UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE APOIO Á PESQUISA
PROGRAMA INSTITUCIONAL DE INICIAÇÃO CIENTÍFICA

RELATÓRIO FINAL PIBIC
PIB-E/0003/2015
AVALIAÇÃO DE PROJETOS DE FILTROS DIGITAIS DE PONTO-FIXO USANDO VERIFICAÇÃO DE MODELOS

Valdeson Dantas de Souza

Manaus-AM
2016

Aluno: Valdeson Dantas de Souza

Orientador: Prof. Dr. Lucas Carvalho Cordeiro

RELATÓRIO FINAL PIBIC

PIB-E/0003/2015

AVALIAÇÃO DE PROJETOS DE FILTROS DIGITAIS DE PONTO-FIXO USANDO VERIFICAÇÃO DE MODELOS

Manaus-AM

2016

AGRADECIMENTOS

Em vista desse desenvolvimento de projeto, tive a oportunidade de conhecer muitas pessoas que maximizaram além do meu conhecimento científico a amizade e o comprometimento na jornada para almejar ao projeto proposto. A essas pessoas ofereço minha gratidão, porque através das nossas discussões teóricas e científicas, resultou na obra desenvolvida.

Muito obrigado Mônica Marcelly B. da Rocha, que me ajudou com as discussões bastante produtivas e que me auxiliou no desenvolvimento dos algoritmos projetados e sendo crítica nos mesmos e fazendo com toda a dedicação que só ela tem.

Agradeço também aos pais que sempre me deram suporte e motivação tanto nas horas difíceis quanto alegres. E por fim, a Deus.

RESUMO

Em processamento digital de sinais é uma ciência que estuda regras que governam o comportamento de sinais discretos, bem como os dispositivos que os processam, surge então, uma alternativa ao processamento analógico de sinais, podendo alterar, corrigir ou até mesmo atualizar essas aplicações. Assim, para que o sinal seja processado digitalmente necessita ser, primeiramente, discretizado. E é por isso uma das principais tarefas na área de PDS é o projeto de filtros digitais, e atualmente tal procedimento é realizado com a ajuda de ferramentas computacionais e, consequentemente, surge a discussão de como os dados seriam representados em sistemas digitais, e como iriam ser realizadas as operações aritméticas nessa ferramenta. Portanto deve-se pensar em um sistema que represente e distinga o sinal do número, a parte inteira e a parte decimal. Em virtude disso, esse trabalho constitui-se então no desenvolvimento de uma biblioteca que tenha as funções básicas da aritmética, como adição, subtração, multiplicação e divisão para que sejam futuramente usadas para filtros digital e ferramentas afins. Nesse processo foi adotada a notação de ponto flutuante simples e dupla, que é definia na norma IEEE 754-1985(*INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS,1985*), com 32 e 64 bits.

Entretanto, durante a fase de implementação, a representação dos coeficientes obtidos , podem muitas vez resultar em comportamentos inesperados ou instáveis para filtros. O presente trabalho aborta esse problema e propõe uma verificação de modelos baseada em verificadores do tipo eficiente *SMT-based context-bounded model Checker* [1],[2], com o objetivo de analisar se a quantidade de bits utilizada, na representação dos coeficientes, resultará na correta aritmética escolhida.

Palavras – Chave: ESBMC, Verificação de Modelos, Filtros digitais, Ponto flutuante, PDS.

ABSTRACT

In digital signal processing is a science that studies the rules that govern the behavior of discrete signals, and the devices that process then arises, an alternative to analog signal processing, may alter, amend or even upgrade these applications.Thus, for the signal is digitally processed needs first be discretized. And that's why one of the main tasks in the PDS area is the digital filter design, and currently this procedure is performed with the help of computational tools and, consequently, there is the discussion of how data would be represented in digital systems, and how arithmetic operations would be performed on this tool. Therefore one should think of a system that is and distinguish the signal number, the integer part and the decimal part. As a result, this work constitutes then the development of a library that has the basic functions of arithmetic, such as addition, subtraction, multiplication and division to be used for future digital filters and related tools. This process adopted the notation about single- and double- floating-point precisions, which are defined according to the IEEE 754-1985 (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, 1985), with 32 and 64 bits.

However, during the implementation phase, the representation of the obtained coefficients, can often turn the result into unexpected or erroneous behaviour for filters. This study deals with this problem and proposes a model checking procedure based on efficient satisfiability modulo theories solvers [1], [2], in order to examine whether the number of bits used to represent the filter coefficients will result in the chosen correct arithmetic representation.

Keywords: ESBMC, Model Checking, Digital Filters, Floating Point, PDS.

