CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

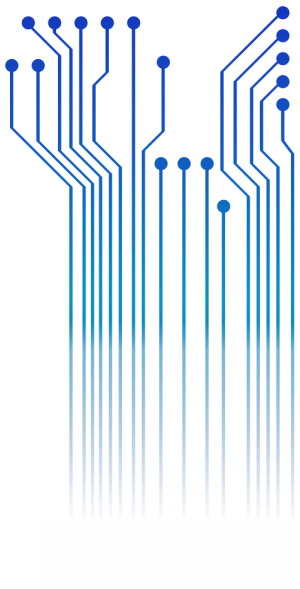
Universidade Federal
de Campina Grande

HELON AIRÃ SOARES E FERREIRA

Centro de Engenharia
Elétrica e Informática

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Departamento de
Engenharia Elétrica



Campina Grande, Paraíba
2021

HELON AIRÃ SOARES E FERREIRA

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Relatório de Estágio Supervisionado submetido à Coordenação do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Orientador:
Professor Edmar Candeia Gurjão, D. Sc.

Campina Grande, Paraíba
2021

HELON AIRÃ SOARES E FERREIRA

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Relatório de Estágio Supervisionado submetido à Coordenação do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento de Sinais

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Edmar Candeia Gurjão, D. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

AGRADECIMENTOS

Agradeço a Deus, em primeiro lugar, pela minha vida e pelo dom da perseverança, que me permitiu concluir este trabalho.

Agradeço aos meus pais, Genivaldo Ferreira e Paula Soares, que sempre me deram todas as oportunidades e incentivos desse mundo, para estudar e buscar sempre ir mais longe, e me fazer acreditar sempre no meu potencial. De forma igual aos meus dois irmãos mais novos, cada um com seu jeito, mas sempre se fizeram presentes em todas as etapas de minha vida.

Agradeço a minha namorada, Milka, que sempre está ao meu lado, me dando forças, conselhos e suporte. Sempre presente comigo nos momentos de dificuldades, me ajudando para que conseguisse retomar a jornada.

Agradeço ao professor Edmar, pois sempre que bati na porta da sua sala se mostrou solicito tirando dúvidas, dando pequenos conselhos e também a oportunidade deste estágio.

Agradeço também ao professor Angilberto, que tive a oportunidade de conhecer durante o estágio, pois me ajudou e tirou minhas dúvidas incontáveis vezes.

Gostaria de agradecer também por todos os colegas que estiveram presentes na minha jornada.

Por fim agradecer aos funcionários da coordenação, especialmente Adail e Tchai, que sempre estão a postos para auxiliar, independente qual seja o problema, tanto acadêmico quanto da vida pessoal.

"A determinação das pessoas é como a água, ninguém pode parar. Ela rodeia montanhas, cai em cascatas, mas no fim chega onde tem que chegar"

Chef Andoni Aduriz.

RESUMO

Neste trabalho são apresentadas as atividades do aluno Helon Airã Soares e Ferreira, realizadas durante o Estágio Supervisionado no Laboratório de Metrologia (LabMet), sob orientação do Professor Edmar Candeia Gurjão. O estágio é um trabalho integrado ao projeto BINGO (*Baryon Acoustic Oscillations from Integrated Neutral Gas Observation*) e dividido em etapas, as quais constituem: estudo da transformada rápida de *Fourier* (FFT), pesquisa de formas para implementar um receptor de sinais utilizando uma placa FPGA (*Field Programmable Gate Array*) e por fim realizar os testes e a validação. Como ferramentas metodológicas utilizou-se os *softwares Quartus II 13.0 Web Edition*®, *Microsoft Visual Studio Community Edition 2019*®, e ferramentas como a placa FPGA (*Field Programmable Gate Array*) Saxo.

Palavras-chave: BINGO, FFT, receptor de sinais, FPGA.

ABSTRACT

This report presents the activities of the student Helon Airã Soares e Ferreira, conducted during the Supervised Intership at the Metrology Laboratory (LabMet), under supervision of professor Edmar Candeia Gurjão. The intership is a integrated work with the Project BINGO (*Baryon Acoustic Oscillations from Integrated Neutral Gas Observation)* and divided in steps that consisted: study the *Fast Fourier Transform* (FFT), search of ways to implement a signal receiver using an FPGA (*Field Programmable Gate Array*) and at the end make tests and validation. As methodological tools were used the softwares *Quartus II 13.0 Web Edition®*, *Microsoft Visual Studio Community Edition 2019®*, and tools such an FPGA (*Field Programmable Gate Array*) Saxo.

