Projeto de Graduação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

30 de Junho de 2015

DEMODULAÇÃO FSK POR MEIO DE FPGA

Rudah Maciel Guedes

PUC RIO

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

DEMODULAÇÃO FSK POR MEIO DE FPGA

Aluno(s): Rudah Maciel Guedes

Orientador(es): Marco Antônio Grivet

Trabalho apresentado com requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Agradeço a Alexandre Fabres pela ideia do projeto e pelas sugestões, orientação e incentivo, a Gustavo Amaral por conceder espaço e material para o desenvolvimento do projeto, por me ensinar programação de FPGA e por me orientar e ajudar, e a Rafael Aoude e Victor Takahashi, por me ajudarem e contribuírem para o projeto de diversas formas ao longo de todo o processo.

Resumo

A modulação FSK é amplamente utilizada para a transmissão digital de dados em diversas áreas. No caso do monitoramento de dutos de transporte de petróleo por meio de sensores, o sistema de transmissão possui limitações físicas, como altas temperaturas e pressão, de modo que é restrito à métodos de transmissão mais simples, como a modulação FSK binária (B-FSK). Torna-se, portanto, fundamental que o sistema receptor seja capaz de realizar uma demodulação eficiente. Uma série de métodos pode ser considerada para essa tarefa, dentre eles o algoritmo Goertzel. Esse algoritmo é capaz de implementar um estreito filtro passa-banda sobre o sinal de entrada identificando a presença ou ausência de uma componente de frequência. Nesse contexto, o Field Programable Gate Array (FPGA) representa uma solução robusta de baixo custo para a demodulação de sinais FSK - através do algoritmo Goertzel - principalmente por conta da possibilidade de paralelização das estruturas digitais que tornam a execução do algoritmo menos demorada. Para a aplicação do algoritmo no FPGA foram desenvolvidas unidades de multiplicação e soma em ponto flutuante. Essa abordagem mostrou-se rápida – 1 ciclo de clock para a multiplicação e 4 ciclos de clock para a soma em ponto flutuante – e, ao mesmo tempo, custosa em termos dos recursos disponíveis na FPGA. Utilizando um conversor analógico-digital trabalhando a uma taxa de 500 mil samples por segundo na estação receptora e um transmissor operando a uma taxa de transmissão de 2500 bits/s foi possível observar que a aplicação do algoritmo Goertzel no FPGA com operações em ponto flutuante é eficaz, apresentando uma taxa de erros nula na demodulação de sinais BFSK de frequências entre 5 kHz e 200 kHz.

Palavras-chave: FPGA; FSK; Goertzel; Demodulação;

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

FSK demodulation using FPGA

Abstract

FSK modulation is widely used for digital transmission of data in several areas. In the case of oil transport ducts, monitored by sensors, the transmission system has physical limitations such as, high temperatures and high pressure. Due to these limitations, the system is restricted to simpler transmission methods, like binary FSK (BFSK) modulation. It is therefore fundamental that the receptor system is able to perform an efficient demodulation. Among the possible methods that could be considered for the task, the Goertzel algorithm was chosen. This algorithm can implement a narrow band-pass filter over the input signal, identifying the presence or the absence of a frequency's component. In this context, the Field Programmable Gate Array (FPGA) represents a robust low-cost solution for the FSK signals demodulation – through the Goertzel algorithm – mainly because of the possibility of parallelizing the digital structures that makes the algorithm's execution faster. For the application of the algorithm in the FPGA, floating-point sum and multiplication units were developed. This approach is fast – 1 clock cycle for the multiplication and 4 clock cycles for the sum – but expensive in terms of the FPGA's available resources. It was observed that, through the use of an analog-digital converter that works on a 500k samples per second rate in the receptor station and a transmitter operating at a 2500 bits per second rate, the application of the Goertzel algorithm in the FPGA with floating-point operations was effective. This presented a null error rate in the BFSK demodulation of frequencies between 5 kHz and 200 kHz.

Keywords: FPGA; FSK; Goertzel; Demodulation;

Sumário

Lista de figuras

1. Introdução

1.1. Motivação

Na área de extração de petróleo, tanto em plataformas terrestres como em plataformas off-shore, é fundamental que se faça uma análise constante do estado dos dutos de transporte durante sua operação. O monitoramento permite a realização de um diagnóstico de eventuais problemas - como corrosão - evitando problemas graves - como a ruptura do duto - que podem causar a perda de todo o sistema. Além de evitar um grande prejuízo para as empresas de prospecção, um monitoramento constante pode impedir que desastres ambientais ocorram [1].

O monitoramento é realizado por meio de sensores que são instalados no interior do duto e se comunicam com uma estação central através de fibras ópticas. Tanto os sensores quanto as fibras devem suportar temperaturas muito altas, estando muitas vezes a dezenas de quilômetros da superfície [1]. As condições extremas de temperatura e pressão bem como a distância elevada entre transmissor e receptor impõem uma Relação Sinal-Ruído (SNR) muito estreita, o que limita o sistema de comunicação. A própria natureza da informação gerada pelos sensores, contudo, não exige uma banda de transmissão larga, de modo que o sistema pode optar por transmitir os dados de forma simples almejando alcançar menores taxas de erro [2]. Uma opção interessante nesse contexto é a modulação FSK binária (B-FSK).

Apesar do transmissor não demonstrar complexidade elevada, o projeto do receptor deve ser robusto o suficiente para compensar o baixíssimo SNR fruto do ruído que corrompe o sinal e da atenuação intrínseca que está envolvida na transmissão por longas distâncias. O algoritmo Goertzel apresenta uma solução robusta para a demodulação B-FSK, pois é simples e mais eficiente se comparado a outros métodos como a própria FFT (ver seção 1.3).

O algoritmo Goertzel implementa um Filtro IIR que seleciona componentes de frequência dentro de uma faixa estreita. Tanto a frequência central quanto a banda do filtro são determinadas por parâmetros do sistema, os coeficientes do filtro e o limite do decisor [3]. O algoritmo, portanto, identifica a presença ou ausência de uma determinada frequência no sinal recebido. Utilizando uma estrutura de processamento paralelo como o Field Programable Gate Array (FPGA) [4], é possível realizar o algoritmo paralelamente, identificando frequências diferentes simultaneamente no sinal, o que permite que a demodulação seja mais precisa e que mais de um canal seja demodulado simultaneamente.

1.2. FPGA

O FPGA (Field Programmable Gate Array) é um hardware definido por software sendo, portanto, veloz e flexível. Essa plataforma permite montar circuitos digitais dentro de um chip sem a necessidade de conectar os componentes por fios manualmente. Isso é feito através da programação de uma estrutura de memória que controla as conexões dos blocos lógicos. A linguagem utilizada para a programação é a VHDL – VHSIC Hardware Description Language. Ao contrário das linguagens de programação convencionais, como linguagem C, o código é interpretado e é gerado um arquivo de configuração num processo conhecido como *Place And Route (PAR)*. Esse arquivo contém as instruções de controle dos elementos do FPGA, mostrados na figura abaixo [5].

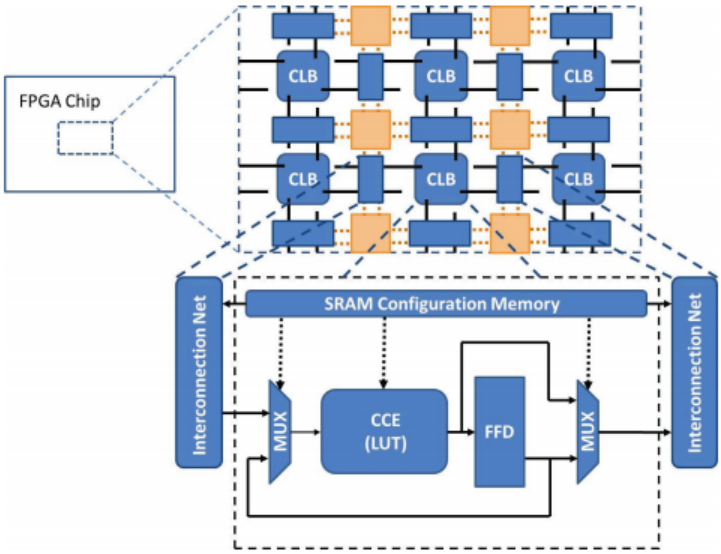

Figura 1.1 – diagrama interno do FPGA

Os CLBs são *Configurable Logic Blocks*, que são conectados através de redes de interconexão programável (*Interconnection Net*). No interior dos CLBs os *Configurable Combinational Elements* (*CCEs*) são conectados a flip-flops D e as entradas e saídas são criadas através de uma combinação dos sinais internos e externos nos multiplexadores digitais. Todas as estruturas programáveis e configuráveis são estabelecidas pela SRAM (*Static Random Access Memory*) [5].

Uma das características mais interessantes do FPGA é o processamento paralelo. O FPGA pode ser conectado a múltiplas entradas e saídas e processá-las individualmente de forma simultânea, o que representa uma grande vantagem sobre estruturas seriais como os microcontroladores [5].

A capacidade de processamento do FPGA reside no número de células e na frequência de clock máxima permitida pela estrutura. A primeira geração de FPGAs da Xilinx tinha até 100 CLBs e frequência de clock de até 50 MHz. Atualmente, placas disponíveis no mercado, como a Virtex-7, podem suportar até 2 milhões de CLBs e até 500 MHz de frequência de clock [5].

1.3. Modulação FSK

Em contraste com sistemas de transmissão analógicos, em que o sinal correspondente à mensagem é diretamente usado na modulação, em sistemas digitais a mensagem é codificada em bits, que são então transmitidos através de um entre vários protocolos [2]. Um deles é o chamado Frequency Shift Keying (FSK), ou chaveamento de frequências, em que a portadora carrega os dados codificados na transição entre frequências. A portadora, ou sinal modulado, é um sinal senoidal que tem sua frequência alterada de acordo com o estado do bit (ou bits) a ser transmitido. A figura abaixo é um exemplo de modulação FSK binária. Ela leva esse nome porque um bit é transmitido de cada vez, de forma que a transição é realizada entre apenas duas frequências, ou estados.

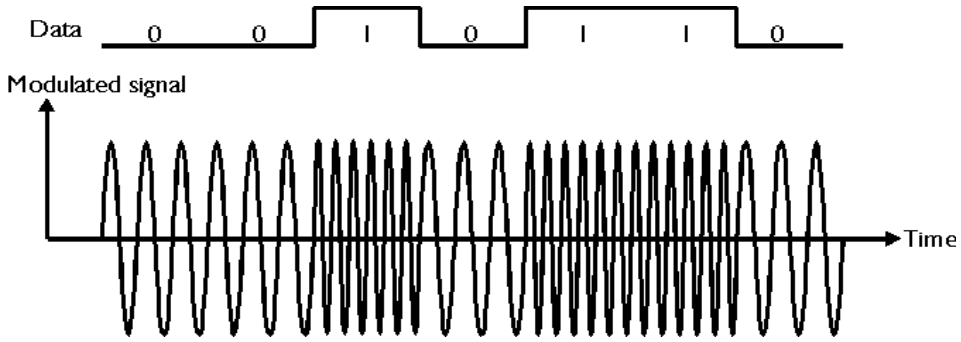

Figura 1.2 – modulação FSK binária

Na modulação FSK binária uma frequência é associada a cada estado do sinal de informação (bit um e bit zero). O sistema receptor deve identificar o bit enviado de acordo com a frequência do sinal recebido.

1.4. Algoritmo Goertzel

O algoritmo Goertzel utiliza um método iterativo para o cálculo de termos individuais da DFT de N pontos de um sinal discreto. Como no cálculo direto da DFT, o algoritmo atribui uma componente espectral para uma determinada frequência do sinal, por outro lado, contudo, utiliza coeficientes constantes reais e aritmética real [6]. Ele é mais eficiente que o cálculo direto de DFT e mais eficiente que o método FFT quando o número de componentes espectrais que se deseja analisar é menor que $\log_2 N$. Outra vantagem do algoritmo é o fato de que não é necessário que N seja potência de dois [3]. A figura 1.2 mostra a DFT de N pontos de um sinal discreto x. O algoritmo Goertzel calcula um termo individual da DFT, definido como o k-ésimo termo.

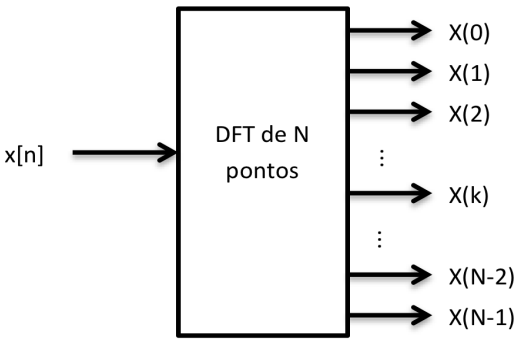

Figura 1.3 – DFT de N pontos

A figura 1.3 abaixo mostra o diagrama de blocos do algoritmo.

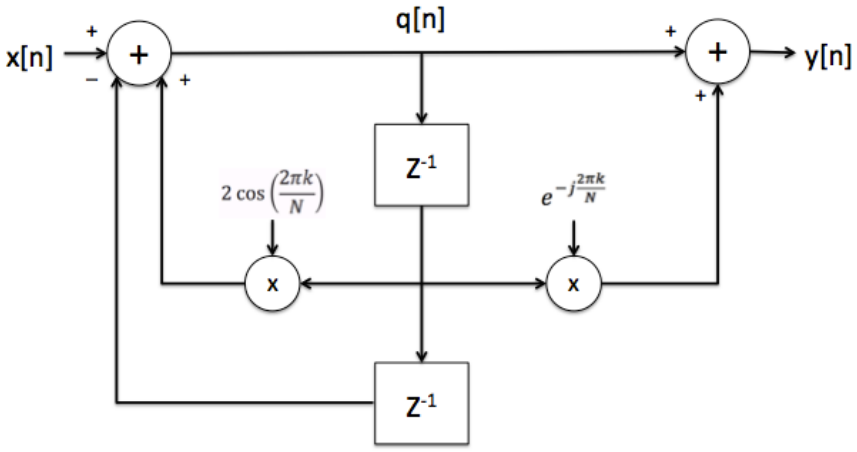

Figura 1.4 – Algoritmo Goertzel: diagrama de blocos

O vetor *q* que aparece na figura 1.3 é um sinal intermediário que armazena memória das amostras de *x* ao longo das iterações do algoritmo. *k* é definido por

$$k = N\frac{f_T}{f_s} \tag{1.1}$$

onde f_s é a frequência de amostragem do sinal modulado *x* e f_T é chamada de frequência *target*: é a frequência que se deseja identificar no sinal *x*, referente ao k-ésimo termo da DFT.