LISTA DE SIGLAS

BMC – *Bounded Model Checking*

CBMC – *Bounded Model Checker*

ESBMC – *Efficient SMT-Based Context-Bounded Model Checker*

FIR – *Finite Impulse Response*

FPGA – *Field Programmable Gate Array*

IA – Inteligência Artificial

IDE – *Integrated Development Environment*

IEEE – Instituto de Engenheiros Eletricistas e Eletrônicos

IIR – *Infinite Impulse Response*

PDS – Processamento Digital de Sinais

SMT – *Satisfiability Modulo Theories*

LISTA DE FIGURAS

Figura 1: Ilustração da operação de um filtro digital..14
Figura 2: Ilustração do filtro digital e PDS...14
Figura 3: Ilustração do PDS...14
Figura 4: Conversão de decimal para binário..16
Figura 5: Representação em ponto flutuante na precisão simples............................17
Figura 6: Representação em ponto flutuante na precisão dupla...............................17
Figura 7: Função *convert* com seu valor de entrada..19
Figura 8: Biblioteca *ponto_flutuante.h* com função Deslocamento..........................19
Figura 9: Arquitetura do ESBMC...20

SUMÁRIO

1. INTRODUÇÃO..9

2. REVISÃO BIBLIOGRÁFICA..11

3. METODOLOGIA..13

4. RESULTADOS E DISCUSSÕES...14

5. CONCLUSÃO...21

6. CRONOGRAMA...22

REFERÊNCIAS BIBLIOGRÁFICAS...23

ANEXOS...26

1. INTRODUÇÃO

Atualmente, depende-se cada vez mais dos sistemas computacionais para realizar nossas atividades, sejam as mais complexas ou mais corriqueiras, e nesse mundo globalizado em que se vive, existe a relação entre homem e máquina, que desde século XVIII, está crescendo em muito na forma de como interagimos com máquinas e/ou sistemas computacionais, e um dos principais motivos para essa maximização é a incorporação de sistemas eletrônicos numa grande variedade de produtos tais como *automóveis, eletrodomésticos, e equipamentos de comunicação*. A grande maioria dos microprocessadores e microcontroladores projetados que utilizam componentes digitais mais baratos, cerca de 90%, são usadas em dispositivos que usualmente não são chamados de computadores, dentre alguns dispositivos estão *telefones celulares, caixas automáticos, veículos, e em uma infinidade de espécies de equipamentos e dispositivos*. O que diferencia este conjunto de dispositivos de um computador convencional *PC – Desktop, Notebook*, conhecido por todos é o seu projeto baseado em um conjunto dedicado e especializado constituído por *Hardware, Software e Periféricos.*

Como já exposto, o século XVIII teve uma grande importância tanto para o avanço das tecnologias quanto após a revolução industrial, dentre uma série de mudanças que ocorreram, surgiu em 1985 o padrão IEEE 754 [3] que definia algumas regras de normalização a serem seguidas, tendo em vista que antes disso cada fabricante e computadores e outros dispositivos, possuía um formato de representação diferente. Sendo assim, para que estejam de acordo com as normas, devem-se obedecer sinal, expoente e mantissa do número real.

Nos últimos anos, planejamento automatizado [4] vem sendo cada vez mais requisitado em aplicações práticas em diversas áreas, pois o principal desafio dessa área é a inteligência Artificial (IA) que estuda este processo de deliberação por meio da computação [4], ou seja, desenvolver algoritmos de planejamento eficientes para obtenção de soluções confiáveis para problemas reais. Por outro lado, embora essas aplicações requeiram soluções com um alto nível de confiabilidade, o uso de métodos formais [5] para síntese e validação automática de planos tem recebido relativamente pouca atenção. De fato, na literatura da área de planejamento automatizado, quase não

há referências ao uso de métodos formais para síntese ou validação de planos. Além disso, as poucas referências encontradas na área estão quase sempre relacionadas à abordagem baseada em verificação de modelos [6], para problemas com metas de alcançabilidade simples, especificadas por meio de fórmulas da lógica temporal de tempo ramificado CTL [7].

Após a criação das bibliotecas com as respectivas aritméticas em ponto flutuante com precisão simples e dupla, serão avaliadas através do Efficient SMT-Based Context-Bounded Model Checker (ESBMC) [1], que é um verificador de modelos de contexto limitado para códigos embarcados ANSI-C/C++ e possui a capacidade de verificar estouro de limites de vetores, divisão por zero e assertivas definidas pelo usuário. E os resultados serão comparados com o MATLAB que é tanto um ambiente quanto uma linguagem de programação e um de seus aspectos mais poderosos é o fato de que a linguagem Matlab permite-lhe construir suas próprias ferramentas reutilizáveis.

Dessa forma, este projeto de pesquisa visa criar funções específicas com as propriedades básicas: adição, subtração, multiplicação e divisão em ponto flutuante com os dois tipos de precisões na linguagem C, que converte em uma linguagem de máquina — instruções para o computador por meio de várias sequências de bits e com as ferramentas de verificação, determinar veracidade de cada função desenvolvida e modificada.

2. REVISÃO BIBLIOGRÁFICA

Bilhões de sistemas são produzidos anualmente para mais diferentes ferramentas; esses sistemas estão embutidos em equipamento eletrônicos maiores e executam repetidamente uma função específica de forma transparente para o usuário do equipamento. O avanço das tecnologias, principalmente após a revolução industrial do século XVIII, proporcionaram uma série mudanças, dentre elas surgiu em 1985 o padrão IEEE 754 [3] que definia algumas regras de normalização a serem seguidas, haja vista que antes disso cada fabricante de computadores e outros dispositivos, possuía um formato de representação diferente. Para que esteja de acordo com as normas, devem-se obedecer sinal, expoente e mantissa do número real.