Keywords: BINGO, FFT, signal receiver, FPGA.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

LabMet	Laboratório de Metrologia
BINGO	*Baryon Acoustic Oscillations from Integrated Neutral Gas Observation* – Oscilações Acusticas de Bárions Integrado a Observação de Gás Neutro
FFT	Transformada Rápida de Fourier
FPGA	*Field Programmable Gate Array* – Arranjo de Portas Programáveis em Campo
FRB	*Fast Radio Bursts* – Rajadas Rápidas de Rádio
USB	*Universal Serial Bus* – Porta Serial Universal
API	*Application Programming Interface* – Interface de Progamação de Aplicações
SoC	*System-on-Chip* – Sistema em um *Chip*
FIFO	*First In, First Out* – Primeiro a entrar, Primeiro a Sair

LISTA DE SÍMBOLOS

MHz	Megahertz
MB/s	Megabyte por segundo
Mbit/s	Megabit por segundo

SUMÁRIO

1 INTRODUÇÃO

Estágio supervisionado, o qual as atividades são descritas neste relatório, teve duração de 180 horas e foi realizado no Laboratório de Metrologia (LabMet), no período de 17 de março de 2021 até 14 de maio de 2021, sob a orientação do professor Edmar Candeia Gurjão.

A prática do estágio é uma atividade obrigatória para a obtenção do diploma de Engenheiro Eletricista, mas também é de suma importância para o aluno ter a oportunidade de unir os conhecimentos adquiridos, durante o curso, com a atividade prática.

A atividade realizada teve como objetivo implementar um receptor de sinais, que será utilizado no desenvolvimento de um espectrômetro específico do projeto BINGO (*Baryon Acoustic Oscillations from Integrated Neutral Gas Observation*). O BINGO é um projeto de colaboração internacional, que tem como foco estudar a parte escura do universo.

1.1 LOCAL DO ESTÁGIO

O estágio foi realizado nas dependências do LabMet, do Departamento de Engenharia Elétrica (DEE), localizado na Universidade Federal de Campina Grande (UFCG), no endereço Rua Aprigio Veloso 822, 58429-900, Campina Grande. O laboratório tem como funções a criação, desenvolvimento e aperfeiçoamento de ensaio e calibração de sistemas.

Figura 1 - Entrada do LabMet

Fonte: Autor

Para o cumprimento do trabalho foi disponibilizada a sala 18 do laboratório mostrado na figura 2, onde estavam disponíveis os equipamentos necessários, como computador e a placa FPGA Saxo.

Figura 2 - Sala 18 LabMet

Fonte: Autor

1.2 PROJETO BINGO

O BINGO (*Baryon Acoustic Oscillations from Integrated Neutral Gas Observation*) é um projeto de colaboração internacional entre Brasil, Inglaterra, China, França, Itália, Espanha, Alemanha, África do Sul e Suíça. As instituições de ensino e pesquisa brasileiras envolvidas no projeto são a Universidade de São Paulo (USP), Instituto Nacional de Pesquisas Espaciais (INPE) e a Universidade Federal de Campina Grande (UFCG).

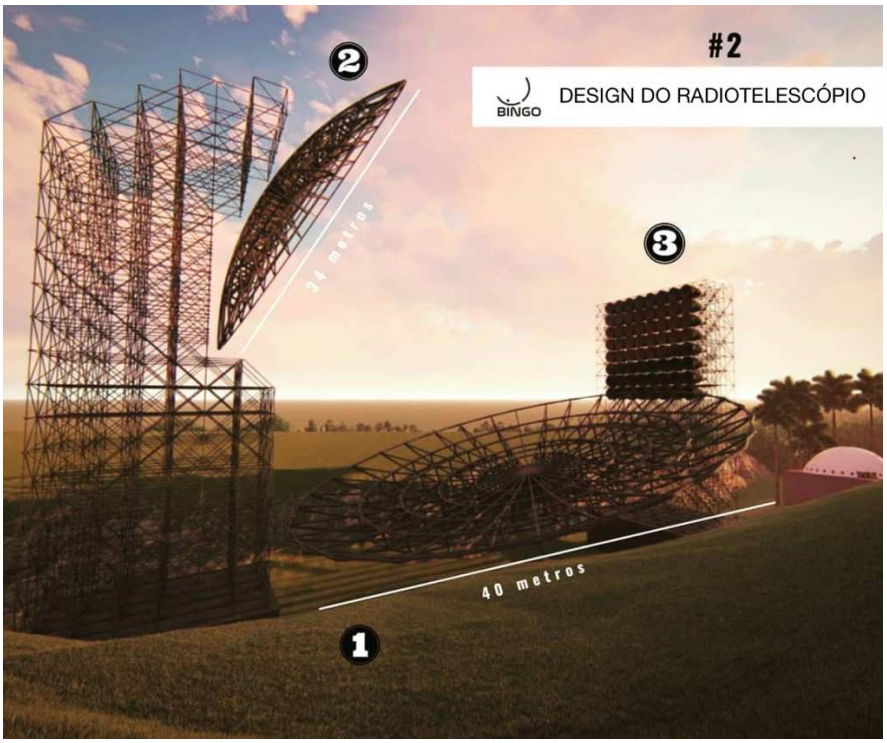
O Projeto foi idealizado pela *University of Manchester* e por meio de uma parceria com a USP, iniciou-se a fase de estudo e desenvolvimento no ano de 2016, com o apoio financeiro da FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo).