O algoritmo é definido por duas equações:

$$q[n] = 2\cos\left(\frac{2\pi k}{N}\right)q[n-1] - q[n-2] + x[n] \tag{1.2}$$

$$y[n] = q[n] - e^{-j\frac{2\pi k}{N}}q[n-1] \tag{1.3}$$

O objetivo da implementação do algoritmo é a identificação da presença ou ausência da frequência f_T no sinal *x*. Uma forma de realizar a identificação é através da análise da magnitude de $X(k)$, que é dado por:

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi k}{N}n} \tag{1.4}$$

Aplicando a Transformada Z em (1.2) e (1.3) encontra-se a função de transferência do filtro:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - e^{j\frac{2\pi k}{N}}z^{-1}} \tag{1.5}$$

Observa-se que o algoritmo é um filtro IIR de primeira ordem, formado pela ligação em série de um filtro IIR de segunda ordem ($H_1 = \frac{Q(Z)}{X(Z)}$) e um filtro FIR de primeira ordem ($H_2 = \frac{Y(Z)}{Q(Z)}$).

Em seguida aplica-se a Transformada Z inversa, obtendo-se a resposta impulsional:

$$h[n] = e^{j\frac{2\pi k}{N}n}\,u_{-1}[n] \tag{1.6}$$

onde $u_{-1}[n]$ é o degrau unitário. O sinal de saída *y* é definido como a convolução da entrada com a resposta impulsional:

$$y[n] = \sum_{r=-\infty}^{\infty} x[r]\, h[n-r] \tag{1.7}$$

$$y[n] = \sum_{r=-\infty}^{\infty} x[r] e^{j\frac{2\pi k}{N}(n-r)} u_{-1}[n-r] \tag{1.8}$$

$$y[n] = \sum_{r=0}^{N-1} x[r] e^{j\frac{2\pi k}{N}(n-r)} \tag{1.9}$$

Multiplicando o lado direito da equação 1.4 por $e^{j\frac{2\pi k}{N}N}$, que é igual a um, obtemos:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{j\frac{2\pi k}{N}(N-n)} \tag{1.10}$$

Ou

$$X(k) = \sum_{r=0}^{N-1} x[r] e^{j\frac{2\pi k}{N}(N-r)} \tag{1.11}$$

$$X(k) = \left(x[n] * e^{j\frac{2\pi k}{N}n}\right), n = N \tag{1.12}$$

que é o mesmo que

$$X(k) = \left(x[n] * e^{j\frac{2\pi k}{N}n} u_{-1}[n]\right), n = N \tag{1.13}$$

Logo

$$X(k) = y[N] \tag{1.14}$$

Pode-se, então, calcular a magnitude do k-ésimo termo da DFT de *x* pelo módulo de y[N] na equação 1.3:

$$|X(k)|^2 = q[N-1]^2 + q[N-2]^2 - q[N-1]q[N-2]\, 2\cos\left(\frac{2\pi k}{N}\right) \tag{1.15}$$

O quadrado da magnitude de $X(k)$ é calculado após N iterações do algoritmo, com N amostras de *x*. O valor da magnitude varia de acordo com a frequência das N amostras de *x* e com a frequência *target* definida. Se a frequência de $x[n]$ a $x[n+N-1]$ for igual a f_T então $|X(k)|^2$ apresentará um valor alto, se comparado ao valor apresentado quando as duas frequências forem diferentes. A figura 1.5 mostra a distribuição da magnitude em função do índice $m = k$, referente ao m-ésimo termo da DFT [6].

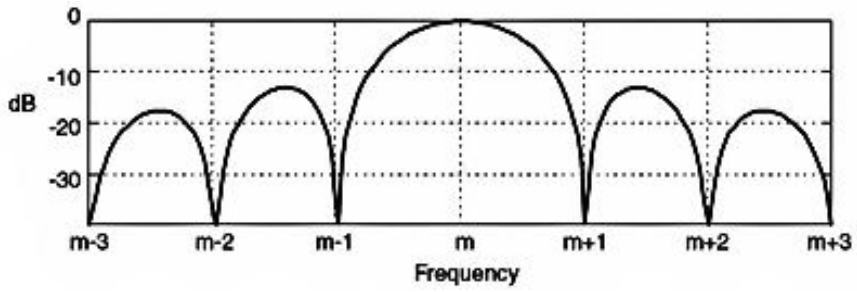

Figura 1.5 – Algoritmo Goertzel: magnitude em função do índice da DFT

A partir da figura 1.5 e da equação 1.1 obtém-se a relação

$$f_{k\pm1} = f_T \pm \frac{f_s}{N} \tag{1.16}$$

A partir da equação 1.16 é possível calcular o par de frequências relativo aos primeiros mínimos da distribuição de magnitudes em relação à frequência central (*target*).

Antes de implementar o demodulador é necessário fazer uma rodada de testes com o algoritmo. O teste consiste no envio de bits previamente conhecidos e na análise das magnitudes apresentadas. Observa-se os valores de $|X(k)|^2$apresentados para cada estado de bit para que se possa estabelecer um limiar de decisão.

O decisor funciona da seguinte forma: suponha que $f_T = f_{bit1}$. Envia-se uma sequência de, por exemplo, dez bits 1 e dez bits zero. O menor valor de$|X(k)|^2$dentre os dez referentes aos bits 1 é de, por exemplo, dez mil, enquanto que o maior valor de $|X(k)|^2$dentre os dez referentes aos bits 0 é de dez. Podemos aplicar um limiar de decisão centrado num valor intermediário entre os dois valores, por exemplo cinco mil. Dessa forma, quando $|X(k)|^2$ apresentar um valor acima de cinco mil, um bit 1 será atribuído às N amostras. Caso contrário, será atribuído um bit 0.

Essa é a maneira mais simples de se utilizar o algoritmo Goertzel para demodulação FSK binária. O diagrama abaixo mostra as etapas do processo de demodulação de 1 bit (para L bits, o processo abaixo deve ser realizado L vezes). Considere que $q[n] = q0$, $q[n-1] = q1$, $q[n-2] = q2$ e $2\cos\left(\frac{2\pi k}{N}\right) = coef$. No diagrama, é considerado que o cálculo de $|X(k)|^2$ pode ser executado paralelamente e que ele é mais rápido que N iterações do algoritmo.

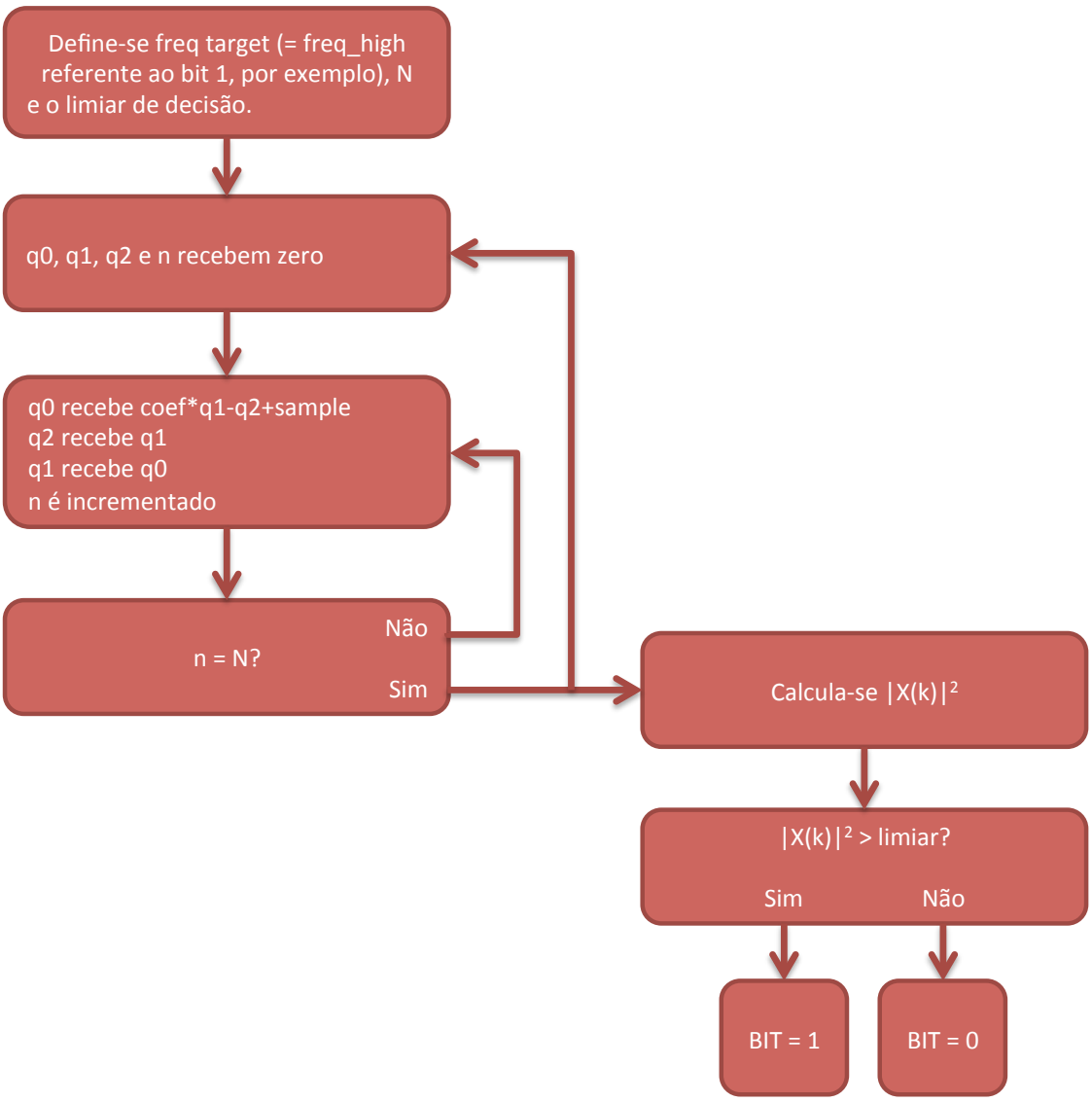

Figura 1.6 – Algoritmo Goertzel: diagrama de demodulação

Se a frequência *target* fosse definida como a frequência referente ao bit 0 as saídas do decisor seriam invertidas.

2. Estruturação do Algoritmo Goertzel no FPGA

2.1. Representação em ponto flutuante

A função do demodulador Goertzel é identificar bits em um sinal senoidal discreto. Cada ponto desse sinal é dado por um número real. A eletrônica digital trabalha apenas com bits, ou seja, zeros e uns, logo é necessária uma representação digital de números reais para que se possa realizar operações matemáticas com o sinal. A representação utilizada é chamada *ponto flutuante*.

Atualmente o padrão mais utilizado é o IEEE 754 [7] que divide o número real *x* da seguinte forma:

$$x = (-1)^{sinal} \cdot mantissa \cdot 2^{expoente} \tag{2.1}$$

Essa representação de 32 bits, que é chamada de representação simples, nos permite fazer operações com números de 2^{-126} a 2^{127}.

O sinal é dado por um bit (‘0’ se positivo, ‘1’ se negativo) ; a mantissa, de 23 bits, representa a parte fracionária. O valor do enésimo bit mais significativo é 2^{-n}, ou seja, o bit mais significativo vale ½, o segundo vale ¼ e assim por diante. Existe um bit "invisível" de valor 1 que ocupa a posição mais significativa da mantissa, porém ele não é armazenado no sinal. Isso significa que o valor da mantissa varia de um a dois ; o expoente é um número de 8 bits subtraído de 127.

A aplicação do algoritmo pode ser resumida em programar o FPGA para realizar as equações 1.2 e 1.15. Essas equações utilizam operações de soma e multiplicação. Dessa forma, é necessário definir no FPGA as unidades básicas de soma e multiplicação de números reais, utilizando a representação em ponto flutuante.

2.2. Unidade de multiplicação

A unidade de multiplicação é responsável por multiplicar dois números na representação em ponto flutuante. Sua saída é dada pela equação abaixo:

$$Multiplier\ Output = S_1 M_1 2^{E_1} \cdot S_2 M_2 2^{E_2} \tag{2.2}$$

Para realizar a multiplicação, as mantissas devem ser multiplicadas, os expoentes devem ser somados e os sinais combinados através da lógica XOR (“ou” exclusivo).

Na multiplicação das mantissas, o bit “invisível” de valor 1 deve ser considerado. Dessa forma, cada mantissa é um número de 24 bits, e o resultado da multiplicação deve ser armazenado num número de até 48 bits. Depois da multiplicação, deve-se alinhar o resultado, o que consiste na eliminação do 1 mais significativo. Os 23 bits mais significativos à direita do primeiro 1 vão compor a mantissa do número resultante da multiplicação. A figura 2.2.1 mostra um exemplo de como esse procedimento funciona.

