https://eventos.utfpr.edu.br//sicite/sicite2018

Estudo da ferramenta Raspberry Pi para aplicações de filtragem digital de sinais de ultrassom

Study of the Raspberry Pi tool for digital ultrasonic signal filtering applications

Marcelo Kawasaki
mar_kawasaki@hotmail.com
Universidade Tecnológica Federal do Paraná, Curitiba, Paraná, Brasil

Amauri Amorin Assef
amauriassef@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Curitiba, Paraná, Brasil

RESUMO

As técnicas de ultrassomestão entre os maiores avanços tecnológicos na área da biomedicina. A grande utilização do diagnóstico médico por ultrassom tem se tornado referência em muitos exames clínicos, onde a filtragem digital é uma das etapas mais utilizadas no processamento digital de sinais. Dessa forma, este trabalho apresenta os estudos iniciais para a implementação de um filtro FIR passa-baixa embarcado em Raspberry Pi para sinais de ultrassom. Para validação do filtro com frequência passante e 1 MHz e frequência de parada igual a 8 MHz, foram utilizados dois sinais de teste. O primeiro sinal representa uma onda Chirp com frequência de 1MHz a 10 MHz e o segundo um sinal real de ultrassom, ambos amostrados com frequência de 40 MHz. Os resultados simulados no Matlab/Simulinke experimentais dos sinais filtrados no Raspberry Pi foram avaliados utilizando a função raiz quadrada do erro médio normalizado (NRMSE). Os resultados da função de custo foram aproximadamente 0,03% para o sinal Chirp e nulo para o sinal de ultrassom. Adicionalmente, os tempos de processamento dos dois sinais no Raspberry Pi também são apresentados.

PALAVRAS-CHAVE: Raspberry Pi. Ultrassom. Processamento de sinais. Filtro FIR Passa-baixa.

ABSTRACT

Ultrasound techniques are among the greatest technological advances in the field of biomedicine. The large use of ultrasound medical diagnosis has become a benchmark in many clinical exams, where digital filtering is one of the most commonly used steps in digital signal processing. Thus, this paper presents the initial studies for the implementation of a low-pass FIR filter embedded in Raspberry Pi for ultrasound signals. For filter validation with 1 MHz pass frequency and stop frequency equal to 8 MHz, two test signals were used. The first signal represents a Chirp wave with a frequency of 1MHz to 10 MHz and the second a real ultrasound signal, both sampled with a frequency of 40 MHz. The simulated results in the Matlab/Simulink and the experimental results of the filtered signals in Raspberry Pi were evaluated using the Normalized Root Mean Squared Error(NRMSE). The results of the cost function were approximately 0.03% for the Chirp signal and zero for the ultrasound signal. Additionally, the processing times of the two signals in Raspberry Pi are also shown.

KEYWORDS: Raspberry Pi. Ultrasound. Processing signal. Low Pass FIR Filter.

Recebido: 02 set. 2018.
Aprovado: 04 out. 2018.

INTRODUÇÃO

Em tempos de tecnologia é observado claramente a necessidade de ganhar tempo e aumentar ambas a qualidade e agilidade quanto à pesquisa cientifica, com o intuito de melhorar as informações obtidas. Como exemplo, pode-se citar a otimização no processamento nas diversas modalidades de ultrassom (US) existentes, contribuindo assim para o desenvolvimento de técnicas inovadoras de formação de imagem. Nesse sentido, o grupo de pesquisa do US da Universidade Tecnológica Federal do Paraná (UTFPR), vem desempenhando um importante papel para o desenvolvimento científico-tecnológico e inovação de equipamentos e materiais no Brasil. Uma das recentes contribuições do grupo foi o desenvolvimento da plataforma de pesquisa ULTRA-ORS (do inglês *Ultrasound Open Research System*) (ASSEF, 2013).

Assim sendo, este trabalho tem como objetivo implementar e estudar computacionalmente a aplicação de filtragem digital com o uso do sistema Raspberry Pi e compará-lo com os resultados obtidos no Matlab para aplicações futuras de processamento dos sinais para a reconstrução de imagens por US.

METODOLOGIA

Para o estudo da técnica de processamento de filtragem digital de sinais de US foi utilizado o *software*Matlab/Simulink e o equipamento Raspberry Pi B+/Raspibian, sendo o último programado em linguagem de programação Python.As avaliações das implementações foramrealizadas de forma a comparar os resultados obtidos na aplicação do filtro FIR e a robustez do equipamento utilizado, fornecido pelo Laboratório de Ultrassom (LUS) da UTFPR.