Filtros digitais têm sido amplamente utilizados em uma grande variedade de aplicações, devido principalmente à sua flexibilidade, aliada a uma baixa complexidade computacional, o que é reforçado pela disponibilidade de processadores digitais de sinais (*digital signal processor* - DSP) e arranjos de portas programáveis em campo (*field programmable gate arrays* - FPGAs). Esses dispositivos podem ser classificados em duas categorias: de ponto fixo ou flutuante, que se referem ao formato usado para armazenar e processar as representações de dados numéricos. Na aritmética de ponto fixo, os intervalos entre os números adjacentes representados são normalmente iguais; a aritmética de ponto flutuante, por sua vez, resulta em intervalos não uniformes, que podem ser até dez milhões de vezes menores que a magnitude do número, para um mesmo comprimento de palavra [8].

Recentemente, a disponibilidade de processadores de ponto flutuante aumentou substancialmente. Todavia, a alta velocidade dos processadores de ponto fixo, em combinação com o seu custo reduzido, ainda os tornam a escolha preferencial para projetos os filtros digitais embarcados. No entanto, efeitos de quantização não-lineares, erros de arredondamentos e estouros aritméticos (*overflow*), se manifestam mais gravemente em implementações de ponto fixo. Todos esses efeitos são causados por operações consecutivas de adição e multiplicação, utilizando palavras de comprimento finito (sequencias de bits de tamanho fixo), o que pode afetar o comportamento do filtro desejado. Por exemplo, em relação a estruturas na forma direta, uma pequena mudança nos coeficientes do filtro, devido à quantização, pode resultar numa grande alteração na localização dos pólos e zeros do sistema[9].

Além disso, diferentes tipos de filtro apresentam diferentes problemas. Por exemplo, os filtros IIR podem sofrer de graves oscilações na saída, mesmo para um sinal de entrada nulo, o que é um conhecido como ciclo limite [10]. Filtros FIR, por sua vez, não manifestam tais efeitos do ciclo limite, contudo também sofrem outros problemas causados por limitações da palavra de comprimento finito (por exemplo, modificação de resposta de frequência). Existem muitos estudos sobre os efeitos de quantização e ciclo limite em filtros digitais, juntamente com técnicas para reduzir os seus efeitos, como previamente relatado por Claasen [11]. No entanto, essas técnicas, como mudança de escala e saturação, normalmente resultam em algumas desvantagens, tais como o aumento da potência de ruído causado por erros de quantização e arredondamentos. Sendo assim, a magnitude do erro deve ser verificada, a fim de assegurar que está em um nível aceitável.

Normalmente, os projetistas de filtro empregam ferramentas avançadas para definir os parâmetros do sistema, de acordo com a operação desejada no domínio do tempo ou da frequência, e usam software de simulação para validar seu comportamento, juntamente com testes extensivos. No entanto, na maioria dos casos, a aritmética de números reais é considerada durante os cálculos, o que pode levar a suposições erradas sobre o desempenho do filtro implementado.

Em vista dos fatos relatados, recentemente,[12] propuseram um método para a verificação de implementações de filtros IIR em ponto fixo, baseado em verificação de modelos limitada (*Bounded Model Checking* - BMC) utilizando solucionadores de teoria do módulo da satisfatibilidade (*Satisfiability Modulo Theory* - SMT), com o objetivo de checar as condições de verificação. A principal ideia da verificação de modelos baseada em SMT é considerar contraexemplos de um tamanho específico k e gerar uma fórmula em lógica de primeira ordem, que pode ser satisfeita se e somente se esse contraexemplo existir [1].

3. METODOLOGIA

As pesquisas realizadas têm como objetivo desenvolver funções específicas para aritmética com precisão simples e dupla em ponto flutuante na forma IEEE754. De forma a alcançar tal propósito, foram adotadas teorias de filtros digitais, conversões de bases, linguagem c, o padrão IEE754, a aritmética computacional, e implementação de verificação de modelos.

Primeiramente, foi realizada uma revisão sobre filtros digitais e seus dois tipos: de resposta ao impulso infinita (*infinte impulse response* - IIR) ou de resposta ao impulso finita (*finite impulse response* - FIR) de para entender onde será aplicada as funções aritméticas projetas. Assim, entendendo o processamento o processamento de sinais que consiste na análise e/ou modificação de sinais utilizando teoria fundamental, aplicações e algoritmos, de forma a extrair informações dos mesmos e/ou torná-los mais apropriados para alguma aplicação específica.

Para desenvolver os algoritmos para as funções aritméticas, foram estudadas as teorias de conversões de bases, para trabalhar com os números reais na sua forma binária, e depois no ambiente desenvolvimento de programas com o uso de técnicas e lógicas de programação na linguagem c. Além disso, com o intuito de padronizar as notações de número em ponto flutuante, seguiu-se a representação no padrão IEEE754.

O sistema operacional Linux foi escolhido para desenvolver as funções e a bibliotecas em ponto flutuante e, junto com o compilador gcc para linguagem c (O gcc é o mais usado e tido como *padrão para o Linux*). Com as funções básicas criadas passou-se para a seguinte etapa de validação dos códigos com o uso da ferramenta MATLAB que dentre várias outras, é a mais útil para a área de processamento de controle, e outras aplicações.

Por fim, após a conclusão das etapas anteriores, foram feitas as verificações e validações através do Efficient SMT-Based Context-Bounded Model Checker (ESBMC) [1]**,** esse verificador possui a capacidade de verificar estouro de limites de vetores, divisão por zero e assertivas definidas pelo usuário. E os resultados serão comparados com o MATLAB e um de seus aspectos mais poderosos é o fato de que a linguagem Matlab permite-lhe construir suas próprias ferramentas reutilizáveis.

