v00,x,y) v10 = (x)/(v00);
#define m11_hypot(v11,v00,x,y) v11 = (y)/(v00);
#define m00_atan2(v00,x,y)  v00 = atan2(x,y);
// TODO atan2 derivatives ?
#define m00_acosh(v00,x)    v00 = acosh(x);
#define m10_acosh(v10,v00,x) v10 = (1.0/(sqrt(x-1)*sqrt(x+1)));
#define m00_asinh(v00,x)    v00 = asinh(x);
#define m10_asinh(v10,v00,x) v10 = (1.0/(sqrt(x*x+1)));
#define m00_atanh(v00,x)    v00 = atanh(x);
#define m10_atanh(v10,v00,x) v10 = (1.0/(1-x*x));

```

```

// standard functions LRM p.58
#define m00_logE(v00,x)    v00 = log(x);
#define m10_logE(v10,v00,x) v10 = (1.0/x);
#define m00_log10(v00,x)   v00 = log10(x);
#define m10_log10(v10,v00,x) v10 = (1.0/x/M_LN10);
#define m00_exp(v00,x)     v00 = exp(x);
#define m10_exp(v10,v00,x) v10 = v00;
#define m00_sqrt(v00,x)    v00 = sqrt(x);
#define m10_sqrt(v10,v00,x) v10 = (0.5/v00);
#define m00_min(v00,x,y)   v00 = ((x)<(y))? (x):(y);
#define m10_min(v10,v00,x,y) v10 = ((x)<(y))? 1.0:0.0;
#define m11_min(v11,v00,x,y) v11 = ((x)<(y))? 0.0:1.0;
#define m00_max(v00,x,y)   v00 = ((x)>(y))? (x):(y);
#define m10_max(v10,v00,x,y) v10 = ((x)>(y))? 1.0:0.0;
#define m11_max(v11,v00,x,y) v11 = ((x)>(y))? 0.0:1.0;
#define m00_pow(v00,x,y)   v00 = pow(x,y);
#define m10_pow(v10,v00,x,y) v10 = (x==0.0)? 0.0:(v00)*(y)/(x);
#define m11_pow(v11,v00,x,y) v11 = (x==0.0)? 0.0:(log(x)*(v00));
#define m00_abs(v00,x)     v00 = ((x)<0)? (-x):(x);
#define m10_abs(v10,v00,x) v10 = (((x)>=0)? (+1.0):(-1.0));
#define m00_floor(v00,x)   v00 = floor(x);
#define m10_floor(v10,v00,x) v10 = 1.0;

#define m00_ceil(v00,x)    v00 = ceil(x);
// TODO ceil derivative, needed?

// analog operator, LRM p.61
#define m00_limexp(v00,x)   v00 = ((x)<80.0?exp(x):exp(80.0)*(x-79.0));
#define m10_limexp(v10,v00,x) v10 = ((x)<80.0?(v00):exp(80.0));

// analog kernel parameter system functions, LRM p.215
#define m00_vt(x)           (kBoverQ*(x))
#define m10_vt(x)           (kBoverQ)

// extra functions (?)
#define m00_div(v00,v10,x,y) double v10=1/(y); double v00=(x)*v10;
#define m10_div(v10,v00,vv,x,y)
#define m11_div(v11,v00,vv,x,y) double v11 = -v00*vv;
#define m00_mult(v00,v10,v11,x,y) double v10=(x); double v11=(y); double v00=v10*v11;
#define m00_add(v00,x,y)    double v00=(x)+(y);

// second derivatives
#define m20_logE(v00)       (-1.0/v00/v00)
#define m20_exp(v00)        exp(v00)
#define m20_limexp(v00)     ((v00)<80.0?exp(v00):0.0)
#define m20_sqrt(v00)       (-0.25/(v00)/sqrt(v00))
#define m20_abs(v00)         0.0
#define m20_pow(x,y)         ((y)*((y)-1.0)*pow(x,y)/(x)/(x))

```