Primeiro e único radiotelescópio que será dedicado a detecção e estudo das Oscilações Acústicas de Bárions por meio de rádio frequência. A detecção será feita na banda de rádio na faixa de 980 a 1260MHz.

Embora o projeto tenha seu principal foco na detecção das oscilações de bárions, em virtude da sua estrutura, da faixa de frequência analisada e posição imóvel, é possível estabelecer metas secundárias de estudo como a detecção de FRB (*Fast Radio Bursts*), sendo pioneiro no hemisfério sul, e emissão de pulsares.

O radiotelescópio será construído na Serra do Urubu, localizado na cidade de Piancó, Paraíba. Ele é constituído de dois espelhos refletores, ou parabólicas, cada uma com aproximadamente 40 metros de diâmetro e um conjunto de 50 cornetas acopladas. Na figura 3, podemos observar como é o *design* do projeto de sua estrutura física. Uma de suas parabólicas receberá incidência de ondas eletromagnéticas, que por sua vez refletirá para a segunda parabólica e por fim enviará os sinais para o conjunto de cornetas, estas por sua vez farão o envio desses sinais até os receptores.

Figura 3 - Design do Rádio Telescópio. (1) Parabólica Refletora primária; (2) Parabólica Refletora Secundária; (3) Conjunto de cornetas;



Fonte: (BINGO TELESCOPE, 2019) e adaptada pelo autor

Na UFCG está instalada o modelo de uma das 50 cornetas que farão parte do radiotelescópio, como pode ser visualizado na figura 4.

Figura 4 - Corneta do BINGO instalada na UFCG

Fonte: Autor

1.3 PLANO DE ESTÁGIO

As atividades propostas para serem realizadas, durante o estágio foram:

- Auxílio na implementação do receptor de sinais Cosmológicos;
- Desenvolvimento de ferramentas de análise de sinais;
- Implementação na FPGA da análise de sinais;
- Teste e validação do sistema;

2 ATIVIDADES REALIZADAS

O objetivo principal do nosso trabalho era de conseguir fazer a recepção de um sinal qualquer e devolver como resultado a FFT desse sinal, pois uma das contribuições da UFCG ao projeto BINGO é auxiliar no desenvolvimento de um espectrômetro específico do telescópio, ou seja, fazer a digitalização de todos os sinais que forem captados.

Para este objetivo, foi escolhido utilizar uma FPGA (*Field Programmable Gate Array*), pois é um dispositivo lógico programável capaz de implementar circuitos digitais. Outro aspecto é que estamos tratando de um *hardware* que realiza instruções a cada ciclo de *clock*, ou seja, ele pode fazer os cálculos, em nosso caso, de multiplicação e adição em paralelo e enviar seus resultados em um único ciclo de *clock*, algo que seria uma tarefa demorada e com grau de complexidade para algum *software*.

Então, tendo o dispositivo a ser utilizado determinado, o primeiro passo era implementar um código em *verilog*, linguagem de descrição de *hardware*, capaz de calcular a FFT e em trabalhos conjuntos com outro colega do laboratório, foi possível a elaboração e obtenção deste código.

Com o ponto de partida, uma vez que observamos que é possível fazer a FFT, o foco passou a ser de que o processo matemático fosse inteiramente realizado na FPGA, e a mesma enviasse para um computador *host* apenas o resultado. Inicialmente, foi estudada a possibilidade de utilizar a placa da Altera DE-01SoC (*System-on-Chip*), pelo fato de ser uma placa já disponível no laboratório, porém quando pesquisamos e buscamos maneiras de executar essa tarefa, logo foi descartada a possibilidade de seu uso, pois como a maneira mais simples de envio dos dados é utilizando a porta USB (*Universal Serial Bus*). Contudo a placa da Altera não possui uma API (*Application Programming Interface*) para que essa comunicação seja feita, então surgiu a solução de ser utilizada uma outra placa FPGA, chamada Saxo da KNJN, pois ela já possui uma API pronta para realizar a comunicação via USB.