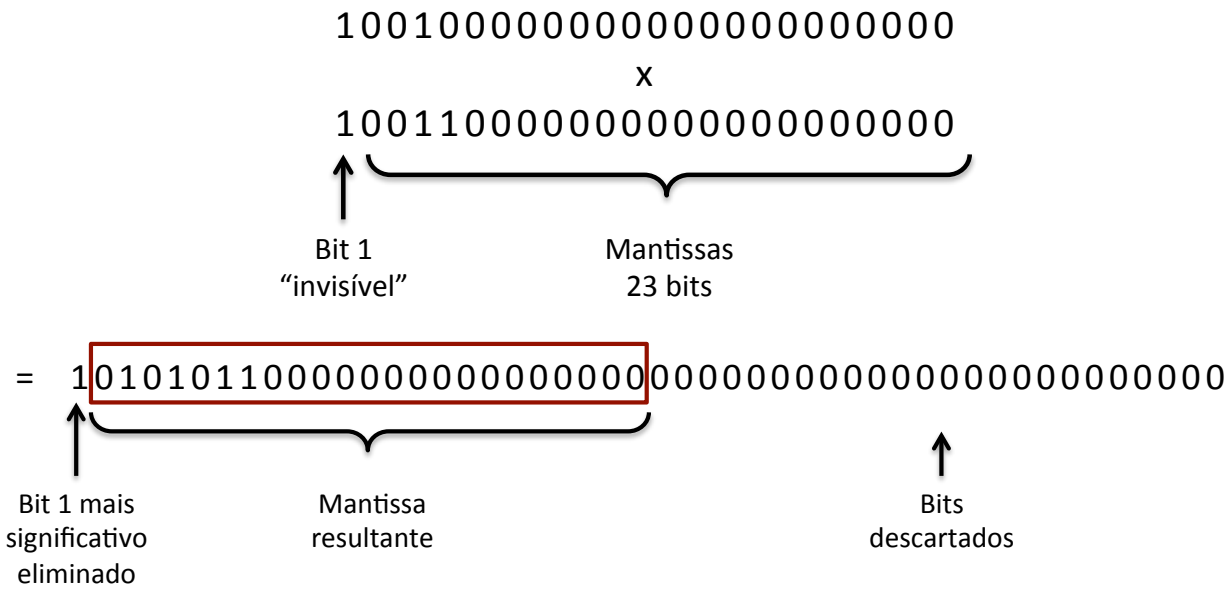

Figura 2.1 – exemplo de multiplicação de mantissas

No exemplo acima, duas mantissas de valores equivalentes a 1.125 e 1.1875, respectivamente, são multiplicadas, resultando num número de 48 bits. Apenas os 23 bits à direita do bit 1 mais significativo compõem a mantissa do resultado. Nesse exemplo o bit 1 mais significativo ocupa a primeira posição do número, mas existem casos onde o bit 1 mais significativo ocupa a segunda posição. Nesses casos o procedimento é o mesmo, com o quadro que compõe a mantissa resultante tendo um deslocamento à direita em relação ao caso do exemplo acima.

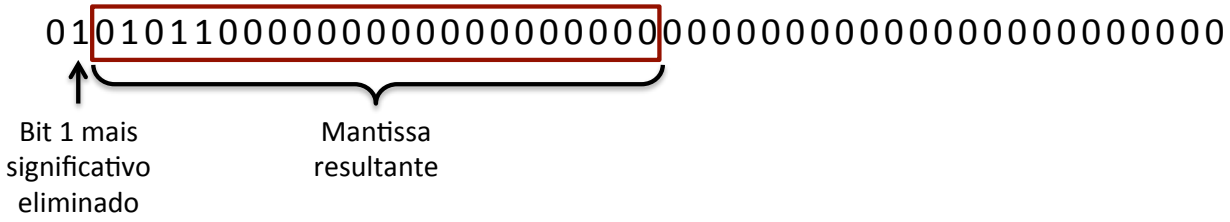

Figura 2.2 – exemplo de bit 1 mais significativo na segunda posição

O deslocamento à direita, que ocorre no caso acima, equivale à uma divisão por 2. Para cancelar esse efeito, deve-se somar 1 ao expoente.

A soma dos expoentes pode ser feita através de somadores completos de 8 bits. Os expoentes têm um *bias* de 127. Dessa forma a soma deve ser feita subtraindo-se 127 de cada expoente e depois somando novamente ao resultado. Isso é o mesmo de somar os expoentes com o *bias* e subtraí-lo depois:

$$E_R = (E_1 - 127) + (E_2 - 127) + 127 = E_1 + E_2 - 127 \quad (2.3)$$

A unidade de multiplicação deve prever casos onde os números são muito pequenos. Dependendo da precisão requisitada, traça-se um limite inferior a partir do qual pode-se considerar que o número é zero.

O diagrama da figura 2.3 mostra o funcionamento do procedimento de multiplicação de dois números reais A e B, representados em ponto flutuante simples (32 bits), a ser implementado no FPGA.

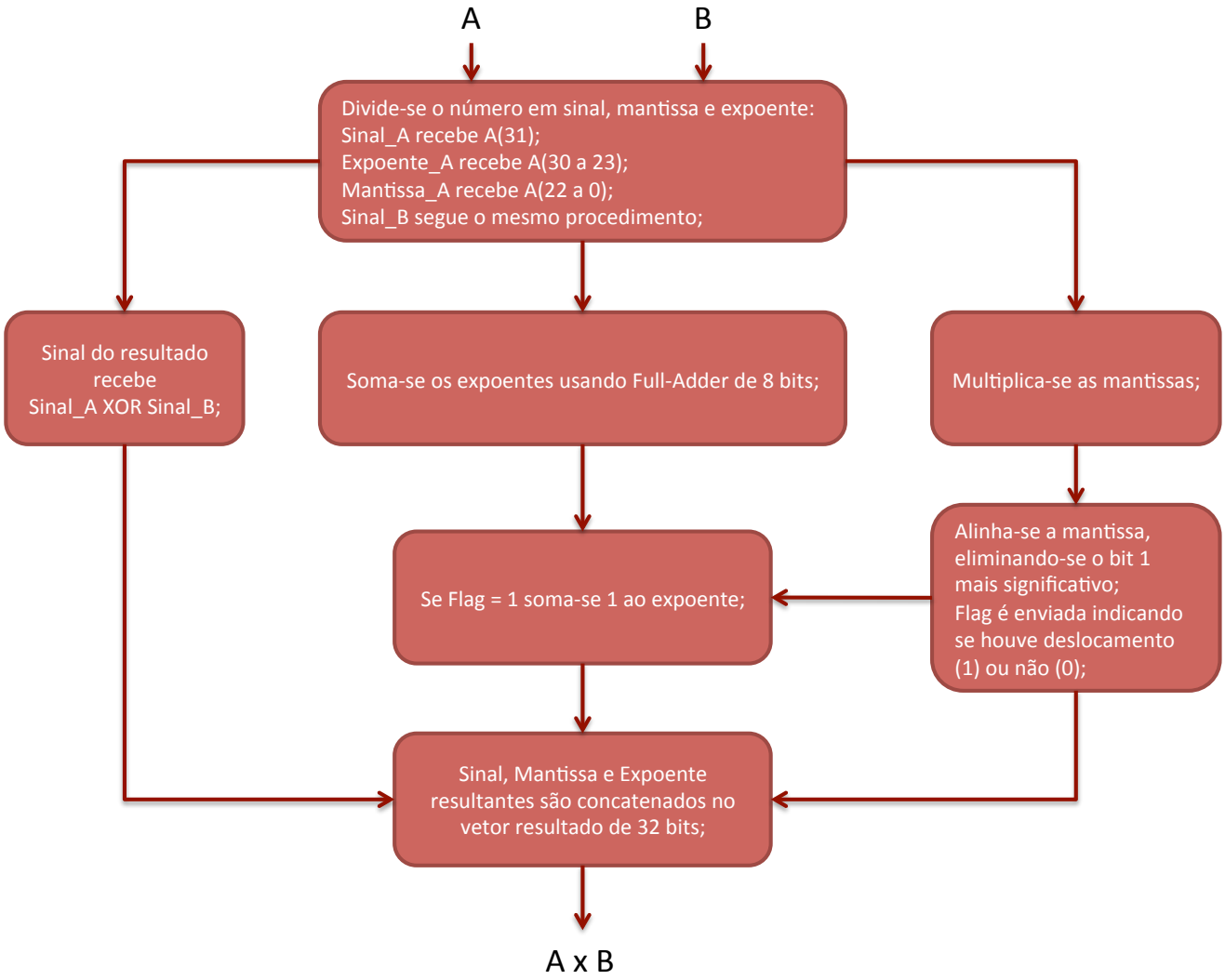

Figura 2.3 – diagrama do procedimento de multiplicação de números float para FPGA

O programa foi desenvolvido sem a utilização de máquinas de estado, executando todas as instruções paralelamente (*pipelined*). A unidade realiza uma multiplicação de dois números float em 1 ciclo de clock.

A figura abaixo mostra a simulação da multiplicação de π com e, cada um com sete casas decimais, realizada pela unidade de multiplicação. A simulação foi feita utilizando-se o software *ISE Degin Suite* da *Xilinx*.

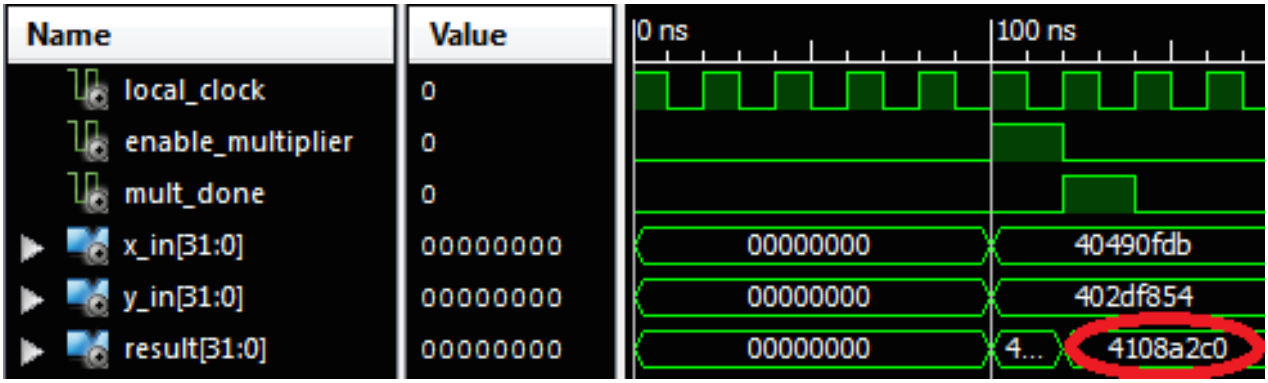

Figura 2.4 – simulação da unidade de multiplicação

Os sinais *enable_multiplier* e *mult_done* indicam quando começa e quando termina a multiplicação. Os valores de *x* e *y* correspondem a 3.1415927 e 2.7182817, respectivamente. O resultado da multiplicação é 8.5397342. O sinal *float_sum* ao final da multiplicação tem valor equivalente a 8.5397340, logo a unidade executou a multiplicação de forma correta até a sexta casa decimal em apenas um ciclo de clock, que na simulação equivale à 20 nano segundos.

2.3. Unidade de soma

Na unidade de soma os expoentes devem ser igualados para que se possa somar as mantissas.

$$Adder\ Output = S_1 M_1 2^{E_1} + S_2 M_2 2^{E_2} \tag{2.4}$$

Se $E_1 = E_2$ a equação fica

$$Adder\ Output = S_{out}(M_1 + M_2)\ 2^{E_1} \tag{2.5}$$

É possível igualar o expoentes através do deslocamento das mantissas. Primeiro compara-se os expoentes. O menor deve ser incrementado para que atinja o mesmo valor do maior. À cada soma do expoente a mantissa deve ser deslocada à direita para cancelar o efeito, completando o número com zeros. Por exemplo, se $E_2 = E_1 + 8$, a mantissa M_1 deve ser deslocada 8 vezes à direita enquanto o expoente é incrementado 8 vezes. Esse exemplo é ilustrado abaixo:

$$\begin{aligned} &1011000000000000000000000 \times 2^{E1} \\ =\ &0101100000000000000000000 \times 2^{E1+1} \\ =\ &0010110000000000000000000 \times 2^{E1+2} \\ &\vdots \\ =\ &0000000101100000000000000 \times 2^{E1+7} \\ =\ &0000000010110000000000000 \times 2^{E1+8} \end{aligned}$$

Figura 2.5 – exemplo de alinhamento de expoentes

Caso um expoente seja mais de 23 vezes menor que o outro, a mantissa de seu número será nula. Nesse caso o número menor pode ser considerado nulo em comparação ao maior: a soma será igual ao número maior.

Depois de igualar os expoentes deve-se definir a operação entre as mantissas (soma ou subtração) baseado no sinal de cada número. Se os sinais forem iguais (ambos positivos ou negativos) as mantissas serão somadas. Se os sinais forem diferentes, a mantissa de menor valor será subtraída da de maior valor. O sinal do resultado depende dos sinais de cada número.

A mantissa resultante deve ser normalizada de forma que a mantissa do resultado seja composta apenas pelos 23 bits à direita do bit 1 mais significativo, como na unidade de multiplicação.