```

// simulator specific definitions
#define _modelname      "Solo"
#define _instancename   getName()
#define _circuit_temp   (getPropertyDouble("Temp")+273.15)
#define _param_given(p) (isPropertyGiven(p)?1:0)

// $vt and $vt() functions
#define _vt_nom          (kBoverQ*_circuit_temp)

using namespace qucs::device;
using qucs::matrix;

/* Device constructor. */
Solo::Solo() : circuit (6)
{
    type = CIR_Solo;
}

/* Initialization of model. */
void Solo::initModel (void)
{
    // create internal nodes
    setInternalNode (n1, "n1");
    setInternalNode (n2, "n2");

    // get device model parameters
    loadVariables ();
    // evaluate global model equations
    initializeModel ();
    // evaluate initial step equations
    initialStep ();
    // evaluate global instance equations
    initializeInstance ();
}

/* Initialization of DC analysis. */
void Solo::initDC (void)
{
    allocMatrixMNA ();
    initModel ();
    pol = 1;
    restartDC ();
    doAC = 1;
    doTR = 0;
    doHB = 0;
}

/* Run when DC is restarted (fallback algorithms). */

```

```

void Solo::restartDC (void)
{
}

/* Initialize Verilog-AMS code. */
void Solo::initVerilog (void)
{
    // initialization of noise variables

    int i1, i2, i3, i4;

    // zero charges
    for (i1 = 0; i1 < 6; i1++) {
        for (i2 = 0; i2 < 6; i2++) {
            _charges[i1][i2] = 0.0;
        }
    }

    // zero capacitances
    for (i1 = 0; i1 < 6; i1++) {
        for (i2 = 0; i2 < 6; i2++) {
            for (i3 = 0; i3 < 6; i3++) {
                for (i4 = 0; i4 < 6; i4++) {
                    _caps[i1][i2][i3][i4] = 0.0;
                }
            }
        }
    }

    // zero right hand side, static and dynamic jacobian
    for (i1 = 0; i1 < 6; i1++) {
        _rhs[i1] = 0.0;
        _qhs[i1] = 0.0;
        _chs[i1] = 0.0;
        _ghs[i1] = 0.0;
        for (i2 = 0; i2 < 6; i2++) {
            _jstat[i1][i2] = 0.0;
            _jdyna[i1][i2] = 0.0;
        }
    }
}

/* Load device model input parameters. */
void Solo::loadVariables (void)
{
    temp = getPropertyDouble ("temp");
    resOhm = getPropertyDouble ("resOhm");
    capF = getPropertyDouble ("capF");
    resCap = getPropertyDouble ("resCap");
}

/* #define's for translated code */
#undef _DDT
#define _DDT(q) q

```

```

#define _DYNAMIC
#define _DERIVATE
#define _DDX
#define _DERIVATEFORDDX

/* Evaluate Verilog-AMS equations in model initialization. */
void Solo::initializeModel (void)
{
}

/* Evaluate Verilog-AMS equations in instance initialization. */
void Solo::initializeInstance (void)
{
}

/* Evaluate Verilog-AMS equations in initial step. */
void Solo::initialStep (void)
{
}

/* Evaluate Verilog-AMS equations in final step. */
void Solo::finalStep (void)
{
}

/* Evaluate Verilog-AMS equations in analog block. */
void Solo::calcVerilog (void)
{
}

/* ----- evaluate verilog analog equations ----- */
_load_static_residual2(p1,p2,(BP(p1,p2)/resOhm));
#if defined(_DERIVATE)
_load_static_jacobian4(p1,p2,p1,p2,(1/resOhm));
#endif
_load_static_residual2(p2,p1,(BP(p2,p1)/resOhm));
#if defined(_DERIVATE)
_load_static_jacobian4(p2,p1,p2,p1,(1/resOhm));
#endif
_load_static_residual2(p1,n1,(BP(p1,n1)/resOhm));
#if defined(_DERIVATE)
_load_static_jacobian4(p1,n1,p1,n1,(1/resOhm));
#endif
_load_static_residual2(n1,p3,BP(n1,p3));
#if defined(_DERIVATE)
_load_static_jacobian4(n1,p3,n1,p3,1.0);
#endif
_load_static_residual2(n1,p0,(BP(n1,p0)/resCap));
#if defined(_DERIVATE)
_load_static_jacobian4(n1,p0,n1,p0,(1/resCap));
#endif

```