4. RESULTADOS E DISCUSSÕES

Durante o processo de pesquisas de filtro digitais, as buscas consistiram-se, basicamente, para entender melhor sobre os resultados esperados em um algoritmo matemático em *hardware* ou *software* que opera sobre um sinal *x[n]* aplicado em sua entrada gerando na saída uma versão filtrada *y[n]* de *x[n]*.

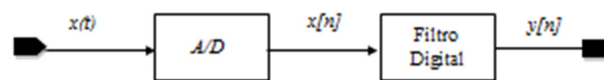


Figura 1: Ilustração da operação de um filtro digital.

Considerando que o filtro está implementado num Processador Digital de Sinais(PDS) e que o objetivo é processar um sinal analógico *x(t)*, temos:

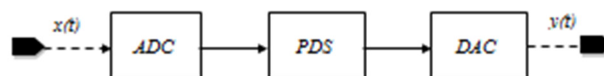


Figura 2: Ilustração do filtro digital e PDS.

Se os sinais a serem processados forem digitais o diagrama se resume a:

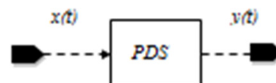


Figura 3: Ilustração do PDS.

O processamento de sinais é definido, identificando-se seu objeto, seus agentes e suas ações, bem como a sua arquitetura genéricas e, assim, essas características são apresentadas a seguir:

Objeto do processamento: sinal (definido como uma entidade que carrega informação).

Agente de processamento: sistema

- "Um sistema é um conjunto de elementos, que interagem entre si, com o objetivo de realizar uma determinada função".
- Arquitetura de um sistema: variáveis, elementos, topologia e função.

Domínio do processamento: domínio no qual a função do agente é definida.

- Tempo/espaço (forma) x frequência (composição espectral).

Ação do processamento: função exercida pelo agente sobre o objeto.

- Conformação (tempo/espaço) x Alteração espectral (frequência).

Arquitetura genérica do processamento:

- Sinal de entrada (ou estímulo ou excitação).
- Condições iniciais (valores de todas as variáveis interna do sistema).
- Sistema.
- Sinal de saída (ou resposta).

Nomenclatura usual: "Sinal"(sinal desejado) x "Ruído"(sinal indesejado).

Dessa forma, o processamento de sinais consiste na análise e/ou modificação de sinais utilizando teoria fundamental como explicado anteriormente, aplicações e algoritmos, de forma a extrair informações dos mesmos e/ou torná-los mais apropriados para alguma aplicação específica. O processamento de sinais pode ser feito de forma analógica ou digital. Este processo utiliza matemática, estatística, computação, heurística e representação, modelagem, análise, síntese, descoberta, recuperação, detecção, aquisição, extração, aprendizagem, segurança e forense[13]. Os objetos de interesse do processamento de sinais podem incluir sons, imagens, séries temporais, sinais de telecomunicações, como sinais de rádio e muitos outros.

Os filtros digitais são classificados quanto ao comprimento da sua sequência de resposta ao impulso como:

- Filtros de resposta ao impulso finita (FIR – Finite Impulse Response);
- Filtros de reposta ao impulso infinita (IIR – Infinite Impulse Response);

Hoje em dia existem diversos dispositivos que podem ser usados no processamento digital de sinais, como processadores digitais de sinais (*digital signal processor* -DPS) e arranjos de portas programável em campo (*field programmable gate arrays* - FPGAs). Esses dispositivos podem ser classificados em duas categorias: ponto fixo e flutuante, que se referem ao formato usado para armazenar e processar as representações de dados numéricos.

Mediante aos fatos decorridos, foram desenvolvidos programas na linguagem c, nos quais tinham como partes dos resultados as conversões do sistema binário-Decimal, que deve ser primeiramente numerada as posições de cada bit, para identificar os pesos.

Para começar numerar, inicia-se com o algoritmo menos significativo para o mais significativo, da direita para esquerda.

Assim, para converter $(1100)_2$ para Decimal, por exemplo , começa-se com a seguinte fórmula:

$N_{10} = 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0$

$= 8 + 4 + 0 + 0$

$= 12$

Para conversão do Decimal-binário, o jeito mais prático de realizar esse método será por divisões sucessivas com 2, e o resto com a divisão do número, irá ser o bit menos significativo, o quociente dessa divisão será dividido por 2 novamente; até que o quociente seja zero. Para exemplificar melhor essa teria vejamos o exemplo abaixo:

Converter $(12)_{10}$ para base binária

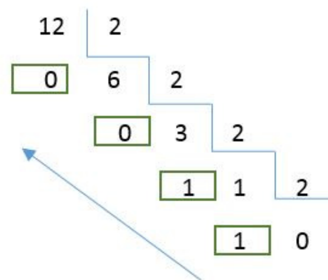


Figura 4: Conversão de decimal para binário.

E com após a conversão de base, houve a padronização no padrão IEEE 754, o qual fundamenta na notação de um número em ponto Flutuante, que inicialmente transforma um número na base binária e fixar num ponto fixo, que no caso o bit 1, por exemplo:

$(12)_{10} = (1100)_2$

A representação para notação científica será, descolocar a vírgula até o ponto fixo que no caso ficará $1{,}100x2^3$.