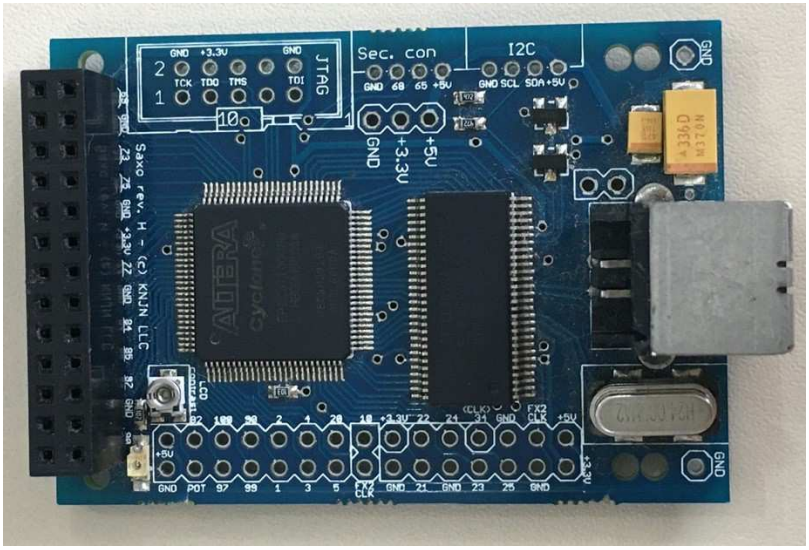
Determinada a ferramenta de trabalho, passamos para a etapa de estudos e compreensão da placa e do seu protocolo de comunicação FX2-USB.

Nas seções seguintes deste capítulo, serão apresentadas a placa utilizada e o protocolo de comunicação.

2.1 PLACA SAXO

Como já exposto, foi escolhido a placa Saxo. Ela é fabricada pela KNJN, uma desenvolvedora norte americana de kits eletrônicos. Uma grande vantagem de suas placas é a facilidade de implementação e simples comunicação USB, o qual possui uma API pronta para tal função, o *FPGA_config*.

Figura 5 - Placa Saxo

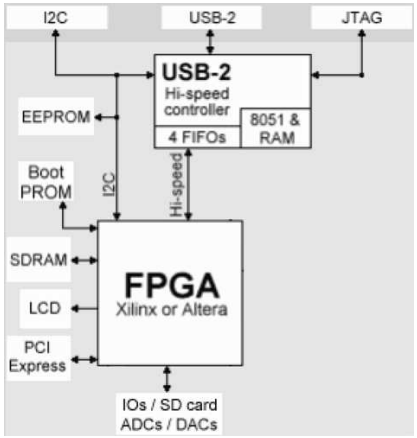


Fonte: Autor

Dos componentes da placa, podemos destacar:

- *Chip* FPGA Altera EP1C3;
- *Chip Cypress* CY7C68013 *high-speed* UBS2 – FX2;
- Porta USB;
- Conjunto de portas de I/O;

Figura 6 - Diagrama de Blocos Saxo



Fonte: (KNJN, 2016) e adaptada pelo autor

A porta USB 2.0 *high-speed*, mostrado no diagrama de blocos, proporciona uma comunicação da FPGA com o *host* na velocidade de 30MB/s a 40MB/s (240Mbit/s a 320Mbit/s).

Para poder utilizar a porta e estabelecer a comunicação, é necessário fazer a instalação dos *drivers* USB, CyUSB.

Como também já destacado, a vantagem para escolha da placa Saxo, foi por ela possuir o *FPGA_config*, a API para comunicação. O processo para sua utilização é prático e rápido, pois ele é um arquivo executável, onde é carregado o programa em *verilog*, que conterá, em nosso caso, o código para o cálculo da FFT e as ordens de envio de dados.

2.2 USB-2.0 – FX2

FX2 é o nome utilizado para fazer referência ao chip *Cyspress CY7C68013 high-speed USB-2* presentes em todas as placas da KNJN. O chip faz a interface de comunicação entre o computador e a FPGA, o qual também permite configurar a placa.

O protocolo de comunicação USB estabelece como padrão dois tipo de pacotes de dados: *Bulk Packets* e *Isochronous Packets*, onde o *Bulk* a entrega é garantida e o *Isochronous* a largaura da banda é garantida.

Na placa Saxo é utilizado o padrão *Bulk*, isso significa que caso ocorra um erro no envio de dados, o dado será retransmitido automaticamente até que a recepção seja confirmada, que é um sinal informando que a entrega foi bem-sucedida.

