UNIVERSIDADE FEDERAL DO AMAZONAS

PROREITORIA DE PESQUISA E PÓS-GRADUAÇÃO

DEPARTAMENTO DE APOIO A PESQUISA

PROGRAMA INSTITUCIONAL DE INICIAÇÃO CIENTÍFICA

Verificação de Modelos de Hardware que usam a Biblioteca SystemC

Bolsista: Luciano Nunes Sobral, FAPEAM

Manaus

2013

UNIVERSIDADE FEDERAL DO AMAZONAS

PROREITORIA DE PESQUISA E PÓS-GRADUAÇÃO

DEPARTAMENTO DE APOIO A PESQUISA

PROGRAMA INSTITUCIONAL DE INICIAÇÃO CIENTÍFICA

RELATÓRIO FINAL

PIB-E-0214/2013

Verificação de Modelos de Hardware que usam a Biblioteca SystemC

Bolsista: Luciano Nunes Sobral, FAPEAM

Orientador: Lucas Carvalho Cordeiro

Manaus

2013

RESUMO

Sistemas digitais são largamente utilizados no mundo contemporâneo e seu crescimento tecnológico nas últimas quatro décadas aparentam ser gigantescas, basta olhar para a capacidade de processamento que os hardwares atuais possuem em relação à década passada. Os avanços industriais e científicos se beneficiam desses sistemas, tornando o trabalho mais produtivo e lucrativo, e isso mostra a dependência que a sociedade tem da tecnologia digital. Por isso, para alcançar melhores resultados, são criados modelos computacionais para representar esses sistemas a fim de reduzir gastos, encontrando falhas antes que o projeto entre em fase de implementação física. No entanto, esses modelos são projetados por humanos e estão sujeitos à falhas e erros funcionais, o que leva a uma falsa segurança de que o modelo computacional esteja funcionando. Afim de lidar com esse fator humano, foram criadas ferramentas que verificam o código fonte que implementam esses modelos, entre elas está o ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*). Este trabalho apresenta uma tentativa de estender esta ferramenta para que verifique modelos computacionais de hardware que foram implementados usando a biblioteca SystemC. Após a implementação foram verificados portas lógicas digitais básicas como AND, NOT, OR e suas combinações NAND, NOR além de outros exemplos, mas nos limitamos a apresentar apenas um exemplo que abrange todas as extensões feitas.

Palavas-chaves: ESBMC, Model Checking, C++, Sistemas Digitais, Modelos Computacionais

SUMÁRIO

1 INTRODUÇÃO

Atualmente existe uma alta dependência de sistemas digitais no cotidiano do homem contemporâneo, bastar olhar em volta para ver que é quase impossível andar sem passar por pelo menos um desses sistemas, ainda que imperceptíveis. Exemplos desses sistemas podem ser encontrados em elevadores, câmeras de segurança, portas automáticas, aparelhos de comunicação, móveis, televisores, computadores, condicionadores de ar, aparelhos hospitalares e assim por diante.

Os projetistas desses sistemas usam modelos computacionais para representar o hardware, a fim de reduzir o custo de materiais (uma vez que não há perdas físicas caso o modelo esteja incorreto), melhorar o processo de desenvolvimento (módulos de software e hardware podem ser desenvolvidos em paralelo) e realizar testes exaustivos antes da etapa de manufatura. Esses modelos são criados usando linguagens de programação específicas conhecidas como HDL (*Hardware Description Language*) entre elas estão o VHDL e Verilog. Nesse projeto de pesquisa, será usada uma expansão da linguagem C++ chamada de SystemC usada para criar modelos de hardware num nível mais alto de abstração que as HDL's.