Em sistemas computacionais, os números em ponto flutuante será representação em três campos:

Sinal: Indica se o número será positivo ou negativo através de um bit.

Expoente: Indica o expoente do número em base binária através de um conjunto de bits.

Significando: Indica o conjunto de bits localizados após a vírgula, ou seja, a parte fracionária.

Obs: Despreza-se a parte inteira, pois está implícito que sempre será 1.

Para precisão Simples: 23 bits após a vírgula para o resultado final

Figura 5: Representação em ponto flutuante na precisão simples.

Para precisão Dupla : 52 bits após a vírgula para o resultado final

Por exemplo, a representação binária do número $-0{,}75_{10}$ na precisão dupla no padrão IEEE 754.

$$(-1)^S \ x \ (1 + \text{mantissa}) \ x \ 2^{(1022-1023)}$$

Mantissa: 0,100...00

Ou seja:

Figura 6: Representação em ponto flutuante na precisão dupla.

No algoritmo, inicialmente utilizou-se a IDE (CodeBlock) que ficará responsável pela compilação dos código das funções com as aritméticas básicas produzidas, e para trabalhar com as aritméticas optou-se em construir as seguintes etapas:

Adição no Padrão IEEE-754 Função que realiza a operação de soma em ponto flutuante de dois números reais. **Protótipo Final**

```
soma_binarioInteiro( numeroDireito1, numero2, j);
```

```
soma_fracionario(numeroEsquerdo1);
```

1. *numeroDireito1*: vetor que armazenará os bits antes da vírgula, depois de sua normalização dos seus expoentes dos números em ponto flutuante.
2. *numero:* vetor que armazenará os bits do segundo número real depois da vírgula
3. *O inteiro j:* a variável inteira, o mesmo será a soma da quantidade de casas percorridas pelo número com maior expoente, com a diferença entre os dois expoentes dos dois números trabalhados.
4. *numeroEsquerdo1:* vetor que armazenará um conjunto de bits depois da vírgula do número com maior expoente, ou seja, a parte fracionária da operação.

Subtração no Padrão IEEE-754 Função que realiza a operação de subtração em ponto flutuante de dois números reais.

Protótipo Final:

```
subtracao_binario( numeroDireito1, numero2, j );
```

```
subtracao_inteira(numeroEsquerdo1);
```

Essas funções poderão ser inseridas no lugar da função de soma relatada anteriormente.

Multiplicação no Padrão IEEE-754 Função que realiza a operação de multiplicação em ponto flutuante de dois números reais através da multiplicação bit a bit, e com as suas respectivas posições de vírgula; após a normalização dos seus expoentes.

Protótipo Final:

```
multiplicacao_binario( num1, num2 ,p,q);
```

Num1, Num2: vetor de bits de um número em ponto flutuante.
p, q: quantidade de casas após a normalização para que ocorra o processo de multiplicação.

Divisão no Padrão IEEE-754 Função que realiza a operação de divisão em ponto flutuante de dois números reais.

Protótipo Final:

```
divisao_binario( num1, num2, m, n);
```

Num1, Num2: vetor de bits de um número em ponto flutuante.
m, n: quantidade de casas após a normalização para que ocorra o processo de divisão.

No desenvolvimento das funções, teve-se que criar duas bibliotecas para finalizar as precisão simples e dupla chamadas *convert_simples,h*, ponto_flutuante_ simples.h, *convert_duplo,h*, e ponto_flutuante_duplo.h. As bibliotecas *converts* visão completar às funções básicas da aritmética proposta pelo projeto, pois as mesmas respectivamente transformam números decimais para a forma IEEE 754 no formato simples com 32 bits e dupla com 64 bits. Assim, esperava diminui os erros de arredondamento. Diminuiu, demasiadamente, após converter para a forma decimal.

```
int main(){
   int sinal1, sinal2,i,j;
   float number_1,number_2;

   printf("Informe o primeiro numero binario: \n");
   scanf("%f",&number_1);

   convert(number_1);
   for ( i = 0; i < 31; i++) printf(" %d",array_IEEE[i]);
    i = 0;
    for( j = 0; j <= 31; j++){
        numero1[i] = array_IEEE[j];
        printf(" %d",numero1[i]);
        i++;
    }
```

```
#include <stdio.h>


float number,frac_number,rest,result;
int int_number,signal,quoc,flag;
int array_binary[100],array_frac[32],i;
int size_array_frac, size_array_binary;
int exp_int, k,size_array_IEEE,j;
int array_IEEE[32];

void convert(float number){

    signal = 0;
    if(number < 0.0) signal = 1;
    if(number < 0.0) number *=(-1);

    int_number = number/1;
    frac_number = number - int_number;
    //printf("\n%d %f",int_number,frac_number);
    i = 0;

while (int_number >= 2) {


    rest = int_number %  2;
    quoc = int_number / 2;
    int_number = quoc;
    array_binary[i] = rest;
    i++;

    }
```