Para comparação e avaliação, foram utilizados dados brutos reais de US adquiridos pela plataforma de pesquisa ULTRA-ORS (ASSEF, 2013), com frequência de amostragem de 40 MHz e resolução de 12 *bits*. Na avaliação quantitativa do modelo proposto com relação aos implementados no Matlab foi empregada a função de custo da raiz quadrada do erro quadrático médio normalizado – NRMSE (*Normalized Root Mean Squared Error*), conforme a Eq. (1).

$$NRMSE = \sqrt{\left(\frac{\sum_i^N = 1\,((Mi - Si)^2)}{\sum_i^N = 1\,((Mi - \bar{M})^{2)}}\right)} * 100 \qquad (1)$$

Onde: *i* é o índice de amostra, *N* é o número de amostras, *Mi* são as amostras modelo e *Si* são as amostras de referência. Segundo a literatura científica, o modelo é considerado excelente quando o valor do NRMSE é menor que 10%, bom para (10%< NRMSE < 20%) e justo para (20% < NRMSE < 30%) e pobre caso (NRMSE > 30%). (JAMIESON; PORTER; WILSON, 1991; DETTORI et al., 2011).

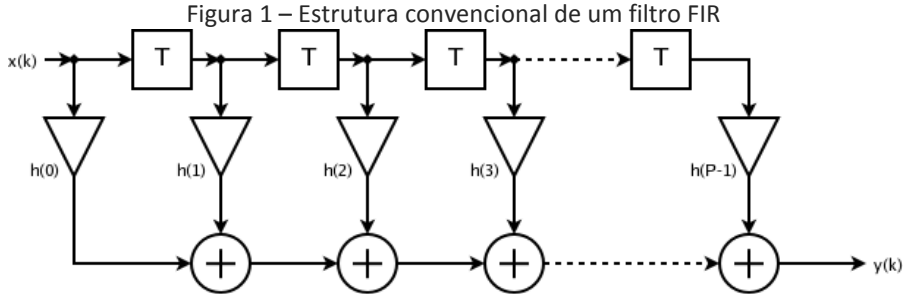
A estrutura de um filtro FIR é expressa matematicamente em termos de uma soma de convoluções, conforme indicado na Eq. (2).

$$y[k] = \sum_{k=0}^{M-1} b_k x[n-k]\,, k = \ 0{,}1{,}2\ \ldots \qquad (2)$$

Onde: M define a ordem do filtro FIR, *n* é o índice da amostra, *k* é o índice do coeficiente e b_k são os coeficientes do filtro.

A representação da estrutura convencional de um filtro FIR, apresentada na Figura 1, é implementada a partir da interpretação da Equação 2 do filtro discreto, onde $x(k)$ representa o sinal de entrada a ser filtrado, T o fator de atraso e $y(k)$ o sinal de saída processado.

Figura 1 – Estrutura convencional de um filtro FIR



Fonte: ASSEF (2016).

O projeto e avaliação do filtro digital foi realizado com o auxílio da ferramenta de projeto e avaliação de filtros digitais FDATool, disponibilizada no Matlab. Os parâmetros especificados para o projeto do filtro constam na Tabela 1.

Tabela 1 – Parâmetros do filtro digital FIR passa-baixa

Parâmetro	Especificação
Frequência de amostragem – Fs	40 MHz
Método do filtro	FIR – *Generalized Equiripple*
Frequência de banda passante	1 MHz
Frequência de banda de parada	8,0 MHz
Atenuação de banda passante	-1dB
Atenuação de banda de parada	-50dB

Fonte: ASSEF (2016).

Com os parâmetros definidos, utilizou-se a ferramenta FDATool para obter os coeficientes de filtragem, resultando em um filtro simétrico de 16 coeficientes:-0,0081, -0,0203, -0,0288, -0,0165, 0,0304, 0,1085, 0,1934, 0,2489, 0,2489, 0,1934, 0,1085, 0,0304, -0,0165, -0,0288, -0,0203 e -0,0081.

RESULTADOS E DISCUSSÕES

O filtro FIR passa-baixa foi implementado em código Python (vide Apêndice) seguindo a estrutura convencional apresentada anteriormente. Os coeficientes foram utilizados conforme os dados obtidos pela ferramenta FDATool, assim como os valores para a criação dos vetores relacionados aos dados de entrada dos sinais analisados. Os dados de saída gerados no programa Python foram comparados com os resultados do Matlab. Os tempos de processamento do filtro digital no Raspberry Pi, desde a entrada até a saída da referida variável, são demonstrados na Tabela 2.