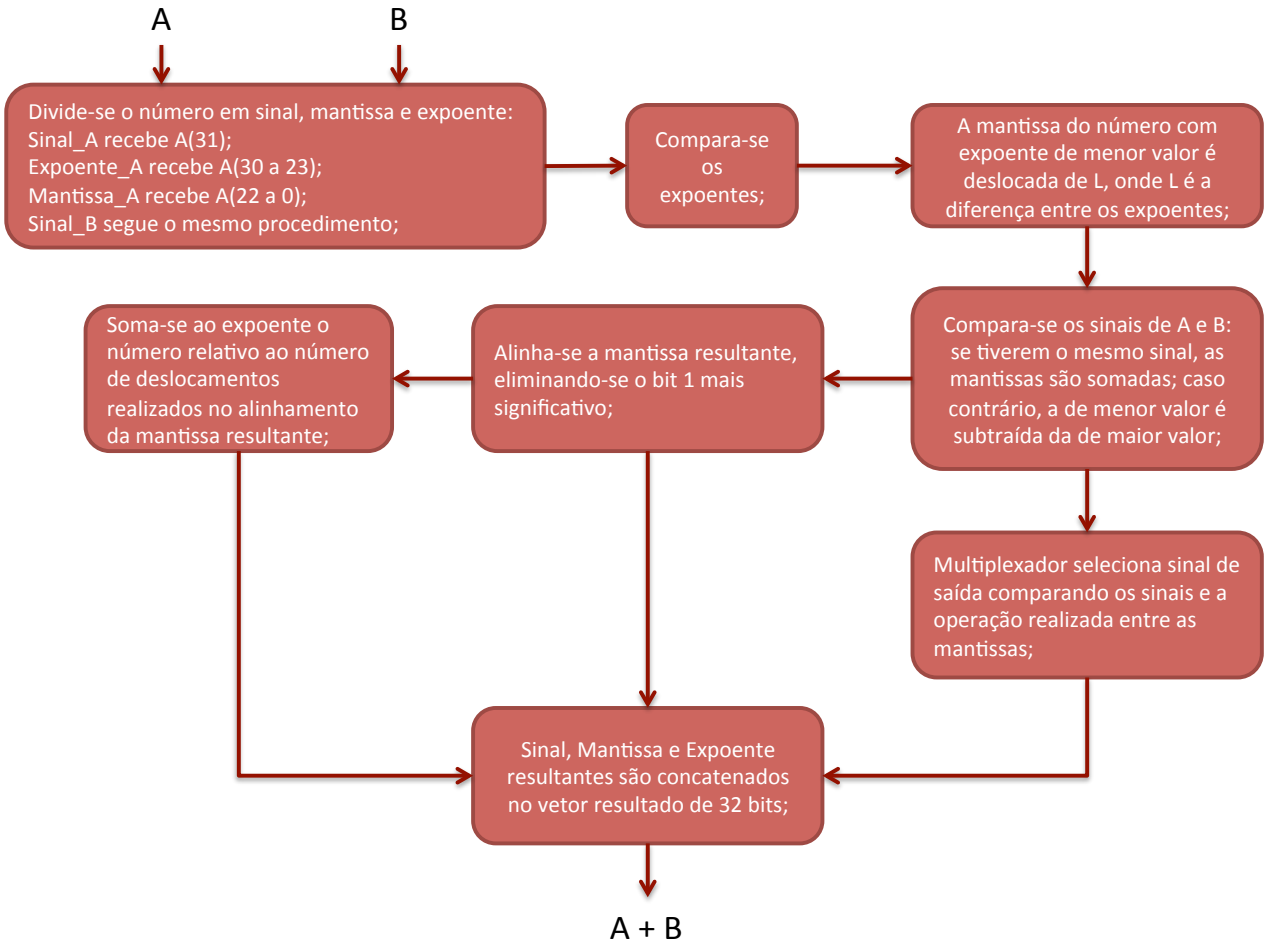

Figura 2.6 - diagrama do procedimento de soma de números float para FPGA

Como na unidade de multiplicação, o programa foi desenvolvido sem a utilização de máquinas de estado, executando todas as instruções paralelamente (pipelined). A unidade realiza uma soma de números em ponto flutuante em 4 ciclos de clock.

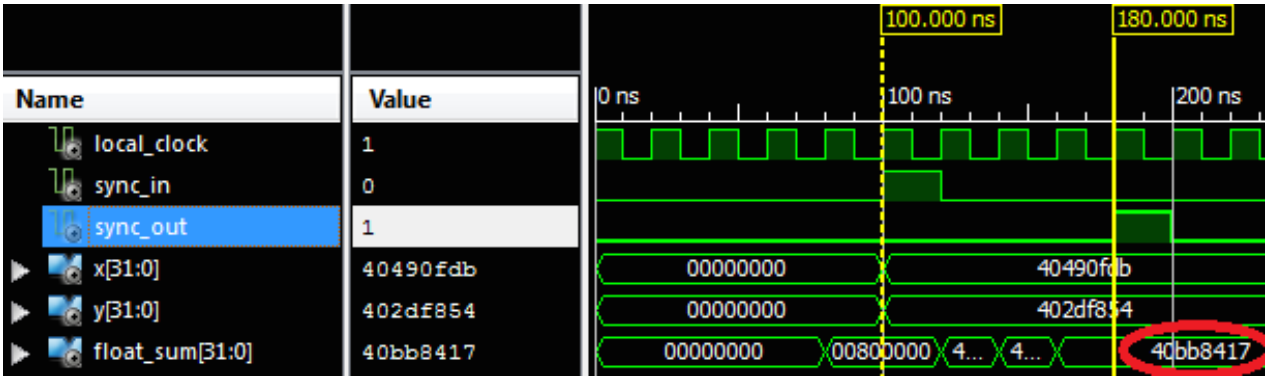

Figura 2.7 – simulação da unidade de soma

Os sinais *sync_in* e *sync_out* são *flags* que indicam, respectivamente, quando a operação começa e termina. Observa-se que a operação dura 80 ns – o equivalente à 4 ciclos de clock de 20 ns. Os sinais *x* e *y* são, respectivamente, π e e, como na simulação da unidade de multiplicação. O resultado da soma desses valores é 5.8598745 e o valor do sinal *float_sum* quando a operação termina é 5.8598742.

2.4. Unidade Goertzel

Definidas as unidades de soma e multiplicação, podemos montar o diagrama de funcionamento do algoritmo no FPGA. A unidade Goertzel é dividida em duas partes: a unidade aritmética, que utiliza a equação 1.2 para a atualização das variáveis *q* a partir das amostras (samples) do sinal FSK recebido, e a unidade de magnitude, que utiliza a equação 1.15 para o cálculo de $|X(k)|^2$. Na figura 2.8 os blocos Q0, Q1 e Q2 são flip-flops tipo D.

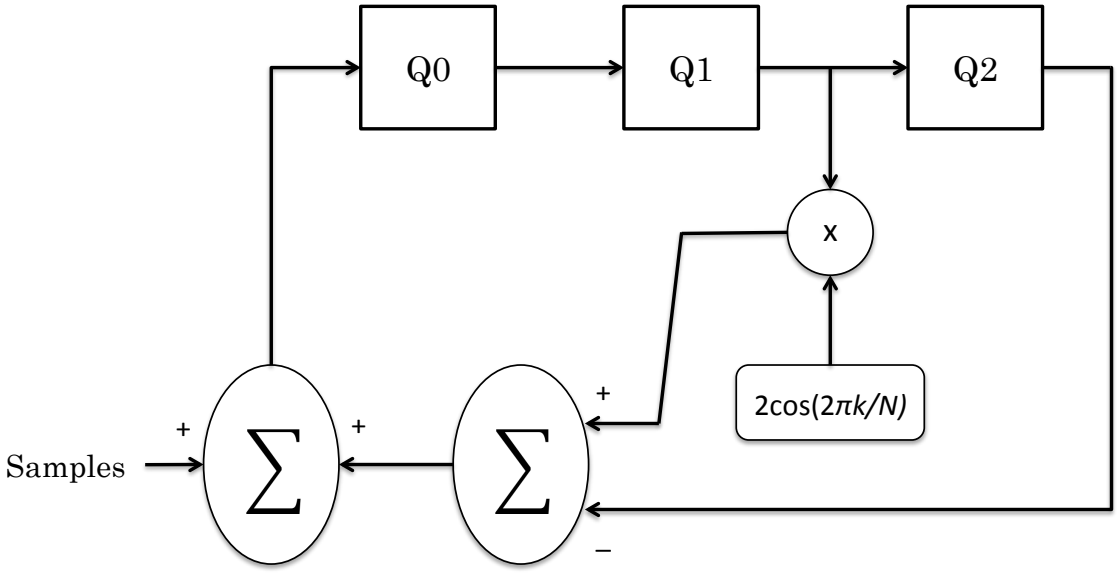

Figura 2.8 – Unidade Aritmética

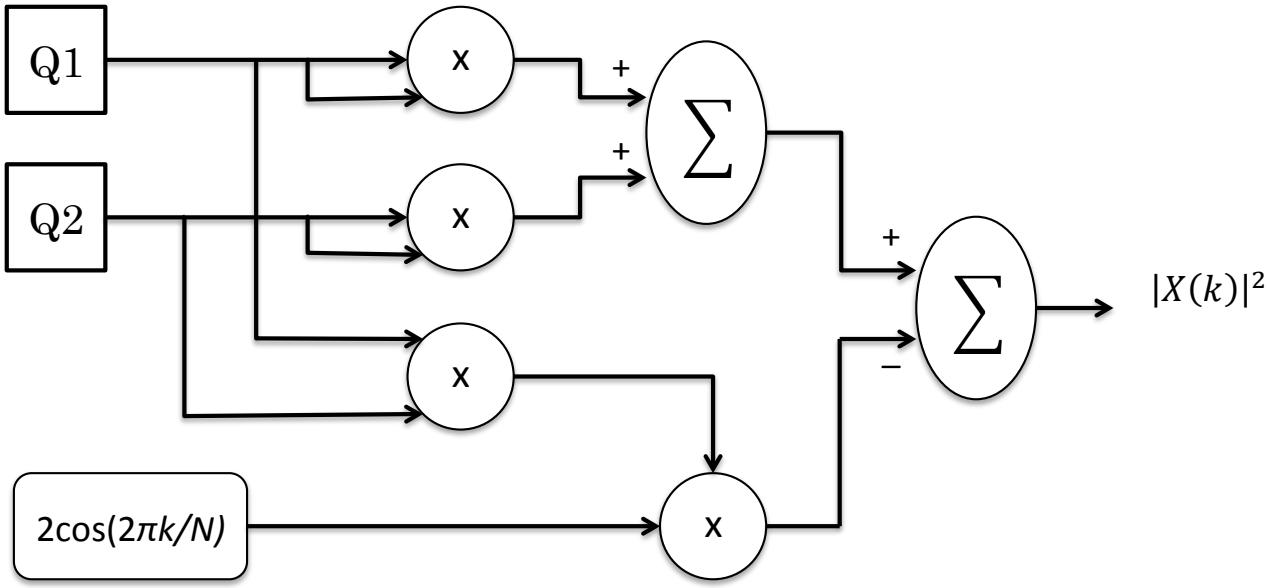

Figura 2.9 – Unidade de Magnitude

As duas partes devem ser executadas paralelamente. Um contador deve indicar quando a N-ésima iteração da unidade aritmética é completada (quando o N-ésimo *sample* é processado). A unidade de magnitude deve então ser habilitada, armazenando os valores atuais de Q1 e Q2 em flip-flops para a realização dos cálculos. A unidade aritmética deve reiniciar, zerando os valores de Q0, Q1, Q2 e do contador para que comece um novo ciclo de N iterações.

3. Simulação

Inicialmente o algoritmo foi testado no Matlab, afim de comprovar seu funcionamento. O teste foi realizado da seguinte forma:

1. cria-se arbitrariamente um vetor de L bits;
2. inicializa-se as frequências atribuídas a cada estado de bit, a frequência de amostragem e a taxa de transmissão de bits;
3. calcula-se N e COEF (onde COEF = $\mathbf{2}\cos\left(\frac{2\pi k}{N}\right)$);
4. gera-se o sinal modulado, corrompendo-o com ruído branco gaussiano, mantendo-se uma relação sinal ruído relativamente alta (acima de 10 dB);
5. realiza-se L rodadas do algoritmo para que se possa delimitar o *threshold*;
6. cria-se um vetor de L bits através de um gerador pseudoaleatório;
7. utiliza-se o mesmo procedimento para gerar o sinal modulado ruidoso (com a mesma SNR);
8. realiza-se L rodadas do algoritmo (executado como na figura 1.5);
9. compara-se os bits de saída com os bits de entrada;

Após verificar o funcionamento do algoritmo no Matlab, a unidade Goertzel foi testada. As simulações dos processos a serem implementados no FPGA foram realizadas no software *ISE Design Suite* da *Xilinx*, que utiliza linguagem de programação *VHDL*.

Para verificar o funcionamento da unidade Goertzel, a simulação no *ISE* foi realizada paralelamente à simulação no Matlab da seguinte forma: o Matlab gerava as amostras em ponto flutuante a partir de um sinal modulado corrompido por ruído, com frequência referente ao bit 1 igual a 10 kHz e frequência referente ao bit 0 igual a 5 kHz. A taxa de amostragem utilizada foi de 200 kHz e a taxa de bits foi definida como 2500 bits/s. Assim, o número de amostras por bit, ou N, é 80. Após gravar as amostras geradas num arquivo .txt, o Matlab executava o algoritmo. Após uma rodada de testes, definiu-se *threshold* = 1000. O ISE lia as amostras do arquivo .txt e simulava a unidade Goertzel. Os bits de saída eram armazenados em outro arquivo .txt para comparação. O diagrama abaixo ilustra o procedimento de simulação.

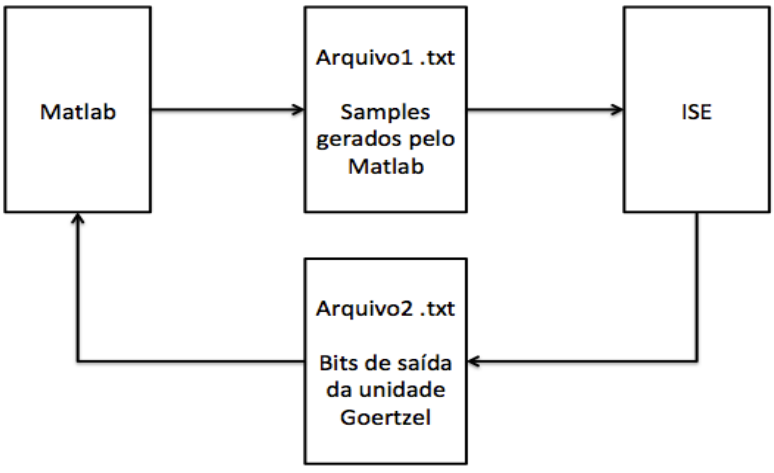

Figura 3.1 – procedimento de simulação da unidade Goertzel

Dessa forma pôde-se comparar os resultados dos bits gerados pelo Matlab com os resultados da demodulação do próprio Matlab, o que nos mostrava se a relação sinal-ruído fazia o algoritmo errar a identificação de bits, e com os resultados do ISE, que deveriam ser iguais aos do Matlab. As variáveis *q* do Matlab e da unidade Goertzel também foram comparadas.