Figura 7: Função *convert* com seu valor de entrada

As outras duas bibliotecas contêm as funções aonde há as aritméticas com seus respectivos parâmetros para suas sequências lógicas, tanto para 32 quanto para 64 bits.

```
#include <math.h>
#include <stdio.h>
#include <string.h>

  int fp_convert_binary(int number[])
  {
    int size, resultIEEE, base, e1;
    resultIEEE = 0;
    size = 1;
    while(size <= 8){
      base = pow(2,8-size);
      resultIEEE += number[size] * base;
      size++;
    }
    e1 = resultIEEE - 127;
    return e1;
  }

  int fp_shift_left(int decimalExponent, int number[])
  {
    int i, j;
    int rightSize,leftnumber;
    int rightnumber1[32],leftnumber1[32];
    j = 0;
    for(i = decimalExponent + 9; i <= 31; i++){
      rightnumber1[j] = number[i];
      j++;
    }
```

```
//Deslocamento para Esquerda

    expoenteDecimal = e1 - e2;
    if(expoenteDecimal<0) expoenteDecimal*=-1;
    fp_shift_left(expoenteDecimal,numero1);

    //Deslocamento para Direita - Normalizacao

    tamDireito -= 1;

    for(i = tamDireito; i <= 31;i ++ ) numeroDireito1[i] = 0;

    //Soma da parte inteira em binario

    j = tamDireito + expoenteDecimal;
    soma_binarioInteiro(numeroDireito1, numero2,j);

    //Soma da parte fracionaria em binario

    numeroEsquerdo1[0] = 1;

    j = expoenteDecimal + 1;
```

Figura 8: Biblioteca *ponto_flutuante.h* com função Deslocamento

Os limites escolhidos para as duas precisões na representação em ponto flutuante, seguiram-se a ordem dos extremos do próprio *array,* no qual está armazenado os bits no formato IEEE da precisão simples e dupla. Dessa forma, quando os valores de dados de entrada estiverem com muitas casas decimais, possivelmente, existirão arredondamentos inesperados nas aritméticas, mas serão trabalhados futuramente.

A aplicação de ferramentas BMC baseadas em SMT está se tornando popular para a verificação de software, principalmente devido ao advento de solucionadores SMT sofisticados, que foram construídos sobre solucionadores de satisfatibilidade booleana eficientes [14], [15], [16]. Dessa forma, [1] apresentam a ferramenta ESBMC com uma abordagem eficiente para a verificação de programas embarcados ANSI-C utilizando BMC (*Bounded Model Checking*) e solucionadores SMT.A ferramenta ESBMC faz uso dos componentes do C Bounded Model Checker(CBMC) [17];

Em particular, o ESBMC é um verificador de modelos, baseado em SMT, para programas ANSI-C/C++.Com o uso do ESBMC, é possível realizar a validação de programas sequenciais ou multitarefas e também verificar bloqueio fatal, estouro aritmético, divisão por zero, limites de *array* e outros tipos de violações. Garantindo um resultado final correto e conciso para os programas desenvolvidos.

A ferramenta escolhida foi o ESBMC[18], que utiliza o solucionador SMT Z3[19] e o front-end do framework CProver, o qual é base dos verificadores CBMC [17] e SATABS[20].

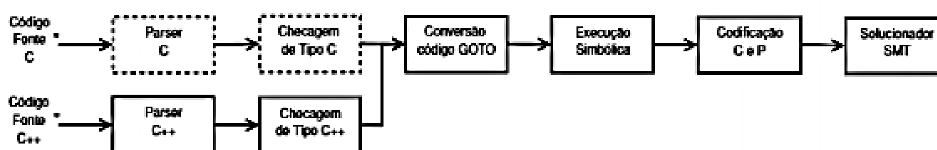


Figura 9: Arquitetura do ESBMC.

O esquema na Figura 9 mostra uma visão geral da arquitetura e do funcionamento da ferramenta ESBMC. Os dois primeiros blocos do processo de verificação, funcionam de forma diferente no caso da verificação de programas em C e C++ e que consistiram nos testes finais para cada aritmética desenvolvida para o projeto.

5. CONCLUSÃO

Neste trabalho foi abordado a verificação e desenvolvimento de uma biblioteca para implementações de filtros digitais que utilizam representação numérica em ponto flutuante com precisão simples e dupla. A metodologia proposta busca verificar a corretude das implementações através da verificação formal. Para isso é utilizado o ESBMC, um verificador limitado de modelos para software escritos em linguagem ANSI-C, baseado em teorias do módulo da satisfatibilidade e a ferramenta MATLAB para realizar as devidas precisões da saída da informação fornecida pelo programa proposto. Sendo assim, dependendo dos modelos de filtros faz-se o uso de uma biblioteca para aritmética em ponto flutuante com precisão simples e dupla, que permite avaliar estruturas com diferentes comprimentos da palavra de dados.

O método proposto permite que o projetista verifique formalmente uma determinada implementação em uma estrutura específica, pois a biblioteca proposta será acoplada a ferramenta de verificação chamada "DSVerifier" que possui verificadores de modelos inseridos a ela.

Adicionalmente, existe uma contribuição com o uso do SMT, o qual é utilizado para verificar as funções básicas desenvolvidas em linguagem C. Os resultados finais mostram a importância de saber estimar a exatidão do resultado de um cálculo computacional e como esses erros podem afetar em muito um procedimento do mesmo, por exemplo, erros nos dados de entrada, erros e arredondamento e erros de truncamento, dentre eles os erros de truncamento que são os mais comuns em algoritmos numéricos. Ocorrem quando, de alguma maneira, é necessário aproximar um procedimento formato por uma sequência infinita de passos através de um outro procedimento finito. Em virtude disso, qualquer que seja a natureza do erro, o método numérico utilizado nas funções aritméticas em ponto flutuante foram revisadas e comparadas com o Matlab, e não houve qualquer problema nos dados de saída e assim podendo evitar falhas futuras, já que a ordem de execução e a precisão propostas são essencialmente importantes a diversas áreas que utilizam esse tipo de procedimento. Portanto, o conjunto de ferramentas desenvolvido aqui pode ajudar projetistas de sistemas a definir a representação, afim de atender os requisitos funcionais e de desempenho.

