Digital Circuit Builder v0.9.7

Documentação de Software

Eduardo C. Matias

07/2019

Conveniência

X-Y => Série de caracteres (de X a Y) com tamanho e objetos com valores intuitivos;

(ANY: X, Y)^{N} => Uma *string* com n símbolos quaisquer das séries X e/ou Y.

Not, And, Or, Xor etc. => Tipos comuns de portas lógicas;

XY => Concatenação de *strings* X e Y.

Recursos

Tokens: *String* fixa indicada para representar portas – deve ser definida na descrição do circuito. Caso uma porta lógica seja definida, porém nunca chamada por outra porta lógica, será criado uma "porta saída" com referência padronizada que terá o valor calculado sendo o da porta "solitária".

Tamanho de *Token*: A *string* de referência a determinada porta deverá possuir um tamanho entre os limites máximo e mínimo definidos no header specs.h.

Comentários: Linhas podem incluir, ou serem constituídas de, comentários cujo tamanho e informação não serão levados em consideração pelo construtor do circuito. Comentários são precedidos de um símbolo pré-definido pelo desenvolvedor.

Árvore de Circuito Abstrato (ACT): Árvore Binária de análise personalizada para organização de portas declaradas e suas entradas, assim como seus tipos (implementação baseada nas AVLs de compiladores).

Dependências usuais: stdio.h, string.h, stdlib.h, iso646.h, stdbool.h (padrões de +C90).

Scanner

Descrição: Obtém listas de *tokens*, separados por caracteres pré-determinados, verificados por cada linha lida do arquivo. Ignora comentários e retorna um erro caso uma linha do arquivo apresente um formato inadequado (E.g. contém apenas um *token* cujo tamanho é inferior/superior ao limite mínimo/máximo estabelecido).

Dependências: Além das usuais temos **linked_l.h**.

Parser

Descrição: Gera uma lista de ACTs (***A**bstract **C**ircuit **T**rees*) para posterior verificação da declaração explícita ou implícita de cada porta. Cada ACT possui como nó raiz o nome referência da porta. Caso seja uma declaração de *input*, ambos os nós da esquerda e direita terão endereços nulos, caso seja uma declaração de *output*, apenas o nó da esquerda terá referência para um circuito lógico ou entrada (caso a *input* nunca seja chamada, então ela é tanto entrada quanto saída).

Dependências: Além das usuais temos **parser.h** e **ACT.h**.

Bibliotecas:

A versão atual deste programa contém cabeçalhos (*headers*) com funções não-otimizadas, implementadas para abstração da informação em estruturas de dados comuns (E.g. lista encadeada simples, *hashes*, árvore binária etc) para análise e geração de resultados.

Descrição do circuito

É sugerido que o arquivo de formato .txt contenha a referência de portas de entrada, saída e lógicas no seguinte formato:

Formato *Token* (Recomendado):
(ANY: A-Z)(ANY: A-Z, a-z, 0-9)^{N} ‘,’
N é a quantidade de caracteres usadas para numerar a referência, onde:
TAMANHO_TOKEN_MÍNIMO ≤ (N+1) ≤ TAMANHO_TOKEN_MÁXIMO;

A declaração da entrada deve conter um único *token*, de preferência no formato sugerido acima. Sugere-se o uso do caractere ‘,’ no fim da linha serve para que o scanner separe-a em *tokens* de acordo com o algoritmo escolhido. **Na versão atual, ainda não é possível declarar entradas com valores constantes. A declaração de entradas/saídas multicíclicas devem ser feitas manualmente[1].**

E.g. A0000, # Declara uma entrada.

Obs.: *(ANY: A-Z, a-z)(ANY: A-Z, 0-9)^{N} e (ANY: A-Z, a-z)(ANY: A-Z, 0-9)$^{N-1}$* não se referirão à mesma porta lógica.

A declaração explícita da saída é similar à da entrada. Dois *tokens*, um para referência a própria porta e outro à entrada, são declarados. Não é necessário que a entrada da porta seja acompanhada de uma vírgula no final (assim como na declaração de portas lógicas).

Por fim, a declaração de portas lógicas é feito com três ou mais *tokens*. O primeiro, novamente, se refere a própria porta. O segundo é referente ao tipo de entrada (tipos são pré-definidos pelo desenvolvedor). O n-ésimo *token* fará referência às entradas da porta.

Obs.: Note que, para a versão atual, uma porta tipo *NOT* sempre possuirá uma única entrada.

Nenhuma das portas declaradas deve conter, diretamente, ela mesma como entrada.

É preferível que a declaração seja feita na seguinte ordem:

E.g. descrição.txt contém:

ENTRADAS (Optativo)

PORTAS LÓGICAS

SAÍDAS (Optativo)

Desta maneira, a quantidade de ciclos utilizados para verificações de colisão será reduzida.

[1]E.g. **A0,not,Cin**
Cout,A0.
(Cin e Cout representam a mesma “porta entrada/saída” em dois ciclos diferentes)

Detalhes Técnicos

O projeto foi testado em ambiente Microsft Windows 10, usando o compilador Cygwin gcc versão 7.4.0. O mesmo se encontra em um estado não-otimizado para testes iniciais. A complexidade temporal e espacial da maioria dos algoritmos utilizados é NP porém é possível fazer *trade-offs* de memória e legibilidade para diminuir o tempo de análise e resolução do circuito.

Descrição do Projeto

O construtor de circuitos é constituído de sete cabeçalhos (scanner.h, parser.h, circuit.h etc) e um programa principal em C.

A etapa inicial do construtor é analisar o arquivo texto para organizar as linhas e filtrar caracteres indesejados. A segunda etapa é utilizar o *scanner* analisar cada linha para encontrar *tokens* que façam referência a portas lógicas e seus tipos. Em seguida um *parser* estabelece a relação dos *tokens* como determinado anteriormente e demonstram-na com uma árvore binária de análise com a raíz contendo o *token* de referência à própria porta lógica.

Por último, o programa principal inicializa todas as portas lógicas fazendo uma análise semântica completa da *ACT* produzida a partir do *parser*. Um *loop* aciona a função de cálculo de portas contido no cabeçalho circuit.h, para, enfim, escrever os resultados obtidos daquele circuito em um arquivo cujo nome é semelhante ao de entrada.

O número de combinações de entradas (n) será sempre igual a dois à n-ésima potência, portanto é recomendável utilizar a versão atual do construtor apenas para calcular saídas de circuitos com uma quantidade não-elevada de entradas.