Infelizmente, os testes mostram apenas a presença do erro, mas não garante sua ausência. Dessa forma, os desenvolvedores fazem a verificação usando métodos formais para que as especificações desses modelos sejam satisfeitas, entre esses métodos estão o SMT *(Satisfiability Modulo Theories*) e o BMC (*Bounded Model Checking*), ambos utilizados como base para o desenvolvimento da ferramenta ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*) usada nesse projeto de pesquisa. O ESBMC é um verificador de modelos de contexto limitado que permite a verificação de processos simples e paralelos criados em C ou C++ que compartilham uma mesma região de memória. A ferramenta suporta todo o padrão ANSI-C e C++ e consegue verificar programas que fazem uso de vetores, ponteiros, estruturas, alocação de memória e aritmética de ponto fixo. O ESBMC consegue verificar também *underflow* e *overflow* aritméticos, segurança de ponteiro, *vazamento de memória*, violações de atomicidade, bloqueio fatal local e global, corrida de dados, e assertivas definidas pelo usuário. No entanto, o ESBMC ainda não é capaz de verificar sistemas desenvolvidos usando a biblioteca SystemC.

Sendo assim, este trabalho tem como objetivo expandir a ferramenta ESBMC para que faça verificações em modelos de hardware que usam a biblioteca SystemC como ferramenta de desenvolvimento. Desta forma, este trabalho está dividido nos seguintes tópicos:

Revisão Bibliográfica: onde se descreve os trabalhos relacionados usados como base para a criação e execução desse projeto de pesquisa.

Métodos Utilizados: onde se mostra a metodologia utilizada para desenvolver e analisar os resultados obtidos.

Resultados e discussões: onde são apresentados os resultados obtidos até o momento e algumas discussões sobre os mesmos.

2 REVISÃO BIBLIOGRÁFICA

Diversos trabalhos na literatura tratam do uso de métodos formais para verificar sistemas. Até o presente momento, os trabalhos usados como referência são os que seguem:

Bounded Model Checking Using Satisfiability Solving (CLARKE et al., 2001)

O artigo trata do *model checking* usando satisfação Booleana (SAT), explica alguns tipos existentes (TLMC [temporal logic model checking] e BMC [Bounded Model Checking])

Satisfiability Modulo Theories: An Appetizer (MOURA; BJØRNER, 2009)

Explica sobre o problema da satisfabilidade booleana (SAT) e a teoria dos módulos da satisfação (SMT)

SMT-Based Bounded Model Checking for Embedded ANSI-C Software (CORDEIRO; FISCHER; MARQUES-SILVA, 2009)

Onde apresenta a abordagem de um verificador formal (ESBMC) que usa verificação de modelos limitado (BMC) baseado nas teorias do módulo da satisfação (SMT) para checar códigos embarcados em ANSI-C

Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking (CORDEIRO; FISCHER, 2011)

Onde estende a ferramenta ESBMC para verificar códigos que possuem concorrência, apresenta três abordagens de verificação de softwares *multi-threads* usando *Pthread*.

Verifying SystemC: A Software Model Checking Approach (CIMATTI; MICHELI, 2010)

Onde aborda dois métodos de verificação, um transformando códigos em SystemC para C (incluindo o paralelismo de processos) e outro verificando de forma separada os códigos sequenciais dos códigos concorrentes.

SystemC from the Ground Up, 2 ed. David C. Black

Onde mostra as estruturas básicas da biblioteca SystemC, tipos e explica de forma simplificada o funcionamento do núcleo de simulação usado nas concorrências dos processos entre os módulos projetados em SystemC.

3 MÉTODOS UTILIZADOS

Este trabalho de pesquisa tem como objetivo implementar uma estrutura simplificada com a finalidade de verificar a utilização de cada método da biblioteca SystemC e integrá-la no verificador ESBMC. Nesta pesquisa, pretende-se verificar de forma automática aplicações reais desenvolvidas na linguagem C++ que usam a biblioteca do SystemC.