6. CRONOGRAMA DE ATIVIDADES

Nº	Descrição	Ago 2015	Set	Out	Nov	Dez	Jan 2016	Fev	Mar	Abr	Mai	Jun	Jul
1	Estudar a técnica de Verificação Formal de Modelos.	X	X	X									
2	Estudar conceitos de filtros digitais e sistemas digitais.		X	X	X								
3	Estudar as estruturas de realização e a implementação em C.			X	X	X							
4	Análise de filtros digitais e construção de uma suíte de teste.				X	X	X						
5	Análise de *benchmarks* e definição das propriedades para verificar filtros digitais.					X	X	X					
6	Propor uma metodologia de projeto e verificação para filtros digitais.						X	X	X				
7	Aprimorar a implementação para reduzir o tempo de verificação das propriedades, tornando a aplicação do método mais escalonável.							X	X	X			
8	Aplicar a metodologia proposta à verificação de *benchmarks* e, através dos resultados obtidos, concluir sobre a sua eficácia.								X	X	X		
9	Integrar as implementações no ESBMC.									X	X	X	
	- - Elaboração do Resumo e Relatório Final (atividade obrigatória) - Preparação da Apresentação Final para o Congresso (atividade obrigatória)												X

REFERÊNCIAS BIBLIOGRÁFICAS

[1] L. Cordeiro, B. Fischer e J. M. Silva, “SMT-Based Bounded Model Checking for Embedded ANSI-C Software”, In IEEE Transactions on Software Engineering (TSE), v. 38, pp. 957–974, IEEE, 2012. 1999.

[2] L. Cordeiro e B. Fischer, “Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking”. In Intl. Conf. on Software Engineering (ICSE), pp. 331–340, IEEE/ACM, 2011.

[3] VAHID, Frank. Sistemas Digitais: projetos, otimização e HDLs. São Paulo: Artmed, 2008.

[4]Ghallab, M., Nau, D., and Traverso, P. (2004). Automated Planning: Theory and Practice. Morgan Kaufmann Publishers Inc., USA.

[5]Clarke, E. M. and Wing, J. (1996). Formal methods: state of the art and future directions. In ACM Computing Systems Surveys, volume 28.

[6]Cimatti, A., Giunchiglia, F., Giunchiglia, E., and Traverso, P. (1997). Planning via model checking: A decision procedure for AR. In ECP, pages 130–142.

[7]Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching-time temporal logic. In Logic of Programs, Workshop, pages 52–71, London, UK. Springer-Verlag.

[8] J. G. Proakis and D. G. Manolakis. Digital signal processing: Principles, algorithms and applications. Prentice Hall, 1996.

[9] 754-2008, I. Ieee standard for floating-point arithmetic. IEEE Computer Society, 2002.

[10] S. R. Parker and S. F. Hess. Limit-cycle oscillations in digital filters. IEEE Transactions on Circuit Theory, v. 18, n. 6, p. 687–697, 1971.

[11] T. A. C. M. Claasen, W. F. G. Mecklenbrauker and J. B. H. Peek. Effects of quantization and overflow in recursive digital filters. IEEE Transactions on Acoustics, Speech and Signal Processing, v. assf-24, n. 6, p. 517–529, 1976.

[12] A. Cox, S. Sankaranarayanan and Bor-Yuh E. Chang. A bit too precise? Bounded verification of quantized digital filters. TACAS, , n. 7214, p. 33–47, 2012.

[13]Moura, J.M.F. (2009). "What is signal processing?, President's Message". IEEE Signal Processing Magazine (26): 6.

[14] L. M. de Moura and N. Bjørner. Z3: An efficient smt solver. TACAS, v. LNCS 4963, p. 337–340, 2008.

[15] R. Brummayer and A. Biere. Boolector: An efficient smt solver for bit-vectors and arrays. TACAS, v. LNCS 5505, p. 174–177, 2009.

[16] C. Barrett and C. Tinelli. CVC3. Proceedings of the 19th International Conference on Computer Aided Verification, v. LNCS 4590, p. 298–302, 2007.

[17] E. Clarke, D. Kroening, e F. Lerda. A Tool for Checking ANSI-C Programs. In 10th International Conference, TACAS 2004. Springer, v. 2988, pp. 168–176, 2004.

[18] CORDEIRO, Lucas, FISCHER, Bernd, MARQUES-SILVA, João. SMT-based bounded model checking for embedded ANSI-C software.IEEE Transaction of Software Engineering, v. 38, n. 6, p. 50–55, 2002.

[19] MOURA Leonardo de, BJØRNER, Nikolaj. Z3: An efficient SMT solver. In: Lecture Notes in Computer Science. Budapeste, Hungria: Springer, 2008. v. 4963, p. 337–340.

[20] CLARKE, Edmund, KROENING, Daniel, SHARYGINA, Natasha, YORAV, Karen. Predicate Abstraction of ANSI-C Programs using SAT.In:
Formal Methods in System Design. Estados Unidos: Springer US, 2003. p. 105–127.