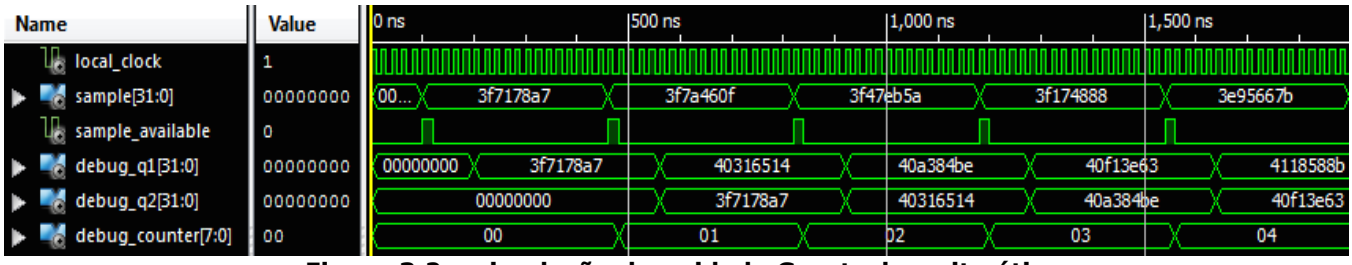

Figura 3.2 – simulação da unidade Goertzel – aritmética

A figura 2.11 mostra a simulação da unidade Goertzel. As amostras (*samples*) geradas em hexadecimal pelo Matlab são lidas do arquivo .txt a cada vez que a *flag sample_available* fica alta. O sinal *debug_counter* é um contador que ativa a unidade de cálculo do quadrado da magnitude quando atinge o valor N+1. Ele é incrementado cada vez que recebe uma nova amostra.

Observa-se que os sinais *debug_q1* e *debug_q2*, que representam *q1* e *q2*, são atualizados após 5 ciclos de clock – 4 ciclos para realizar a multiplicação e 1 ciclo para o flip-flop Q1 processar o resultado (figura 2.8). O flip-flop Q0 foi curto circuitado por causa da lógica utilizada (figura 1.5):

$$q0 = COEF \times q1 - q2 + sample;$$

$$q2 = q1;$$

$$q1 = q0;$$

Dessa forma a variável q1 é atualizada no mesmo ciclo que q0.

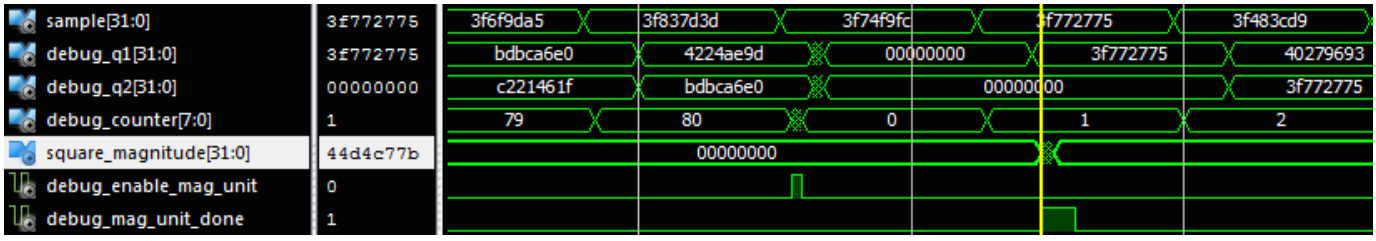

Figura 3.3 – simulação da unidade Goertzel – magnitude

A figura 2.12 é a continuação da simulação mostrada na figura 2.11. Observa-se que o sinal *debug_counter,* ao contar 81 *samples*, ativa a flag debug_*enable_mag_unit*, que ativa a unidade de cálculo da magnitude. A *flag debug_mag_unit_done* indica que a operação de cálculo da magnitude acabou e o sinal *square_magnitude* mostra o resultado do cálculo, 44d4c77b, que equivale à 1702,2338 – referente à um bit 1, pois 1702,2338 > threshold = 1000.

Observa-se que os sinais *q1* e *q2* são reiniciados após a ativação da unidade de magnitude. A amostra 3f74f9fc é perdida no processo, mas não afeta o resultado.

A figura 3.4 mostra um relatório gerado pelo *ISE* que fornece algumas informações em relação ao uso de recursos da placa.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	1,220	17,344	7%
Number of 4 input LUTs	4,360	17,344	25%
Number of occupied Slices	2,402	8,672	27%
Number of Slices containing only related logic	2,402	2,402	100%
Number of Slices containing unrelated logic	0	2,402	0%
Total Number of 4 input LUTs	4,363	17,344	25%
Number used as logic	4,356		
Number used as a route-thru	3		
Number used as Shift registers	4		
Number of bonded IOBs	56	190	29%
Number of BUFGMUXs	2	24	8%
Number of DCMs	1	8	12%
Number of MULT18X18SIOs	20	28	71%
Average Fanout of Non-Clock Nets	3.46		

Figura 3.4 – relatório da utilização de recursos da placa

4. Montagem e Resultados

4.1. Montagem Experimental

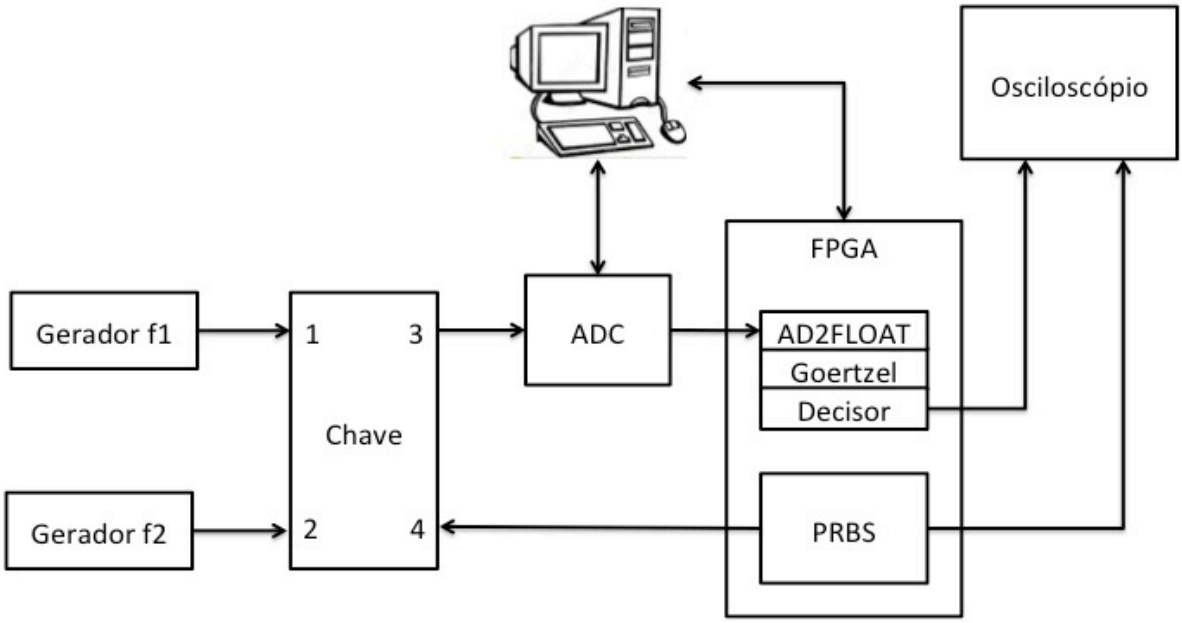

Figura 4.1 – diagrama da montagem experimental

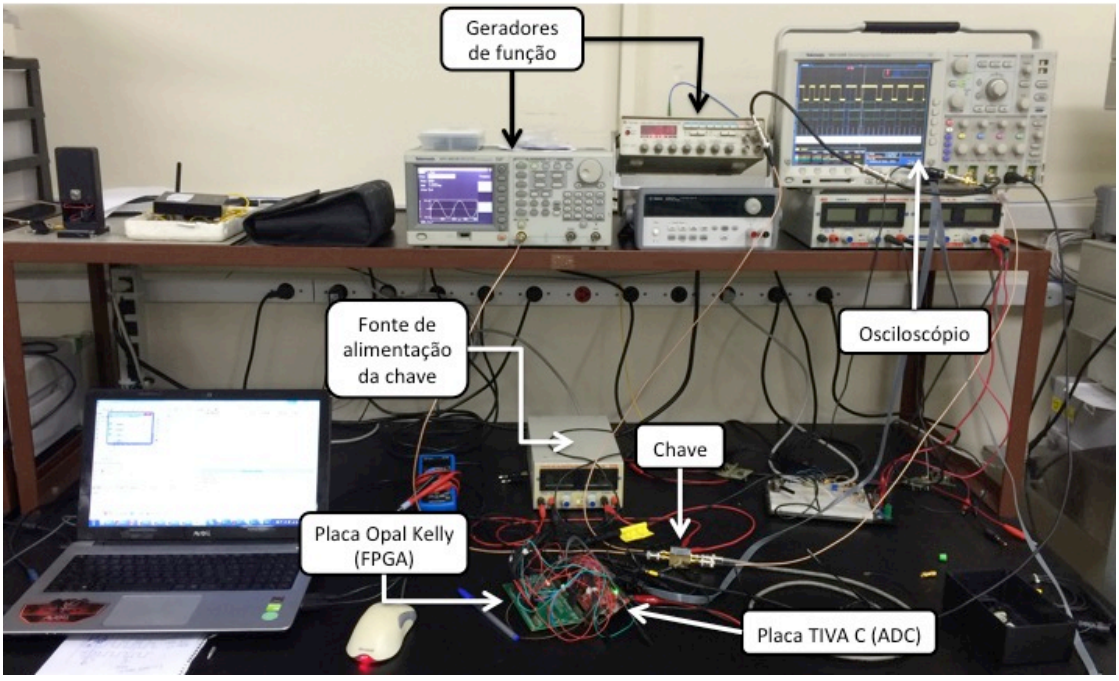

Figura 4.2 – montagem experimental

A placa utilizada para a implementação do algoritmo Goertzel foi uma Opal Kelly XEM 3005 com FPGA Spartan 3E. O FPGA foi programado de forma a gerar uma sequência pseudoaleatória de bits, que correspondem à informação a ser transmitida. Essa sequência é modulada por sinais gerados por dois geradores de função, cada um ajustado para uma frequência.

O *pseudo random bit generator (PRBS)* gera os bits utilizando dez flip-flops tipo D, inicializados de forma arbitrária. As saídas do décimo e do primeiro flip-flops foram combinadas através de uma lógica "ou exclusivo" (XOR). Dessa forma, uma sequência pseudoaleatória de bits é gerada.

A chave foi utilizada para enviar o sinal modulado ao ADC. Os geradores de função enviam sinais para as portas 1 e 2 (RF1 e RF2). A porta 4 (TTL) recebe os bits gerados pelo FPGA. A saída na porta 3 (RFIN) pode ser o sinal da porta 1 ou 2, dependendo do estado do bit na porta 4. Ou seja, a saída na porta 3 é o sinal modulado.

Figura 4.3 – chave

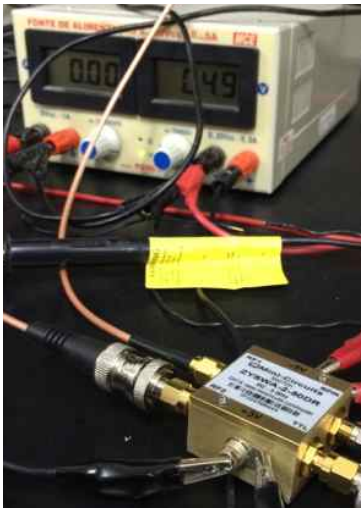

Figura 4.4 – chave com fonte de alimentação

Para o ADC foi utilizada uma placa TM4C1294NCPDT. O ADC da placa é de 12 bits e foi programado para gerar amostras a uma frequência de 500 kHz. Cada bit é enviado a um pino do ADC, e os pinos são conectados às entradas correspondentes do FPGA. A placa foi programada para gerar uma *flag* indicando que uma conversão foi realizada. Essa *flag* também é enviada ao FPGA através de um pino.