A metodologia empregada nesta pesquisa pode ser dividia em três etapas principais. Primeiramente, a revisão da literatura a respeito da teoria de Verificação de Modelos. Nesta etapa, a maioria dos conceitos importantes acerca de Lógica Proposicional, Lógica Temporal Linear (LTL) e Lógica de Árvore de Computação (CTL) serão estudados com o intuito de entender o funcionamento do verificador ESBMC. Esta parte contemplará também um estudo aprofundado sobre a biblioteca SystemC e sua utilização em sistemas digitais, onde serão realizadas análises sistemáticas de códigos que utilizam esta biblioteca.

A partir do conteúdo estudado, nesta etapa será construída uma estrutura simplificada das bibliotecas com os métodos mais utilizados. Essa estrutura será montada a partir da implementação das bibliotecas definidas com as assinaturas dos seus métodos e definições de tipos.

Finalmente, será estudado os métodos definidos na estrutura simplificada, de forma que possa ser checada todas as propriedades do mesmo, e assim validar o seu uso. Vários testes serão realizados nessa etapa para validar o fato de que as implementações desenvolvidas possam ser aplicadas em sistemas reais. Após finalizada a modelagem, todas as implementações serão integradas no verificador ESBMC.

4 RESULTADOS E DISCUSSÕES

O ponto de partida desse trabalho na verificação de programas SystemC foram as portas lógicas AND, NOT, OR, que são os módulos mais básicos que podemos encontrar num circuito digital, todo sistema digital possui ao menos um desses módulos em seu projeto.

4.1 PORTA LÓGICA AND

Uma operação AND deve possuir duas ou mais entradas, e retorna um sinal lógico verdadeiro caso todas as entradas tenham um sinal verdadeiro ou falso caso ao menos uma das entradas possua um sinal falso. A figura abaixo mostra a representação gráfica de uma porta AND com duas entradas.

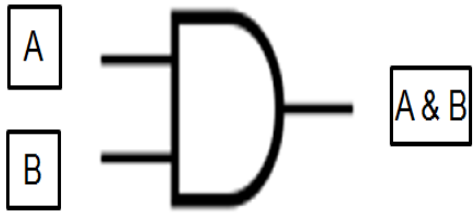

Figura 1- Representação gráfica de uma porta lógica AND

Fisicamente podemos representar uma porta AND na forma de um circuito elétrico simples onde várias chaves estão ligadas em série a uma lâmpada, cada chave representa uma entrada e caso esteja aberta, seu valor lógico seria falso, caso contrário verdadeiro, a lâmpada só será acessa caso todas as chaves estejam fechadas.