[21] LEVINE, John.flex & bison. Estados Unidos: O'Reilly, 2009. 292 p.

[22]Cimatti, A., Roveri, M., and Traverso, P. (1998). Strong planning in non-deterministic domains via model checking. In Artificial Intelligence Planning Systems, pages 36– 43.

[23]Daniele, M., Traverso, P., and Vardi, M. Y. (1999). Strong cyclic planning revisited. In ECP, pages 35–48.

[24]Git. Git- Distribued is the new centralized. Disponível em: <http://git-scm.com/>. Acesso em : 25 de dezembro de 2015.

ANEXOS

Estes são os resultados finais feitos no Sistema Operacional GNU/Linux usando o compilador gcc, que contêm simulações das funções aritméticas básicas trabalhadas durante o período de desenvolvimento do projeto.

- Para precisão simples-32bits:

1. Adição

```
Terminal
Informe o primeiro numero binario:
45
 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Informe o segundo numero binario:
12
 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
res_exp:2

carry: 1

Resultado da Adicao em Decimal: 57.00000000000000000000
1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

2. Subtração

```
Terminal
Informe o primeiro numero binario:
45
 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Informe o segundo numero binario:
12
 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
exp: 5 3

Resultado:

parte inteira: 1 0 0

parte fracionaria: 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Resultado da Subtracao em Decimal: 33.00000000000000000000
```

3. Multiplicação

```
Terminal
Informe o primeiro numero binario:
45
 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Informe o segundo numero binario:
12
 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
carry: 1

Resultado da Multiplicacao em binario: 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Resultado da Multiplicacao em Decimal: 540.000000
```

4. Divisão

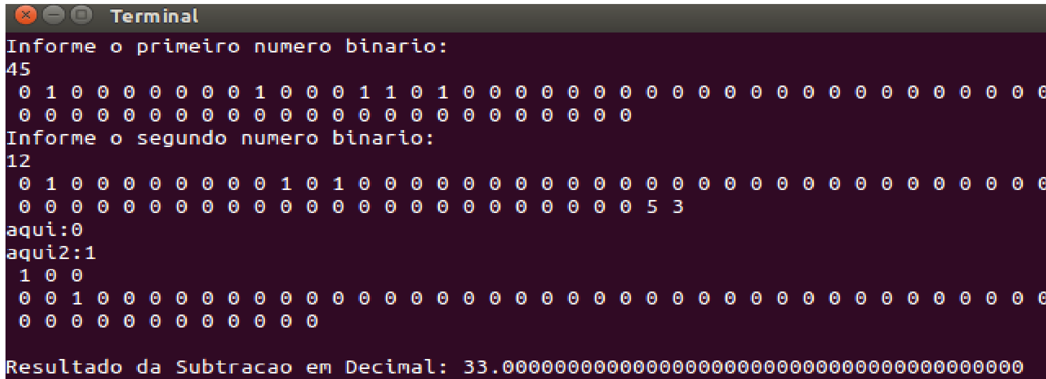
```
Terminal
Informe o primeiro numero binario:
45
 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Informe o segundo numero binario:
12
 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 exp: 5 3
Resultado da divisao: 3.7500
```

- Para precisão simples-64bits:

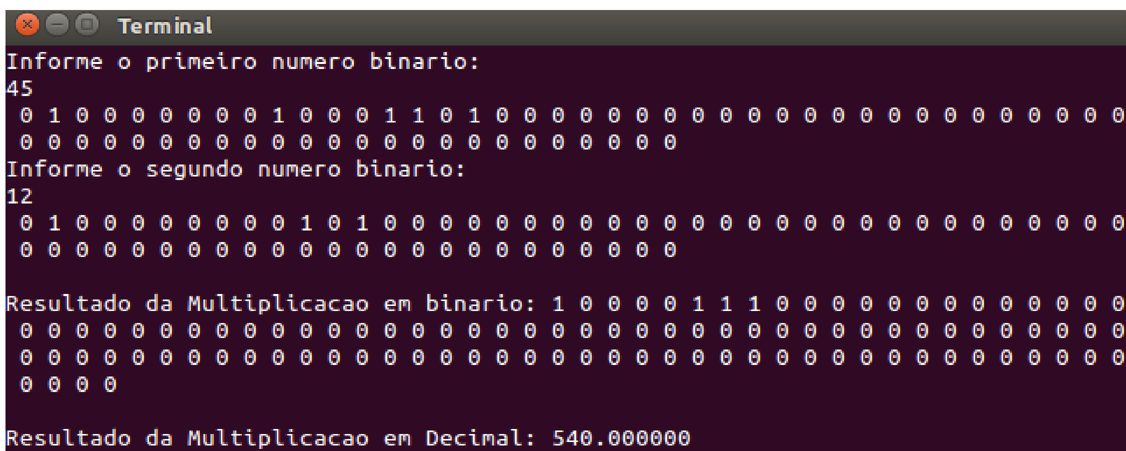
1. Adição

```
Terminal
Informe o primeiro numero binario:
45
 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Informe o segundo numero binario:
12
 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 exp: 5 3
Res_exp:2

Inteiro:


Resultado da Adicao em Decimal: 57.000000
1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

2. Subtração



3. Multiplicação



4. Divisão