```

_load_static_residual2(p0,n1,(BP(p0,n1)/resCap));
#if defined(_DERIVATE)
_load_static_jacobian4(p0,n1,p0,n1,(1/resCap));
#endif
#if defined(_DYNAMIC)
_load_dynamic_residual1(n2,_DDT((NP(n2)*capF)));
#if defined(_DERIVATE)
_load_dynamic_jacobian1(n2,n2,(capF));
#endif
#endif
_load_static_residual2(n2,p0,(BP(n2,p0)/resCap));
#if defined(_DERIVATE)
_load_static_jacobian4(n2,p0,n2,p0,(1/resCap));
#endif
_load_static_residual2(n2,p3,BP(n2,p3));
#if defined(_DERIVATE)
_load_static_jacobian4(n2,p3,n2,p3,1.0);
#endif
_load_static_residual2(n2,p2,(BP(n2,p2)/(resCap*2)));
#if defined(_DERIVATE)
_load_static_jacobian4(n2,p2,n2,p2,(1/(resCap*2)));
#endif

/* ----- end of verilog analog equations ----- */

/* ----- evaluate verilog noise equations ----- */

/* ----- end of verilog noise equations ----- */
}

/* Perform DC iteration. */
void Solo::calcDC (void)
{
    // evaluate Verilog code
    initVerilog ();
    calcVerilog ();

    // fill right hand side and static jacobian
    for (int i1 = 0; i1 < 6; i1++) {
        setI (i1, _rhs[i1]);
        for (int i2 = 0; i2 < 6; i2++) {
            setY (i1, i2, _jstat[i1][i2]);
        }
    }
}

/* Save operating points. */
void Solo::saveOperatingPoints (void)
{
    // save global instance operating points

```

```

}

/* Load operating points. */
void Solo::loadOperatingPoints (void)
{
}

/* Calculate operating points. */
void Solo::calcOperatingPoints (void)
{
}

/* Initialization of AC analysis. */
void Solo::initAC (void)
{
    allocMatrixMNA ();
}

/* Perform AC calculations. */
void Solo::calcAC (nr_double_t frequency)
{
    setMatrixY (calcMatrixY (frequency));
}

/* Compute Y-matrix for AC analysis. */
matrix Solo::calcMatrixY (nr_double_t frequency)
{
    _freq = frequency;
    saveOperatingPoints ();
    matrix y (6);

    for (int i1 = 0; i1 < 6; i1++) {
        for (int i2 = 0; i2 < 6; i2++) {
            y (i1,i2) = nr_complex_t (_jstat[i1][i2], _jdyna[i1][i2] * 2 * M_PI * _freq);
        }
    }

    return y;
}

/* Initialization of S-parameter analysis. */
void Solo::initSP (void)
{
    allocMatrixS ();
}

/* Perform S-parameter calculations. */
void Solo::calcSP (nr_double_t frequency)
{
    setMatrixS (yos (calcMatrixY (frequency)));
}

```

```

}

/* Initialization of transient analysis. */
void Solo::initTR (void)
{
    setStates (2 * 6 * 6);
    initDC ();
}

/* Perform transient analysis iteration step. */
void Solo::calcTR (nr_double_t)
{
    doHB = 0;
    doAC = 1;
    doTR = 1;
    calcDC ();

    int i1, i2, i3, i4, state;

    // 2-node charge integrations
    for (i1 = 0; i1 < 6; i1++) {
        for (i2 = 0; i2 < 6; i2++) {
            state = 2 * (i2 + 6 * i1);
            if (i1 != i2)
                if (_charges[i1][i2] != 0.0)
                    transientCapacitanceQ (state, i1, i2, _charges[i1][i2]);
        }
    }