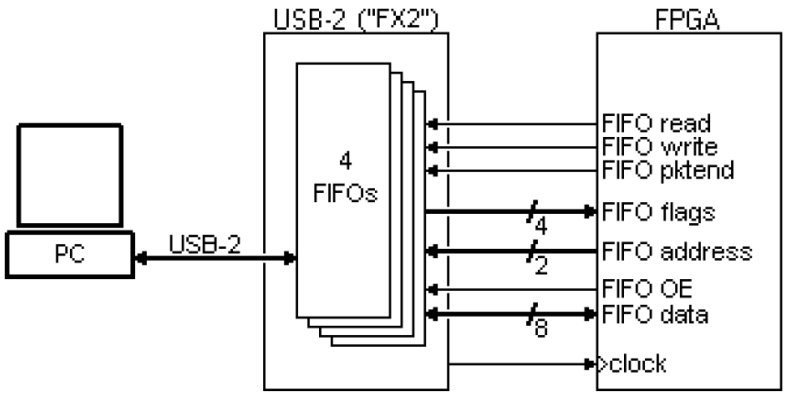
O FX2 está conectado a FPGA por meio de um barramento de dados bidirecional de 8bit. Também implementa 4 FIFO's (*First In, First Out*) em hardware, as quais são

utilizados para fazer a recepção e envio de dados. As FIFO2 e FIFO3 são usadas para o envio de dados do computador para a FPGA, ou seja, o computador escreve e a FPGA lê, já as FIFO4 e FIFO5 fazem o recebimento, no sentido da FPGA para o computador, ou seja, a FPGA escreve e o computador lê.

Para garantir que o envio ou recepção de dados seja realizado de forma correta, o FX2 também conta com quatro *flags*, que são: FLAG2, FLAG3, FLAG4 e FLAG5. As duas primeiras servem para indicar se a placa está pronta para realizar a leitura dos dados e consequentemente, as FLAG4 e FLAG5 indicam se a placa está apta para fazer a escrita dos dados.

Na figura abaixo, podemos visualizar o diagrama de blocos que representa a interface de comunicação.

Figura 7 - Interface FX2



Fonte: (KNJN, 2016)

3 TESTE E VALIDAÇÃO

Na etapa final do estágio, tínhamos como objetivo configurar a placa Saxo para que ela realizasse o cálculo da FFT e enviasse os resultados para o computador. Então, neste capítulo apresentaremos os códigos utilizados, bem como explicar, de maneira geral o que cada um está fazendo e como foi feita a integração.

3.1 FFT

Depois do trabalho de pesquisa, foi utilizado o algoritmo de Cooley-Turkey que possibilitou diminuir a complexibilidade de operações necessárias para calcular a

Transformada Discreta de Fourier, passando a ser chamada de FFT por decimação no tempo.

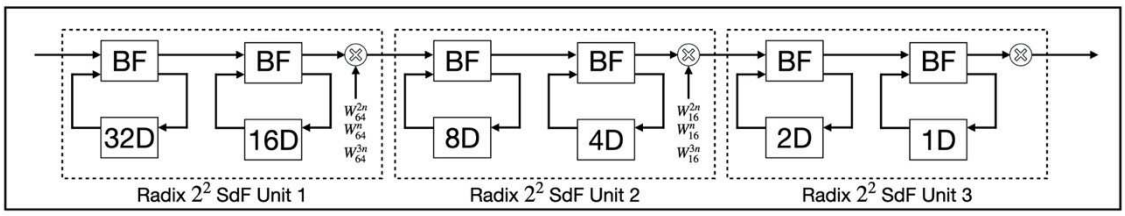
O algoritmo define que necessitamos fazer uma decomposição sucessiva do sinal discreto X[n] em tamanhos menores. É convencionado o uso da decimação no tempo de raiz 2, fazendo a divisão do sinal de N pontos amostrados, em sequências de N/2.

Podemos ainda destacar outra vantagem do algoritmo, é pelo fato dele trabalhar de forma recursiva, sendo mais simples para implementar, uma vez que construímos o processo em pequenos blocos que podem ser reutilizados.

É sabido que quanto maior for o número de pontos amostrados, mais preciso será o resultado da FFT. Devido as limitações de processamento da placa, a qual só tem capacidade de utilizar 2920 células lógicas, e quanto maior for o número de pontos, mais células lógicas vão ser requeridas, devida ao processo de multiplicação, então foi decido implementar uma FFT de 64 pontos, que nos permite fazer a análise de sinais na banda de trabalho, como já especificado anteriormente de 980 a 1260MHz, bem como não ultrapassa o limite de blocos lógicos que podem ser criados pela placa.