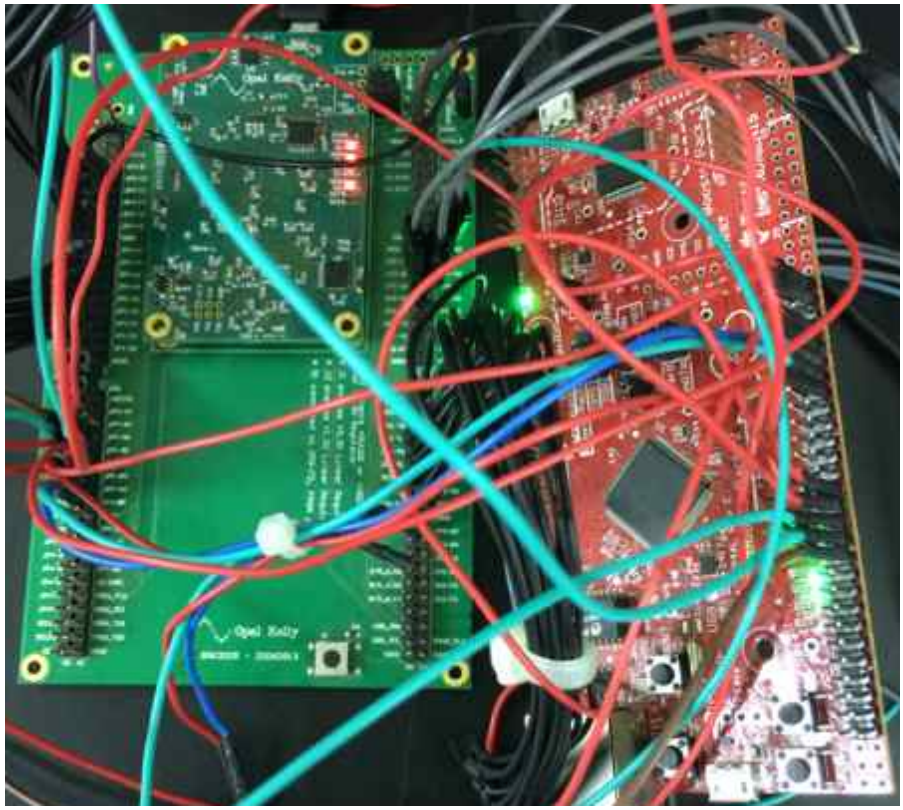

Figura 4.5 – placas Opal Kelly (FPGA), à esquerda, e TIVA C (ADC), à direita

A unidade AD2FLOAT do FPGA recebe os 12 bits do ADC e os converte em ponto flutuante (32 bits). A amostra é então enviada à unidade Goertzel. Quando a N-ésima amostra é recebida, a unidade de magnitude é ativada. Ao final do cálculo, o valor do quadrado da magnitude é enviado ao decisor, que compara o valor com o threshold e envia finalmente o bit demodulado ao osciloscópio. Dessa forma, é possível comparar os bits gerados pelo PRBS e os bits demodulados.

O FPGA e a placa do ADC são alimentados e programados pelo computador através de portas USB. O ADC funciona continuamente enquanto recebe alimentação. Foi desenvolvido um programa em *Python* para controlar o FPGA.

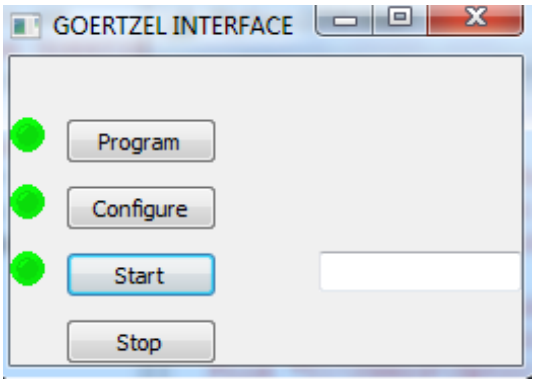

Figura 4.6 – interface de controle do FPGA em Python

O programa também é responsável por salvar os bits gerados pelo PRBS e os bits demodulados em arquivos .txt para que se possa verificar o funcionamento do projeto. O *Python* salva os dados nos arquivos .txt em uma velocidade menor que a geração de bits pelo PRBS e que a demodulação, logo alguns dados não são registrados. Isso não afeta a verificação do funcionamento, pois é possível comparar uma grande quantidade de dados.

4.2. Resultados

Antes de realizar a demodulação, testes foram realizados buscando traçar a banda de aceitação do filtro, ou o quadrado da magnitude, para diferentes frequências *target*. Foi utilizada uma taxa de transmissão de 2500 bits/s e uma taxa de amostragem de 500 kHz, logo o número de amostras por bit, ou N, é igual a 200.

O primeiro teste foi realizado para $f_T = 5$ kHz (COEFF = 1.9961). Foi realizada uma varredura das frequências de 1 kHz a 10 kHz com passo de 10 Hz – equivalente a 900 pontos. Para cada frequência 1000 valores de magnitude foram calculados. O Matlab foi programado para calcular a média das magnitudes para cada frequência e gerar o gráfico do resultado.

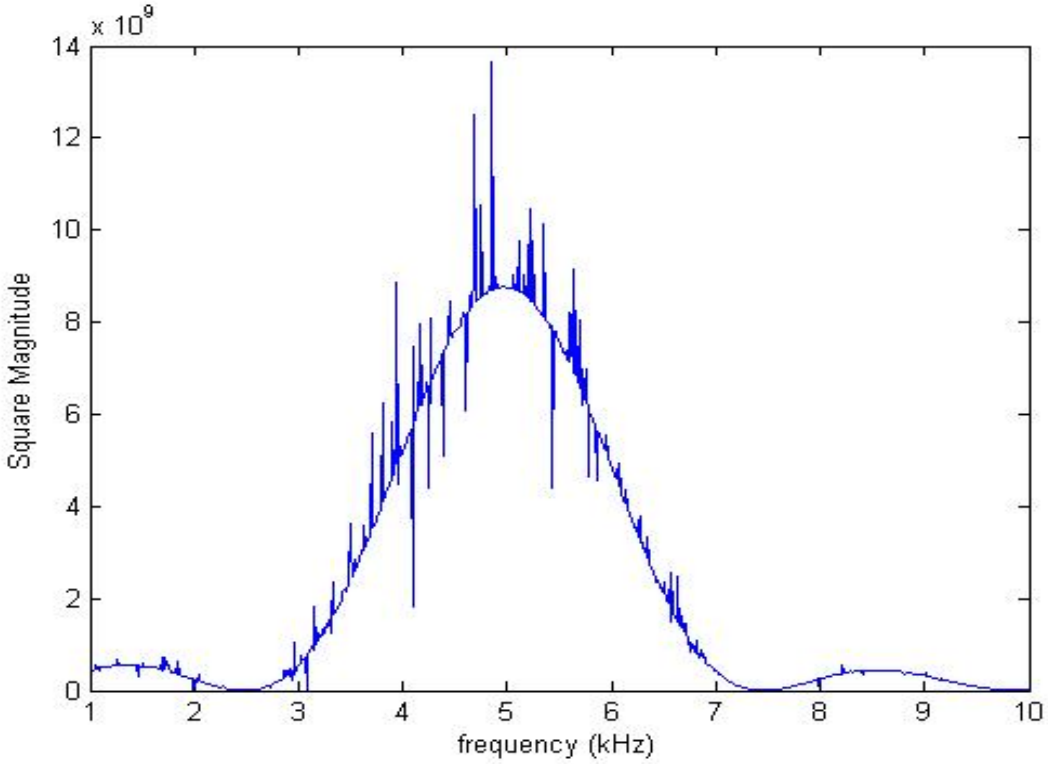

Figura 4.7 - quadrado da magnitude × frequência - COEF = 1.9961

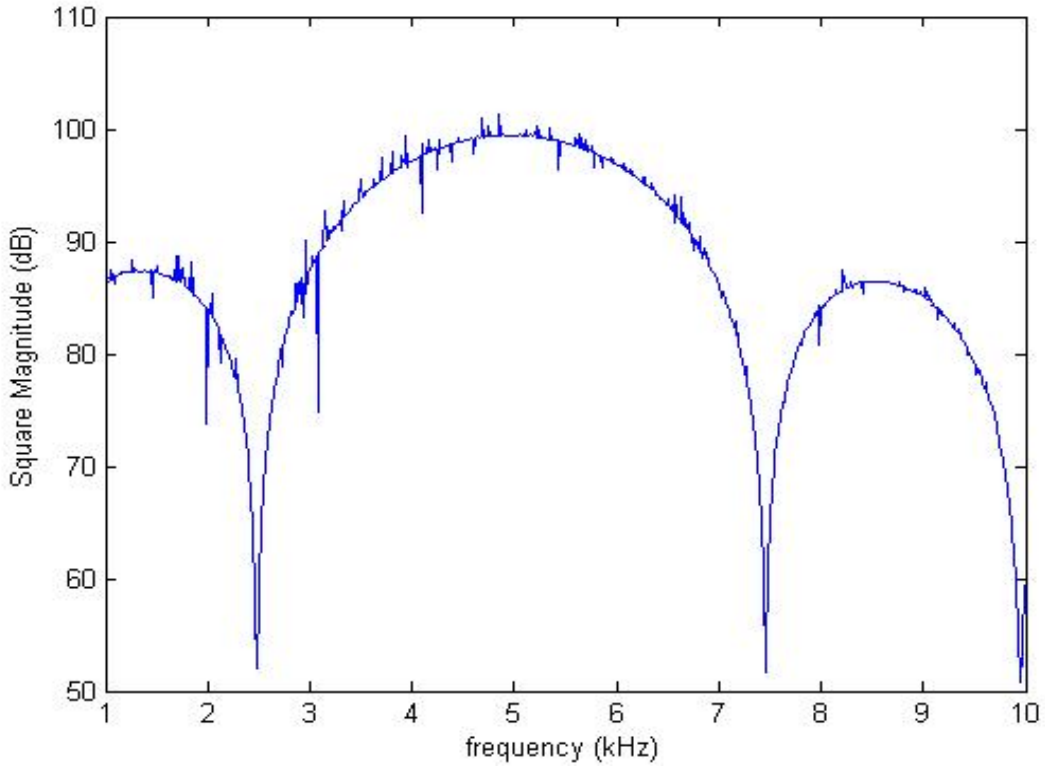

Figura 4.8 - quadrado da magnitude (dB) × frequência - COEF = 1.9961

As figuras acima são coerentes com a figura 1.5, mostrando que o algoritmo funciona como um filtro em torno da frequência *target*, com lobos laterais. De acordo com a equação 1.16, os primeiros mínimos deveriam aparecer em $f_T \pm f_s/N$. Temos que f_s/N = 2500 Hz, logo a figura 4.7 é coerente com a equação 1.16.

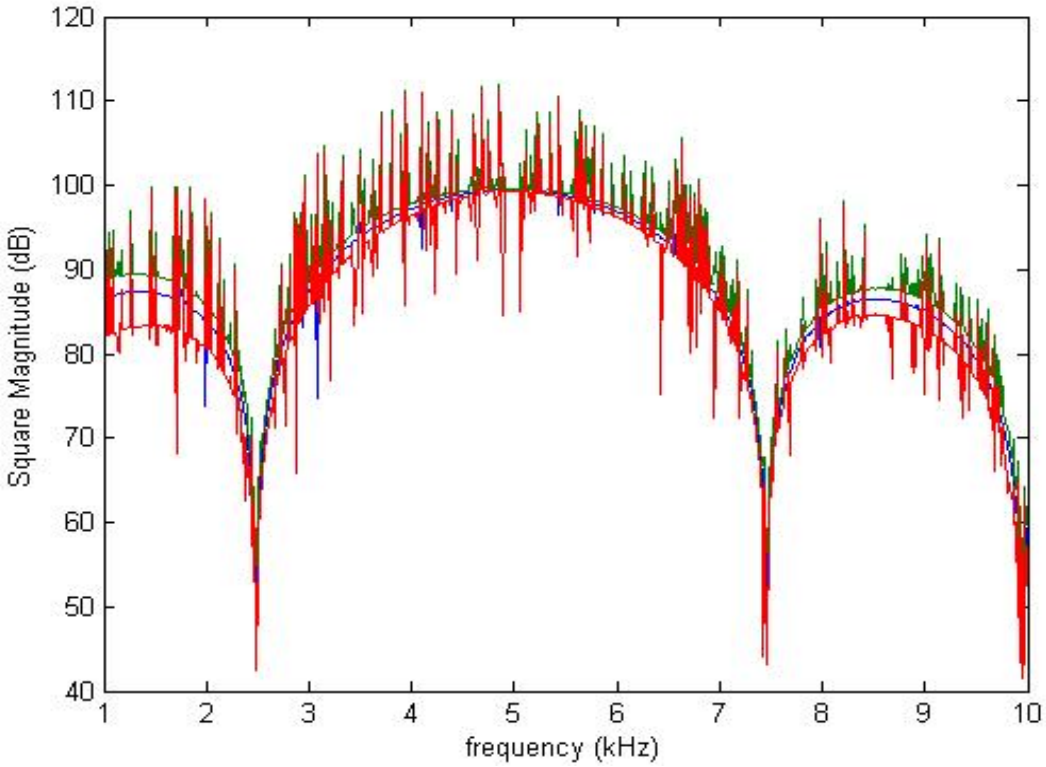

Figura 4.9 - quadrado da magnitude (dB) × frequência - COEF = 1.9961 com desvio padrão

A figura 4.9 mostra como os valores de magnitude variam em torno da média. Apesar de o desvio apresentar muito ruído, é possível observar que ele aumenta conforme a frequência se afasta da frequência *target*.

Os testes seguintes foram realizados para as frequências de 100 kHz (COEF = 0.618), varrendo as frequências de 90 kHz a 110 kHz, 200 kHz (COEF = -1.618), varrendo as frequências de 190 kHz a 210 kHz, e 250 kHz (COEF = -2), varrendo as frequências de 240 kHz a 260 kHz. O passo usado foi de 200 Hz – 100 pontos para cada gráfico.