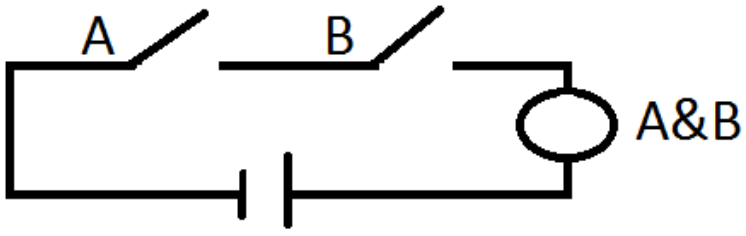

Figura 2 - Porta AND vista como um circuito elétrico simples

Entrada		Saída
A	B	A & B
0	0	0
0	I	0
I	0	0
I	I	I

Figura 3 - Tabela verdade de uma operação AND

A figura acima mostra a tabela verdade resultante de uma porta lógica AND, o valor 0 representa um sinal falso, enquanto o valor 1 representa um verdadeiro. O código abaixo, que implementa uma porta AND usando a biblioteca SystemC, foi o ponto de partida para a modelagem da biblioteca simplificada e será explicado nas seções seguintes.

```
1.  #include <systemc.h>
2.
3.  SC_MODULE( AND )
4.  {
5.    sc_in<bool> entrada_A;
6.    sc_in<bool> entrada_B;
7.    sc_out<bool> saida_AB;
8.    sc_clock relogio;
9.
10.   SC_CTOR( AND )
11.   {
12.      SC_METHOD( processo_and );
13.      sensitive << entrada_A;
14.      sensitive << entrada_B;
15.      SC_THREAD( processo_teste, relogio );
16.   }
17.
18.   void processo_and( void )
19.   {
20.      saida_AB.write( entrada_A.read() & entrada_B.read() );
21.   }
22.
23.   void processo_teste( void )
24.   {
25.      assert( ( entrada_A.read() & entrada_B.read() ) == saida_AB.read() );
26.   }
27. };
28.
29. int sc_main( int argc, char * argv[] )
30. {
31.    sc_signal<bool> sinal_1;
32.    sc_signal<bool> sinal_2;
33.    sc_signal<bool> sinal_3;
34.
35.    sinal_1.write(true);
36.    sinal_2.write(true);
```

```
37.    sinal_3.write(true);
38.
39.    AND porta_and(“Porta AND”);
40.
41.    porta_and.entrada_A(sinal_1);
42.    porta_and.entrada_B(sinal_2);
43.    porta_and.entrada_C(sinal_3);
44.
45.    sc_start();
46.
47.    return 0;}
```

4.2 SC_MAIN

Apesar da biblioteca SystemC ser uma extensão da linguagem C++, a sua estrutura básica apresenta algumas diferenças no modo de programar. A primeira delas pode ser percebida na função principal.

Todo programa precisa iniciar de algum ponto, o ponto inicial de execução nos softwares desenvolvidos em C e C++ é a função main(), a ausência dessa função no códigos fontes causa erro na compilação. Quando a biblioteca systemc.h (linha 1 do código fonte na seção 4.1.) é inclusa num código C++, a função main deixa de ser exigida como principal, passando o ponto inicial para a função sc_main como pode ser visto na linha 30.

O parser da ferramenta ESBMC não entende que sc_main passa a ser a função principal quando system.h é incluso. A solução encontrada foi definir sc_main como main

```
1.  #define sc_main main
```

A partir desse ponto todo código que encontrar um sc_main será convertido para main e o ESBMC entenderá como um código fonte em C++.

4.3 SC_MODULE

A linha 3 do código mostra a definição de um módulo, SC_MODULE pode ser visto como um contêiner de processos, portas, sinais e outros componentes menores, é nele que é feita a comunicação entre processos, seu uso tem a seguinte sintaxe.

```
1.  SC_MODULE( nome_modulo ) {  };
```

A modelagem dessa estrutura foi feita da seguinte forma, primeiro foi criada uma classe chamada sc_module e depois usado um macro para criar uma segunda classe, com o nome definido em SC_MODULE, que herda a primeira.

```
1.  struct sc_module
2.  {
```

```
3.    sc_module() {}
4.    ~sc_module() {}
5.  };
6.
7.  #define SC_MODULE(nome_modulo) struct nome_modulo : public sc_module
```

A vantagem desta abordagem está no uso das características da orientação a objeto, dessa forma qualquer modificação feita na classe sc_module afetará qualquer outro modulo definido com SC_MODULE.

4.4 PROCESSOS

Um processo é um método definido dentro de um módulo, não possui retorno e nem argumentos. São eles que dão funções aos módulos definidos pelo usuário.

```
1.  void processo ( void );
```

Como os processos são reconhecidos como métodos de classe pelo ESBMC, e os módulos foram definidos como classes na modelagem simplificada, não houve necessidade de modelar um processo. As linhas 19 e 24 mostram a definição de dois processos.

4.5 SC_CTOR

SC_CTOR é o construtor do módulo definido em SC_MODULE, é nele que é feito o registro dos processos usando procedimentos próprios para isso. Sua declaração é feita da seguinte forma:

```
1.  SC_CTOR( nome_modulo ){ }
```

O parâmetro nome_modulo deve ser o mesmo do definido em SC_MODULE. A modelagem feita desse construtor foi feita da seguinte forma:

```
1.  #define SC_CTOR(nome_modulo) nome_modulo()
```

Simulando a declaração de um construtor padrão de uma classe em C++.