Portanto, na figura 8 temos o diagrama de blocos que apresenta o algoritmo de cálculo da FFT.

Figura 8 - FFT de 64 pontos usando raiz 2^2



Fonte: Autor

Uma observação que deve ser feita, é sobre a operação de multiplicação, pois é a operação mais utilizada para o cálculo da FFT. É sabido que cada vez que é feita a multiplicação, existe um *delay*, e esse é propagada durante todo o processo, assim, como mostra a figura 8, dividimos em 3 blocos, chamados de Sdf Unit (*single-path delay feedback*), para dimunir a propagação do *delay.*

Em cada uma das unidades Sdf, vai ser feito duas borboletas e apenas uma multiplicação entre o resultado da decimação com o *twiddle factor*.

O código em *verilog* apresentado, mostra a função principal da FFT de 64 pontos. Seguindo a hierarquia do programa, o módulo da FFT 64 é composto pelo módulo da Sdf

Unit, este por sua vez é composto do módulo da borboleta, do *delay*, do multiplicador e do *twiddle factor*.

```
//-----------------------------------------------------------------
//  FFT: 64-Point FFT Using Radix-2^2 Single-Path Delay Feedback
//-----------------------------------------------------------------
module FFT64 #(parameter WIDTH = 16) (
                input             clock,  //  Master Clock
                input             reset,  //  Active High Asynchronous Reset
                input             di_en,  //  Input Data Enable
                input   [WIDTH-1:0]  di_re,  //  Input Data (Real)
                input   [WIDTH-1:0]  di_im,  //  Input Data (Imag)
                output             do_en,  //  Output Data Enable
                output  [WIDTH-1:0]  do_re,  //  Output Data (Real)
                output  [WIDTH-1:0]  do_im   //  Output Data (Imag)
        );
        //-----------------------------------------------------------------
        //  Data must be input consecutively in natural order.
        //  The result is scaled to 1/N and output in bit-reversed order.
        //  The output latency is 71 clock cycles.
        //-----------------------------------------------------------------

        wire            su1_do_en;
        wire [WIDTH-1:0] su1_do_re;
        wire [WIDTH-1:0] su1_do_im;
        wire            su2_do_en;
        wire [WIDTH-1:0] su2_do_re;
        wire [WIDTH-1:0] su2_do_im;


        SdfUnit #(.N(64), .M(64), .WIDTH(WIDTH)) SU1 (
                .clock  (clock    ),  //  i
                .reset  (reset    ),  //  i
                .di_en  (di_en    ),  //  i
                .di_re  (di_re    ),  //  i
                .di_im  (di_im    ),  //  i
                .do_en  (su1_do_en),  //  o
                .do_re  (su1_do_re),  //  o
                .do_im  (su1_do_im)   //  o
        );
```

```
        SdfUnit #(.N(64), .M(16), .WIDTH(WIDTH)) SU2 (
                .clock  (clock    ),  //  i
                .reset  (reset    ),  //  i
                .di_en  (su1_do_en),  //  i
                .di_re  (su1_do_re),  //  i
                .di_im  (su1_do_im),  //  i
                .do_en  (su2_do_en),  //  o
                .do_re  (su2_do_re),  //  o
                .do_im  (su2_do_im)   //  o
        );

        SdfUnit #(.N(64), .M(4), .WIDTH(WIDTH)) SU3 (
                .clock  (clock    ),  //  i
                .reset  (reset    ),  //  i
                .di_en  (su2_do_en),  //  i
                .di_re  (su2_do_re),  //  i
                .di_im  (su2_do_im),  //  i
                .do_en  (do_en    ),  //  o
                .do_re  (do_re    ),  //  o
                .do_im  (do_im    )   //  o
        );
endmodule
```

```