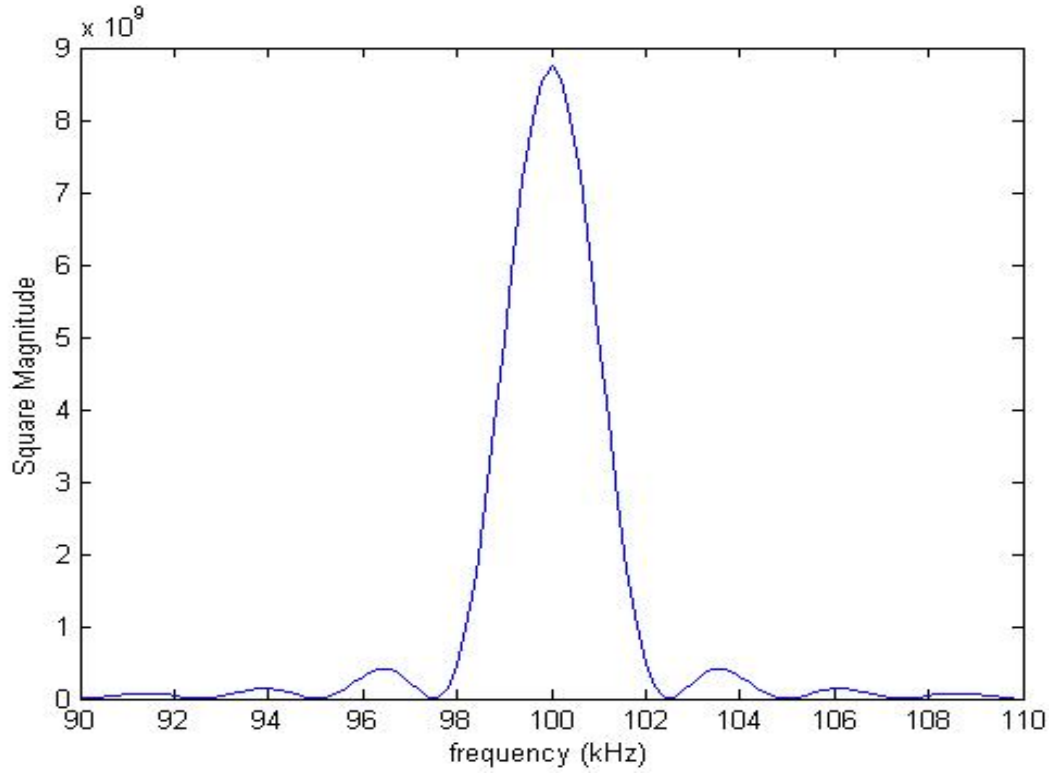

Figura 4.10 - quadrado da magnitude × frequência - COEF = 0.618

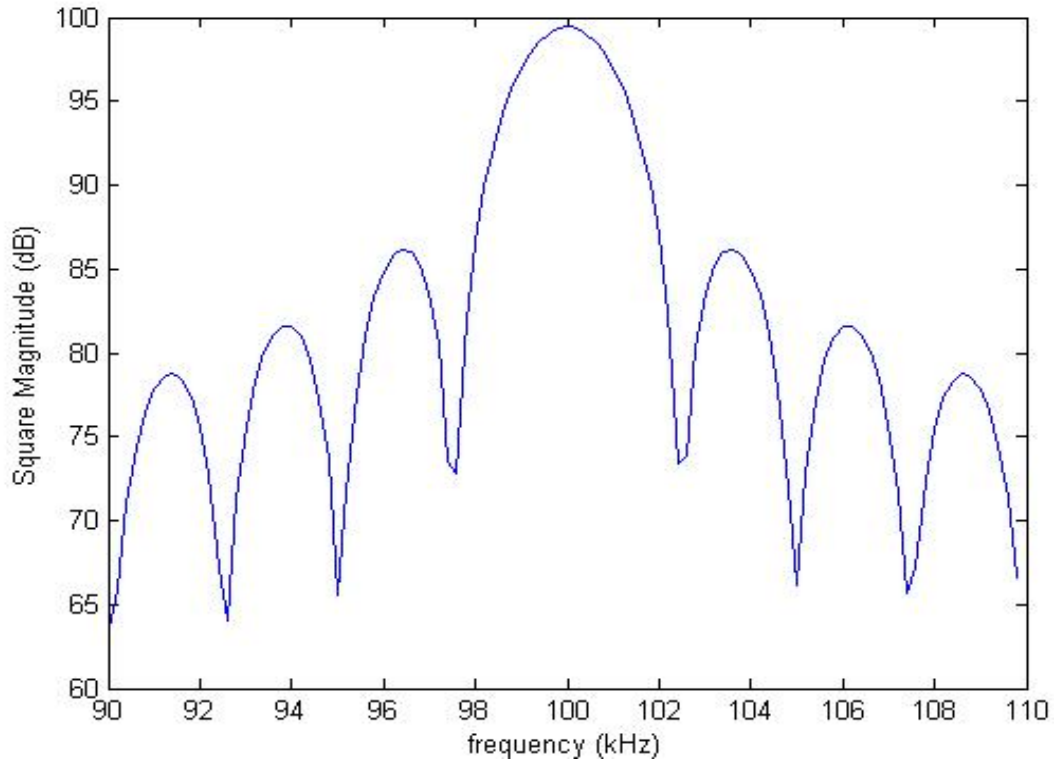

Figura 4.11 - quadrado da magnitude (dB) × frequência - COEF = 0.618

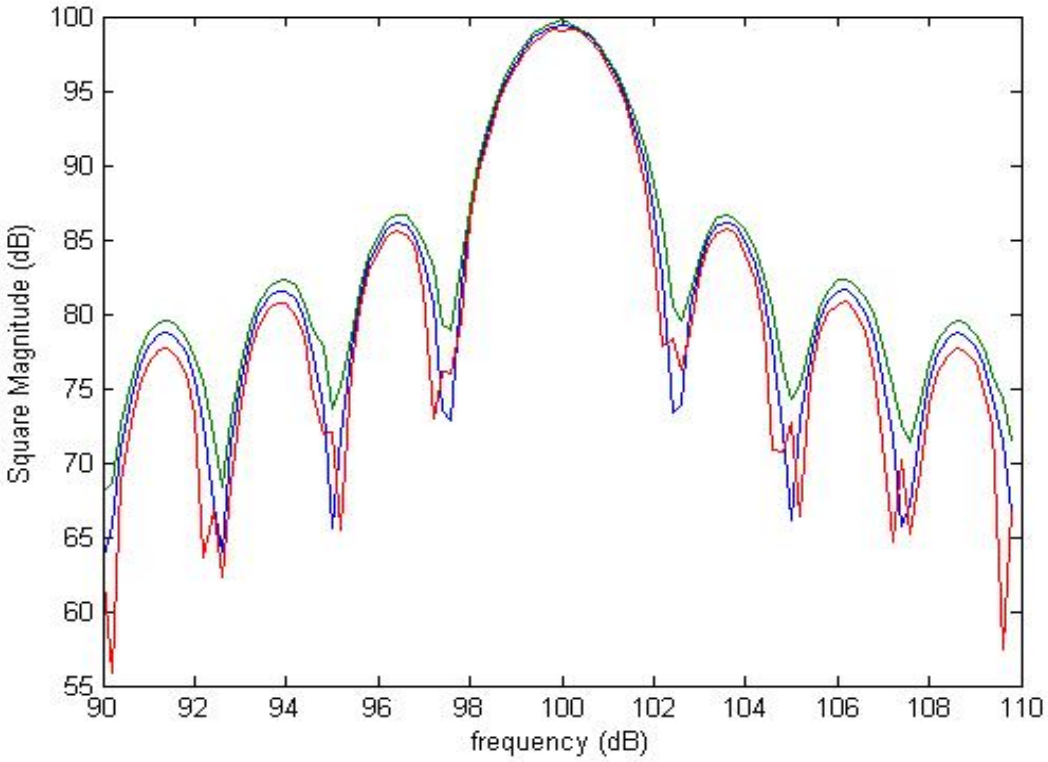

Figura 4.12 - quadrado da magnitude (dB) × frequência - COEF = 0.618 com desvio padrão

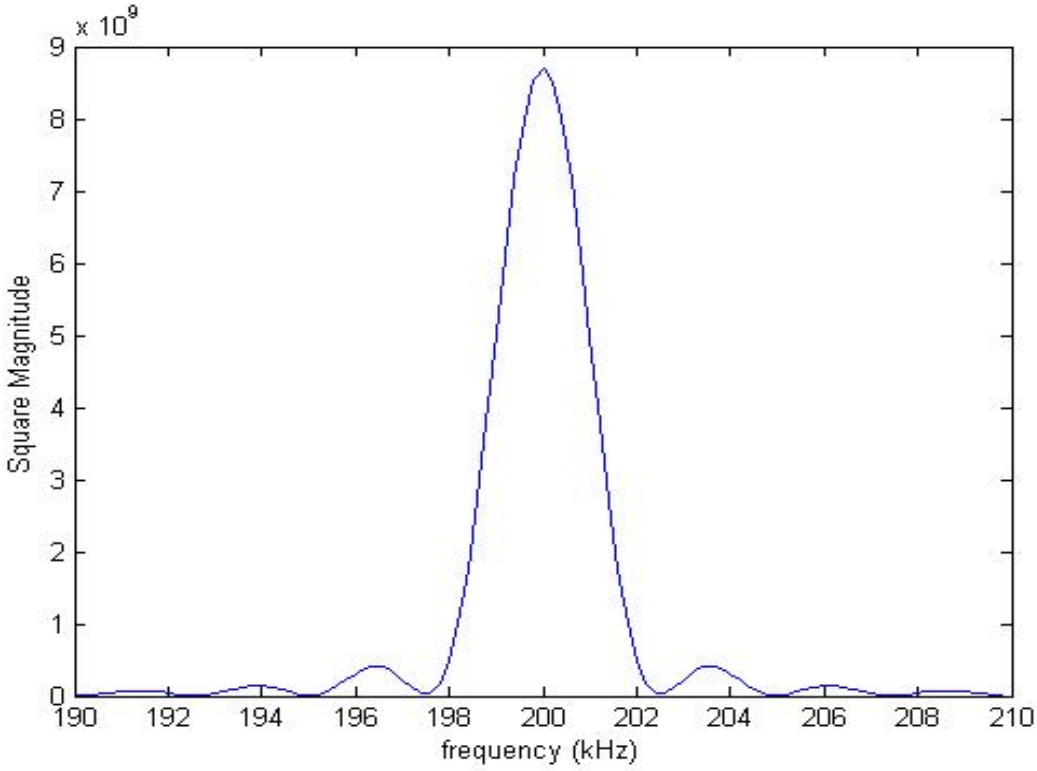

Figura 4.13 - quadrado da magnitude × frequência – COEF = −1.618

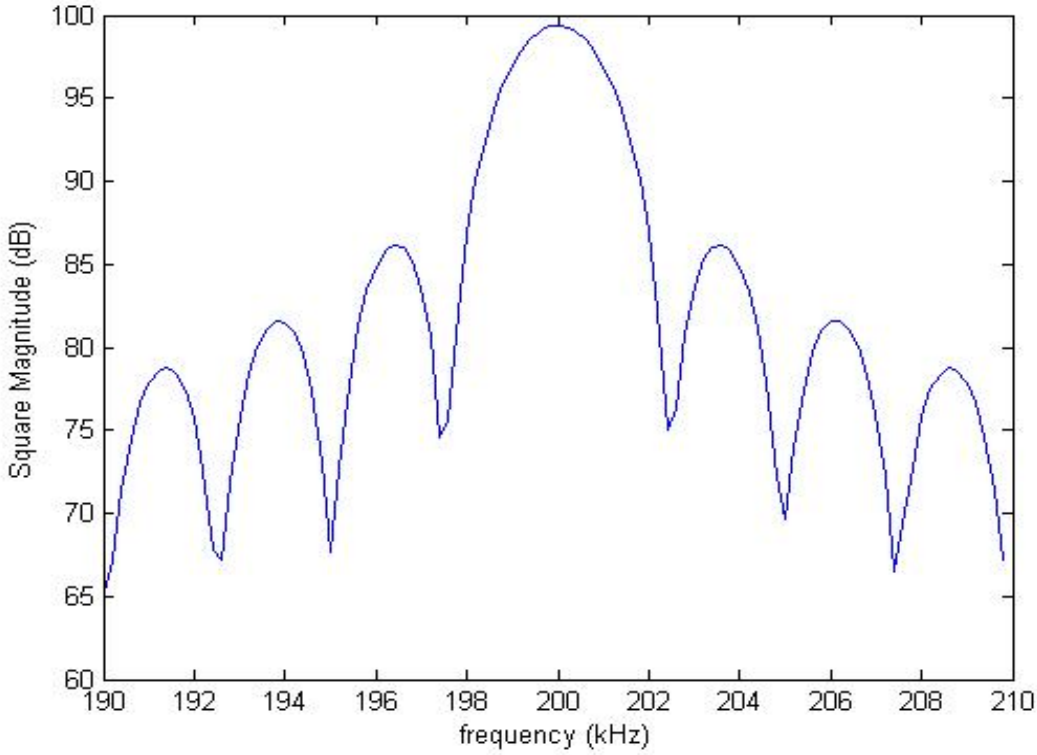

Figura 4.14 - quadrado da magnitude (dB) × frequência – COEF = −1.618

Os gráficos das frequências de 100 kHz e 200 kHz apresentam menos ruído que os gráficos da frequência de 5 kHz pois sua resolução é menor. Observa-se que os gráficos são coerentes com os resultados encontrados nos testes para a frequência de 5 kHz.

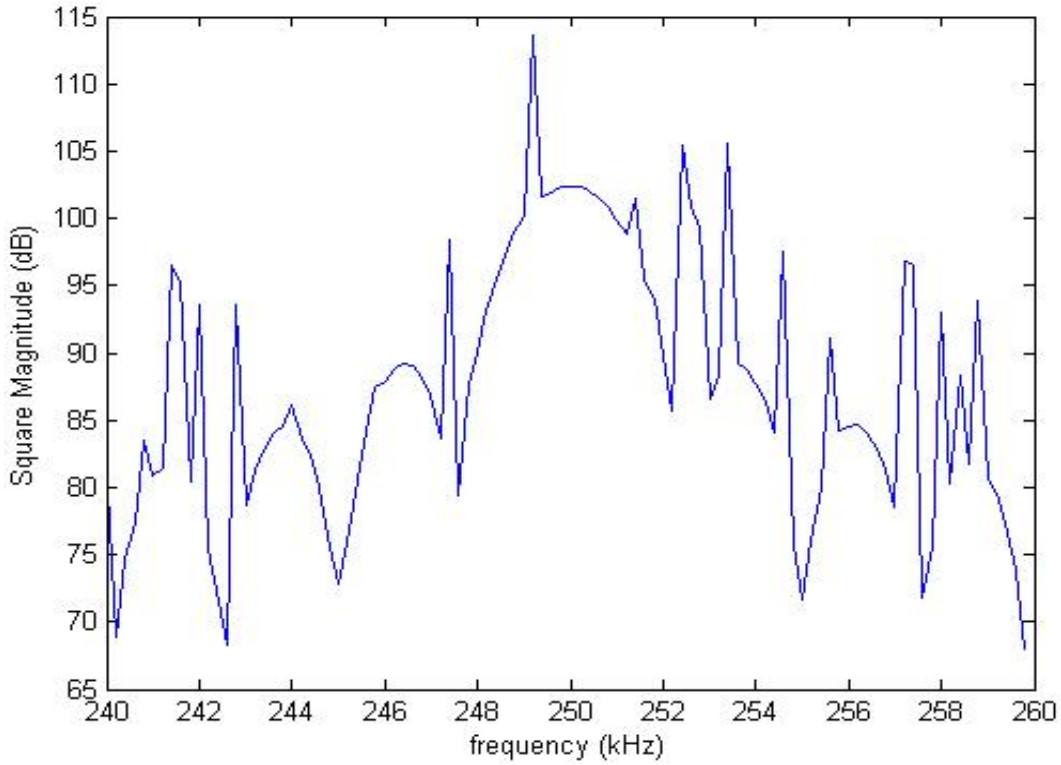

Figura 4.15 - quadrado da magnitude (dB) × frequência - COEF = −2

A figura 4.10 mostra como estar atuando no limite da frequência de Nyquist impossibilita a aplicação do algoritmo.