4.6 SC_METHOD E SC_THREAD

Para que um processo seja executado é necessário que ele seja registrado no núcleo de simulação do SystemC para que haja controle de concorrência. SC_METHOD E SC_CTHREAD são chamados somente dentro do construtor, os processos registrados por esses dois métodos serão executados após a chamada da função sc_start(). A definição de SC_METHOD E SC_CTHREAD é a seguinte.

```
1.  SC_METHOD( processo );
2.  SC_THREAD( processo );
```

Sistemas digitais usam de concorrência para obter uma maior eficiência ao realizar uma tarefa, SystemC usa o macro SC_THREAD para simular essa concorrência, neste trabalho adotamos a biblioteca pthread para implementar essa macro, a criação de uma classe especifica para tratar desse caso foi necessária. Dentro de cada módulo definido pelo usuário criamos automaticamente classes que herdam o comportamento concorrente de threads, foi criado também uma fila de processos, e a cada definição dessas classes é instanciado uma fila com métodos de classe locais, os elementos são inseridos nessa fila com os macros SC_METHOD ou SC_THREAD, a única diferença entre essas duas macros é que os processos registrados pela segunda serão executados em paralelo enquanto que os registrados na primeira, não.

```
1.  #define SC_METHOD( processo ) scmethod(this, &MODULE::processo)
2.  #define SC_THREAD( processo ) scthread(this, &MODULE::processo)
```

4.7 sc_start()

Uma vez que concluímos a fase de registro de processos partimos para a execução dos mesmos, uma classe foi criada para gerenciar os módulos declarados pelo usuário e uma instancia chamada kernel cuida da gerência dos mesmos, o trabalho da função sc_start() é chamar os métodos do objeto kernel, primeiro os processos registrados pela macro SC_METHOD depois os registrados pela macro SC_THREAD

```
1.  void sc_start(void)
2.  {
3.     kernel.run_method();
4.     kernel.run_thread();
5.     Kernel.join_thread();
6.  }
```

A linha 5 chama o método para executar o join de cada thread inicializada na fila dos processos registrados com SC_THREAD, a medida é necessário porque uma vez que uma thread recebe um join nenhuma outra thread pode ser criada.

5 CONCLUSÃO

Pode-se concluir que a verificação de modelos representa uma técnica eficaz para os desenvolvedores de sistemas digitais com o intuito de garantir a qualidade de seus projetos de hardware. Neste relatório, foi abordado a implementação de um modelo simplificado da biblioteca SystemC. No entanto, no momento ainda não há uma modelagem estável da noção de tempo na biblioteca simplificada,pois os processos são chamados linearmente. Ainda assim foi possível verificar mais de dez casos de testes das portas lógicas básicas AND, OR, NOT, além das portas NOR e NAND com threads sendo executadas com precisão e reconhecendo erros de assertivas dentro delas.

6 BIBLIOGRAFIA

CIMATTI, A.; MICHELI, A. Verifying SystemC: a software model checking approach. **Formal Methods in Computer-Aided Design (FMCAD), 2010**, p. 51-59, 2010.

CLARKE, E. et al. Bounded model checking using satisfiability solving. **Formal Methods in System Design**, n. 97, p. 1-20, 2001.

CORDEIRO, L.; FISCHER, B. Verifying multi-threaded software using smt-based context-bounded model checking. **Software Engineering (ICSE), 2011 33rd …**, p. 331, 2011.

CORDEIRO, L.; FISCHER, B.; MARQUES-SILVA, J. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. **2009 IEEE/ACM International Conference on Automated Software Engineering**, p. 137-148, nov. 2009.

MOURA, L. DE; BJØRNER, N. Satisfiability modulo theories: An appetizer. **Formal Methods: Foundations and Applications**, p. 1-16, 2009.