Tabela 2 - Tempo de processamento de dados

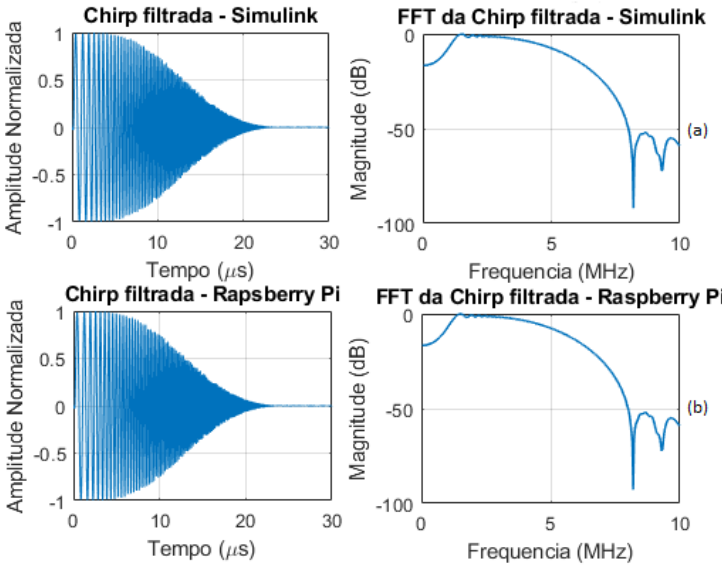
Sinal	Tempo
Senoide	0,80706 s
US	0,73965 s

Fonte: Autoria própria (2018).

O Raspberry mostrou uma boa capacidade de processamento com o algoritmo. Com base nos resultados obtidos através das simulações demonstradas neste trabalho, pode-se comprovar o desempenho coerente do sistema, conforme as FFTs obtidas e a avaliação do filtro FIR através da aplicação do NRMSE.

Apresenta-se também, a comparação do sinal de Chirp no Simulink (a) e no Raspberry Pi (b), onde os componentes de maiores frequências foram atenuados após a filtragem (Figura 2).

Figura 2 – Sinal Chirp processado e sua respectiva FFT. (a) Simulink (b) Raspberry Pi



Fonte: Autoria própria (2018).

Na Figura 3, observa-se os resultados do sinal de US, com a atenuação do componente em 8 MHz em mais de 30 dB após a filtragem com processamento no Simulink (a) e no Raspberry Pi (b), respectivamente.

Figura 3 – Sinal US processado e sua respectiva FFT: (a) Simulink e (b) Raspberry Pi.

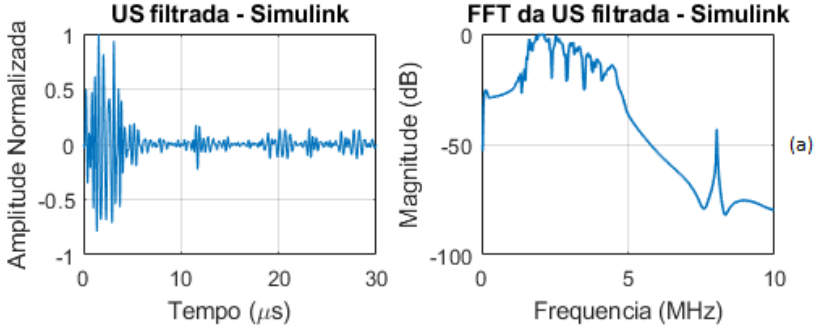
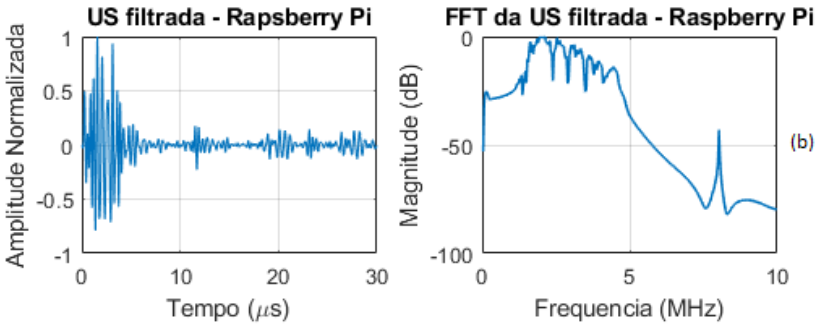