Para os testes experimentais foram utilizadas as frequências de 5 kHz, associada ao bit 1, e 10 kHz, associada ao bit zero. A frequência de 5 kHz foi escolhida como *target* (COEF = 1,9961). Os geradores de função tiveram de ser ajustados de forma que as senóides geradas tivessem valor médio 1 e 1 volt pico a pico, pois o ADC não lê valores negativos. O limiar de decisão (*threshold*) foi definido a partir da figura 4.7 em 8×10^9.

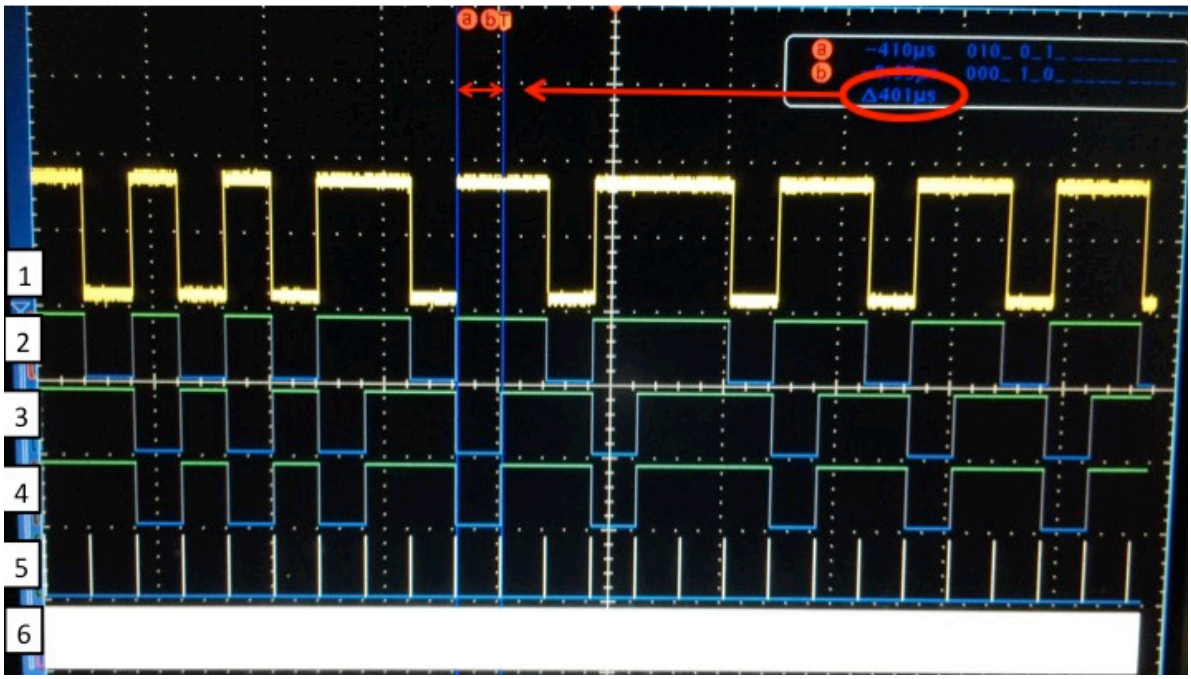

Figura 4.16 – resultado da demodulação no osciloscópio

A figura 4.8 mostra os bits gerados pelo PRBS e os bits demodulados. O sinal 1 é o sinal analógico dos bits gerados pelo PRBS, enquanto que o sinal 2 é o sinal digital. O sinal 3 é uma versão atrasada em um período bit do sinal 2. O sinal 4 é o sinal demodulado, ou os bits que são enviados pelo demodulador. O sinal 5 mostra as *flags* que indicam que a demodulação foi realizada e que um bit está disponível no decisor. O sinal 6 mostra as *flags* que indicam que o ADC gerou uma amostra e que ela está disponível. Não é possível visualizar o sinal 6 com precisão na figura pois a *flag* fica alta 200 vezes entre cada *flag* do sinal 5.

Na figura 4.8 dois cursores mostram o tempo de 401 us entre o bit gerado e o bit demodulado. Esse atraso corresponde ao tempo de bit (o inverso da taxa de bits de 2500 bits/s), com uma diferença de 1 us devido à imprecisão no posicionamento dos cursores. Esse atraso ocorre, pois o algoritmo Goertzel acumula todas as 200 amostras do sinal modulado referentes ao bit antes de decidir qual o valor do bit. Ou seja, o atraso de um período de bit do bit demodulado em relação ao bit gerado é intrínseco ao funcionamento do algoritmo Goertzel.

A figura abaixo mostra o resultado da modulação com o sinal modulado no lugar dos bits gerados pelo PRBS.

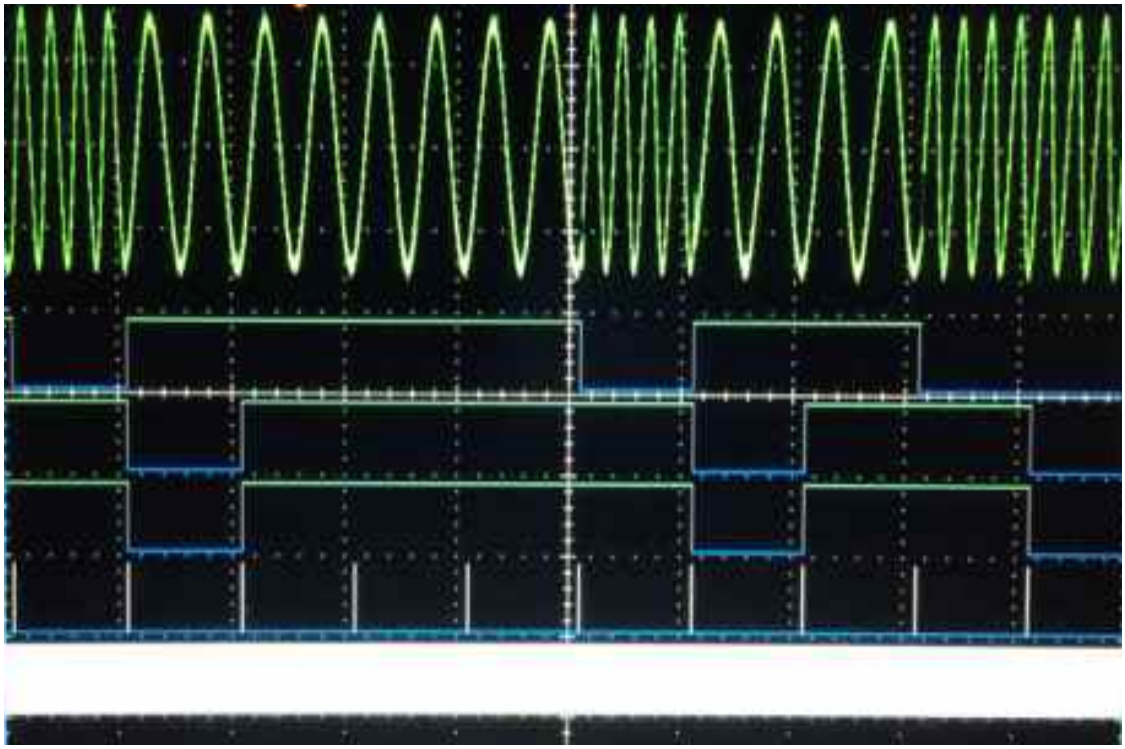

Figura 4.17 – resultado da demodulação com sinal modulado

Foi desenvolvido um programa em Matlab que compara os dados do arquivo .txt que contém os bits gerados com o arquivo .txt que contém os bits demodulados. O teste foi realizado em 733051 bits e nenhum erro foi detectado. A figura abaixo mostra o valor dos primeiros 50 bits gerados (azul) e os 50 bits demodulados (vermelho).

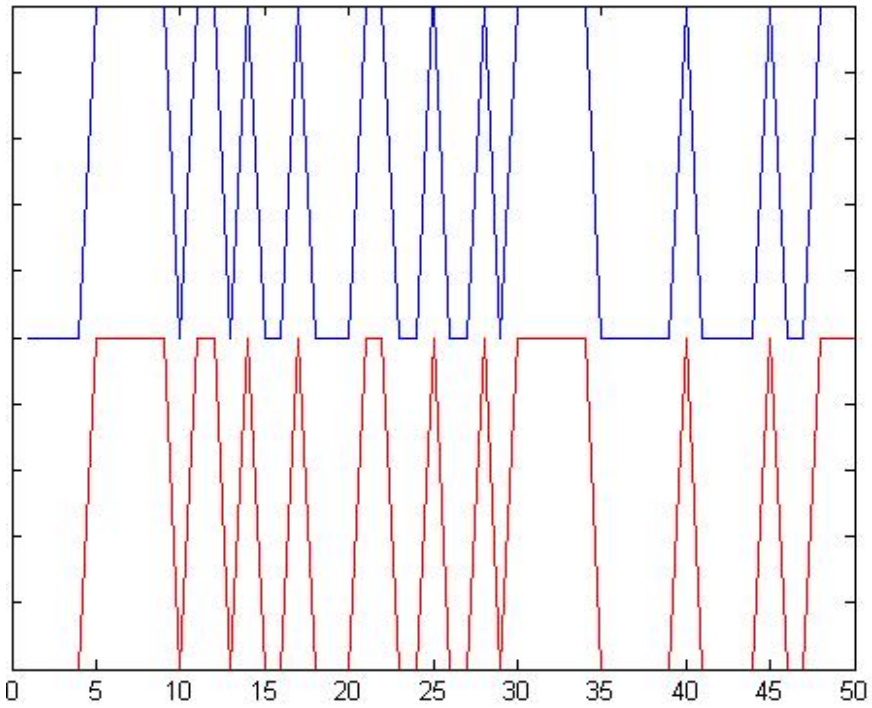

Figura 4.18 – comparação do Matlab entre bits gerados e bits demodulados

Variando-se a frequência do sinal modulado foi possível observar que a demodulação funciona sem erros entre 4.8 kHz e 5.2 kHz para o limiar escolhido. A medida que a frequência se afasta desse intervalo, o algoritmo erra cada vez mais a identificação de bits 1. Para frequências menores 4.3 kHz e maiores que 5.7 kHz a demodulação não identifica nenhum bit 1. Existe uma faixa de 500 Hz em torno da banda de 4.8 kHz a 5.2 kHz em que existe um desvio em relação à média no valor da magnitude que faz com que o decisor acerte alguns resultados.

5. Conclusão

As unidades de operação de soma e multiplicação em ponto flutuante, desenvolvidas ao longo do projeto, permitem que o FPGA seja capaz de utilizar o algoritmo Goertzel para demodular sinais BFSK com uma taxa de transmissão de 2500 bits/s. As simulações em software do circuito digital indicam que o sistema permite uma taxa de transmissão de até 30 kbits/s, utilizando um clock de 50 MHz e um ADC de 3 MHz. A implementação do algoritmo Goertzel em microcontroladores TM4C1294NCPDT e TMS320F28069 permitiu uma demodulação de sinais BFSK para uma taxa de transmissão de no máximo 1200 bits/s. Ou seja, a implementação permite uma performance melhor do que a dos microcontroladores testados em relação à taxa de transmissão.

Apesar do bom desempenho em relação à taxa de transmissão, as operações em ponto flutuante utilizam muito recurso da placa. Dessa forma, não foi possível implementar uma demodulação simultânea de duas frequências na placa XEM 3005, pois para uma frequência a placa utiliza 71% dos multiplicadores de hardware.

Conclui-se que a implementação desenvolvida é veloz e precisa, sendo uma alternativa eficaz para demodulação BFSK. Porém, para uma aplicação mais robusta, onde seja necessário identificar mais de uma frequência simultaneamente, é necessária a utilização de uma placa com mais recursos, o que implica num projeto mais caro.

6. Referências

[1] Petrobras [Online]. http://www.petrobras.com.br/ . [Acesso em 20/05/2015]

[2] S. Haykin, Digital Communication Systems, 1ª ed.

[3] A. V. Oppenheim e R. W. Schafer, Discrete-Time Signal Processing, 3ª ed.

[4] Xilinx [Online]. http://www.xilinx.com/. [Acesso em 20/05/2015]

[5] G. C. d. Amaral, FPGA Applications on Photon Detection Systems, Rio de Janeiro: MsC Thesis, DEE, PUC-Rio, 2014.

[6] Embedded [Online]. http://www.embedded.com/design/real-world-applications/4401754/Single-tone-detection-with-the-Goertzel-algorithm. [Acesso em 25/03/2015]

[7] IEEE 754 [Online]. http://www.h-schmidt.net/FloatConverter/IEEE754.html. [Acesso em 15/04/2015]