    // 1-node charge integrations
    for (i1 = 0; i1 < 6; i1++) {
        state = 2 * (i1 + 6 * i1);
        if (_charges[i1][i1] != 0.0)
            transientCapacitanceQ (state, i1, _charges[i1][i1]);
    }

    // charge: 2-node, voltage: 2-node
    for (i1 = 0; i1 < 6; i1++) {
        for (i2 = 0; i2 < 6; i2++) {
            if (i1 != i2)
                for (i3 = 0; i3 < 6; i3++) {
                    for (i4 = 0; i4 < 6; i4++) {
                        if (i3 != i4)
                            if (_caps[i1][i2][i3][i4] != 0.0)
                                transientCapacitanceC (i1, i2, i3, i4, _caps[i1][i2][i3][i4], BP(i3,i4));
                    }
                }
        }
    }

    // charge: 2-node, voltage: 1-node
    for (i1 = 0; i1 < 6; i1++) {
        for (i2 = 0; i2 < 6; i2++) {
            if (i1 != i2)

```

```

for (i3 = 0; i3 < 6; i3++) {
    if (_caps[i1][i2][i3][i3] != 0.0)
        transientCapacitanceC2Q (i1, i2, i3, _caps[i1][i2][i3][i3], NP(i3));
} } }

// charge: 1-node, voltage: 2-node
for (i1 = 0; i1 < 6; i1++) {
    for (i3 = 0; i3 < 6; i3++) {
        for (i4 = 0; i4 < 6; i4++) {
            if (i3 != i4)
                if (_caps[i1][i1][i3][i4] != 0.0)
                    transientCapacitanceC2V (i1, i3, i4, _caps[i1][i1][i3][i4], BP(i3,i4));
        } } }

// charge: 1-node, voltage: 1-node
for (i1 = 0; i1 < 6; i1++) {
    for (i3 = 0; i3 < 6; i3++) {
        if (_caps[i1][i1][i3][i3] != 0.0)
            transientCapacitanceC (i1, i3, _caps[i1][i1][i3][i3], NP(i3));
    } }
}

/* Compute Cy-matrix for AC noise analysis. */
matrix Solo::calcMatrixCy (nr_double_t frequency)
{
    _freq = frequency;
    matrix cy (6);

    return cy;
}

/* Perform AC noise computations. */
void Solo::calcNoiseAC (nr_double_t frequency)
{
    setMatrixN (calcMatrixCy (frequency));
}

/* Perform S-parameter noise computations. */
void Solo::calcNoiseSP (nr_double_t frequency)
{
    setMatrixN (cytocs (calcMatrixCy (frequency) * z0, getMatrixS ()));
}

/* Initialization of HB analysis. */
void Solo::initHB (int)
{
    initDC ();
    allocMatrixHB ();
}

```

```

/* Perform HB analysis. */
void Solo::calcHB (int)
{
    doHB = 1;
    doAC = 1;
    doTR = 0;

    // jacobian dI/dV and currents get filled
    calcDC ();
    saveOperatingPoints ();

    // fill in HB matrices
    for (int i1 = 0; i1 < 6; i1++) {
        setQ (i1, _qhs[i1]); // charges
        setCV (i1, _chs[i1]); // jacobian dQ/dV * V
        setGV (i1, _ghs[i1]); // jacobian dI/dV * V
        for (int i2 = 0; i2 < 6; i2++) {
            setQV (i1, i2, _jdyna[i1][i2]); // jacobian dQ/dV
        }
    }
}

#include "Solo.defs.h"

// loader constructor interface
extern "C" {
    class va_proxy { public:
        va_proxy(){
            // register the constructor and the definitions with the global factories
            // Note that ::create and ::definition are handled by expanding the CREATOR define
            constructor
            factorycreate["Solo"] = Solo::create;
            factorydef ["Solo"] = Solo::definition;
        }
    };
    // our one instance of the proxy
    va_proxy proxy_instance;
}

```