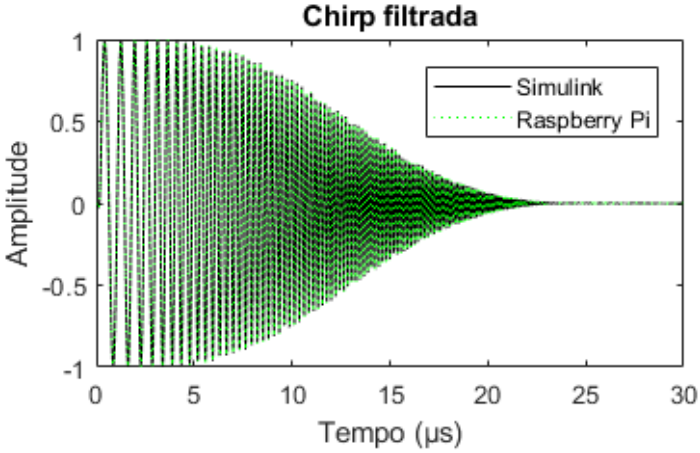
Figura 3 – (continuação) Sinal US processado e sua respectiva FFT: (a) Simulink e (b) Raspberry Pi



Fonte: Autoria própria (2018).

A NRMSE computada para o sinal Chirp ficou igual a 0.026009%. Para melhor ilustração, na Figura 4 são apresentados os sinais sobrepostos, comprovando os perfis de ondas similares.

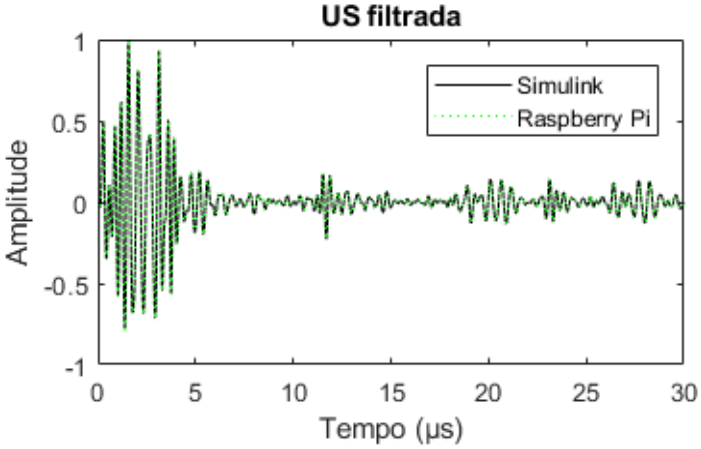
Figura 4 – Comparação do sinal Chirp



Fonte: Autoria própria (2018).

A Figura 5 ilustra a função de custo NRMSE no sinal de US, com um excelente resultado. Neste teste o valor da NRMSE foi igual a 1.6165e-12%.

Figura 5 - Comparação do sinal de US



Fonte: Autoria própria (2018).

CONCLUSÕES

Pode-se concluir que o objetivo desta pesquisa, estudo da ferramenta Raspberry Pi Model B+ para aplicações de filtragem digital de sinais de US em comparação ao *software* Matlab/Simulink foi atingido com sucesso. O modelo do filtro FIR implementado apresentou resultados excelentes, semelhantes ao Matlab, ficando comprovado o desempenho coerente do sistema e evidenciado a correta funcionalidade do filtro com a atenuação do componente em 8 MHz. Os valores resultantes da NRMSE corroboram com o resultado de referência, apresentando um valor muito pequeno no sinal de US, comprovando os resultados de forma qualitativa e quantitativa. Com relação ao tempo de processamento medido para a apuração dos resultados, este deixou a desejar.Nesse sentido, possivelmente, poderá se obter um tempo menor com a utilização de um modelo de Raspberry Pi mais atual com maior capacidade e frequência de processamento.

REFERÊNCIAS

ASSEF, A. A. **Arquitetura de hardware multicanal reconfigurável com excitação multinível para desenvolvimento e testes de novos métodos de geração de imagens por ultrassom**. 2013. Universidade Tecnológica Federal do Paraná, 2013. Disponível em: <http://repositorio.utfpr.edu.br:8080/jspui/handle/1/889>. Acesso em: 7 jun. 2018.

ASSEF, A. A. et al. **Projeto de um filtro digital FIR passa-baixa em FPGA para aplicações de processamento de sinais de ultrassom**. In: Congresso Brasileiro de Engenharia Biomédica - CBEB, 2016, Foz do Iguaçu.

JAMIESON, P. D.; PORTER, J. R.; e WILSON, D. R. **A test of the computer simulation model ARCWHEAT 1 on wheat crops grown in New Zealand.Field Crops Research**, v. 27, p. 337–350, 1991. Disponível em: <http://ac.els-cdn.com/0378429091900403/1-s2.0-0378429091900403-main.pdf?_tid=76aacf02-11e7-86fa-00000aacb362&acdnat=1490216559_74f91ba8c5e39d9 e325c97657487b0fd>. Acesso em: 22 mai. 2018.