//-----------------------------------------------------------------------
//  SdfUnit: Radix-2^2 Single-Path Delay Feedback Unit for N-Point FFT
//-----------------------------------------------------------------------
module SdfUnit #(
   parameter   N = 64,     //  Number of FFT Point
   parameter   M = 64,     //  Twiddle Resolution
   parameter   WIDTH = 16  //  Data Bit Length
)(
   input             clock,  //  Master Clock
   input             reset,  //  Active High Asynchronous Reset
   input             di_en,  //  Input Data Enable
   input   [WIDTH-1:0] di_re,  //  Input Data (Real)
   input   [WIDTH-1:0] di_im,  //  Input Data (Imag)
   output             do_en,  //  Output Data Enable
   output  [WIDTH-1:0] do_re,  //  Output Data (Real)
   output  [WIDTH-1:0] do_im   //  Output Data (Imag)
);

//  log2 constant function
```

```
function integer log2;
   input integer x;
   integer value;
   begin
      value = x-1;
      for (log2=0; value>0; log2=log2+1)
         value = value>>1;
   end
endfunction

localparam  LOG_N = log2(N);    //  Bit Length of N
localparam  LOG_M = log2(M);    //  Bit Length of M

//---------------------------------------------------------------------
//  Internal Regs and Nets
//---------------------------------------------------------------------
//  1st Butterfly
reg [LOG_N-1:0] di_count;   //  Input Data Count
wire            bf1_bf;     //  Butterfly Add/Sub Enable
wire[WIDTH-1:0] bf1_x0_re;  //  Data #0 to Butterfly (Real)
wire[WIDTH-1:0] bf1_x0_im;  //  Data #0 to Butterfly (Imag)
wire[WIDTH-1:0] bf1_x1_re;  //  Data #1 to Butterfly (Real)
wire[WIDTH-1:0] bf1_x1_im;  //  Data #1 to Butterfly (Imag)
wire[WIDTH-1:0] bf1_y0_re;  //  Data #0 from Butterfly (Real)
wire[WIDTH-1:0] bf1_y0_im;  //  Data #0 from Butterfly (Imag)
wire[WIDTH-1:0] bf1_y1_re;  //  Data #1 from Butterfly (Real)
wire[WIDTH-1:0] bf1_y1_im;  //  Data #1 from Butterfly (Imag)
wire[WIDTH-1:0] db1_di_re;  //  Data to DelayBuffer (Real)
wire[WIDTH-1:0] db1_di_im;  //  Data to DelayBuffer (Imag)
wire[WIDTH-1:0] db1_do_re;  //  Data from DelayBuffer (Real)
wire[WIDTH-1:0] db1_do_im;  //  Data from DelayBuffer (Imag)
wire[WIDTH-1:0] bf1_sp_re;  //  Single-Path Data Output (Real)
wire[WIDTH-1:0] bf1_sp_im;  //  Single-Path Data Output (Imag)
reg             bf1_sp_en;  //  Single-Path Data Enable
reg [LOG_N-1:0] bf1_count;  //  Single-Path Data Count
wire            bf1_start;  //  Single-Path Output Trigger
wire            bf1_end;    //  End of Single-Path Data
wire            bf1_mj;     //  Twiddle (-j) Enable
reg [WIDTH-1:0] bf1_do_re;  //  1st Butterfly Output Data (Real)
reg [WIDTH-1:0] bf1_do_im;  //  1st Butterfly Output Data (Imag)

//  2nd Butterfly
```

```
reg             bf2_bf;     //  Butterfly Add/Sub Enable
wire[WIDTH-1:0] bf2_x0_re;  //  Data #0 to Butterfly (Real)
wire[WIDTH-1:0] bf2_x0_im;  //  Data #0 to Butterfly (Imag)
wire[WIDTH-1:0] bf2_x1_re;  //  Data #1 to Butterfly (Real)
wire[WIDTH-1:0] bf2_x1_im;  //  Data #1 to Butterfly (Imag)
wire[WIDTH-1:0] bf2_y0_re;  //  Data #0 from Butterfly (Real)
wire[WIDTH-1:0] bf2_y0_im;  //  Data #0 from Butterfly (Imag)
wire[WIDTH-1:0] bf2_y1_re;  //  Data #1 from Butterfly (Real)
wire[WIDTH-1:0] bf2_y1_im;  //  Data #1 from Butterfly (Imag)
wire[WIDTH-1:0] db2_di_re;  //  Data to DelayBuffer (Real)
wire[WIDTH-1:0] db2_di_im;  //  Data to DelayBuffer (Imag)
wire[WIDTH-1:0] db2_do_re;  //  Data from DelayBuffer (Real)
wire[WIDTH-1:0] db2_do_im;  //  Data from DelayBuffer (Imag)
wire[WIDTH-1:0] bf2_sp_re;  //  Single-Path Data Output (Real)
wire[WIDTH-1:0] bf2_sp_im;  //  Single-Path Data Output (Imag)
reg             bf2_sp_en;  //  Single-Path Data Enable