AGRADECIMENTOS

Ao CNPq, FINEP, Fundação Araucária, CAPES, UTFPR e Ministério da Saúde pelo apoio financeiro ao desenvolvimento da pesquisa.

APÊNDICE

```
---------------------------------------------------------------------------------------------------------------------
# Convolução
---------------------------------------------------------------------------------------------------------------------
y[0]=x[0]*a[0];
y[1]=x[1]*a[0] + x[0]*a[1];
y[2]=x[2]*a[0] + x[1]*a[1]+ x[0]*a[2];
y[3]=x[3]*a[0] + x[2]*a[1]+ x[1]*a[2]+ x[0]*a[3];
y[4]=x[4]*a[0] + x[3]*a[1]+ x[2]*a[2]+ x[1]*a[3]+ x[0]*a[4];
```

```
y[5]=x[5]*a[0] + x[4]*a[1]+ x[3]*a[2]+ x[2]*a[3]+ x[1]*a[4]+ x[0]*a[5];
y[6]=x[6]*a[0] + x[5]*a[1]+ x[4]*a[2]+ x[3]*a[3]+ x[2]*a[4]+ x[1]*a[5]+ x[0]*a[6];
y[7]=x[7]*a[0] + x[6]*a[1]+ x[5]*a[2]+ x[4]*a[3]+ x[3]*a[4]+ x[2]*a[5]+ x[1]*a[6]+ x[0]*a[7];
y[8]=x[8]*a[0] + x[7]*a[1]+ x[6]*a[2]+ x[5]*a[3]+ x[4]*a[4]+ x[3]*a[5]+ x[2]*a[6]+ x[1]*a[7]+ x[0]*a[8];
y[9]=x[9]*a[0] + x[8]*a[1]+ x[7]*a[2]+ x[6]*a[3]+ x[5]*a[4]+ x[4]*a[5]+ x[3]*a[6]+ x[2]*a[7]+ x[1]*a[8]+ x[0]*a[9];
y[10]=x[10]*a[0] + x[9]*a[1]+ x[8]*a[2]+ x[7]*a[3]+ x[6]*a[4]+ x[5]*a[5]+ x[4]*a[6]+ x[3]*a[7]+ x[2]*a[8]+
x[1]*a[9]+ x[0]*a[10];
y[11]=x[11]*a[0] + x[10]*a[1]+ x[9]*a[2]+ x[8]*a[3]+ x[7]*a[4]+ x[6]*a[5]+ x[5]*a[6]+ x[4]*a[7]+ x[3]*a[8]+
x[2]*a[9]+ x[1]*a[10]+ x[0]*a[11];
y[12]=x[12]*a[0] + x[11]*a[1]+ x[10]*a[2]+ x[9]*a[3]+ x[8]*a[4]+ x[7]*a[5]+ x[6]*a[6]+ x[5]*a[7]+ x[4]*a[8]+
x[3]*a[9]+ x[2]*a[10]+ x[1]*a[11]+ x[0]*a[12];
y[13]=x[13]*a[0] + x[12]*a[1]+ x[11]*a[2]+ x[10]*a[3]+ x[9]*a[4]+ x[8]*a[5]+ x[7]*a[6]+ x[6]*a[7]+ x[5]*a[8]+
x[4]*a[9]+ x[3]*a[10]+ x[2]*a[11]+ x[1]*a[12]+ x[0]*a[13];
y[14]=x[14]*a[0] + x[13]*a[1]+ x[12]*a[2]+ x[11]*a[3]+ x[10]*a[4]+ x[9]*a[5]+ x[8]*a[6]+ x[7]*a[7]+ x[6]*a[8]+
x[5]*a[9]+ x[4]*a[10]+ x[3]*a[11]+ x[2]*a[12]+ x[1]*a[13]+ x[0]*a[14];

for i in range(15, 1201, 1):

y[i]=x[i]*a[0] + x[i-1]*a[1]+ x[i-2]*a[2]+ x[i-3]*a[3]+ x[i-4]*a[4]+ x[i-5]*a[5]+ x[i-6]*a[6]+ x[i-7]*a[7]+ x[i-8]*a[8]+
x[i-9]*a[9]+ x[i-10]*a[10]+ x[i-11]*a[11]+ x[i-12]*a[12]+ x[i-13]*a[13]+x[i-14]*a[14]+x[i-15]*a[15];
print (y)
```