reg [LOG_N-1:0] bf2_count;  //  Single-Path Data Count
reg             bf2_start;  //  Single-Path Output Trigger
wire            bf2_end;    //  End of Single-Path Data
reg [WIDTH-1:0] bf2_do_re;  //  2nd Butterfly Output Data (Real)
reg [WIDTH-1:0] bf2_do_im;  //  2nd Butterfly Output Data (Imag)
reg             bf2_do_en;  //  2nd Butterfly Output Data Enable

//  Multiplication
wire[1:0]       tw_sel;     //  Twiddle Select (2n/n/3n)
wire[LOG_N-3:0] tw_num;     //  Twiddle Number (n)
wire[LOG_N-1:0] tw_addr;    //  Twiddle Table Address
wire[WIDTH-1:0] tw_re;      //  Twiddle Factor (Real)
wire[WIDTH-1:0] tw_im;      //  Twiddle Factor (Imag)
reg             mu_en;      //  Multiplication Enable
wire[WIDTH-1:0] mu_a_re;    //  Multiplier Input (Real)
wire[WIDTH-1:0] mu_a_im;    //  Multiplier Input (Imag)
wire[WIDTH-1:0] mu_m_re;    //  Multiplier Output (Real)
wire[WIDTH-1:0] mu_m_im;    //  Multiplier Output (Imag)
reg [WIDTH-1:0] mu_do_re;   //  Multiplication Output Data (Real)
reg [WIDTH-1:0] mu_do_im;   //  Multiplication Output Data (Imag)
reg             mu_do_en;   //  Multiplication Output Data Enable

//------------------------------------------------------------------------
//  1st Butterfly
//------------------------------------------------------------------------
always @(posedge clock or posedge reset) begin
```

```
    if (reset) begin
        di_count <= {LOG_N{1'b0}};
    end else begin
        di_count <= di_en ? (di_count + 1'b1) : {LOG_N{1'b0}};
    end
end
assign  bf1_bf = di_count[LOG_M-1];

//  Set unknown value x for verification
assign  bf1_x0_re = bf1_bf ? db1_do_re : {WIDTH{1'bx}};
assign  bf1_x0_im = bf1_bf ? db1_do_im : {WIDTH{1'bx}};
assign  bf1_x1_re = bf1_bf ? di_re : {WIDTH{1'bx}};
assign  bf1_x1_im = bf1_bf ? di_im : {WIDTH{1'bx}};

Butterfly #(.WIDTH(WIDTH),.RH(0)) BF1 (
    .x0_re  (bf1_x0_re  ),  //  i
    .x0_im  (bf1_x0_im  ),  //  i
    .x1_re  (bf1_x1_re  ),  //  i
    .x1_im  (bf1_x1_im  ),  //  i
    .y0_re  (bf1_y0_re  ),  //  o
    .y0_im  (bf1_y0_im  ),  //  o
    .y1_re  (bf1_y1_re  ),  //  o
    .y1_im  (bf1_y1_im  )   //  o
);

DelayBuffer #(.DEPTH(2**(LOG_M-1)),.WIDTH(WIDTH)) DB1 (
    .clock  (clock      ),  //  i
    .di_re  (db1_di_re  ),  //  i
    .di_im  (db1_di_im  ),  //  i
    .do_re  (db1_do_re  ),  //  o
    .do_im  (db1_do_im  )   //  o
);

assign  db1_di_re = bf1_bf ? bf1_y1_re : di_re;
assign  db1_di_im = bf1_bf ? bf1_y1_im : di_im;
assign  bf1_sp_re = bf1_bf ? bf1_y0_re : bf1_mj ?  db1_do_im : db1_do_re;
assign  bf1_sp_im = bf1_bf ? bf1_y0_im : bf1_mj ? -db1_do_re : db1_do_im;

always @(posedge clock or posedge reset) begin
    if (reset) begin
        bf1_sp_en <= 1'b0;
        bf1_count <= {LOG_N{1'b0}};
```

```
    end else begin
        bf1_sp_en <= bf1_start ? 1'b1 : bf1_end ? 1'b0 : bf1_sp_en;
        bf1_count <= bf1_sp_en ? (bf1_count + 1'b1) : {LOG_N{1'b0}};
    end
end
assign  bf1_start = (di_count == (2**(LOG_M-1)-1));
assign  bf1_end = (bf1_count == (2**LOG_N-1));
assign  bf1_mj = (bf1_count[LOG_M-1:LOG_M-2] == 2'd3);

always @(posedge clock) begin
    bf1_do_re <= bf1_sp_re;
    bf1_do_im <= bf1_sp_im;
end

//---------------------------------------------------------------------
//  2nd Butterfly
//---------------------------------------------------------------------
always @(posedge clock) begin
    bf2_bf <= bf1_count[LOG_M-2];
end

//  Set unknown value x for verification
assign  bf2_x0_re = bf2_bf ? db2_do_re : {WIDTH{1'bx}};
assign  bf2_x0_im = bf2_bf ? db2_do_im : {WIDTH{1'bx}};
assign  bf2_x1_re = bf2_bf ? bf1_do_re : {WIDTH{1'bx}};
assign  bf2_x1_im = bf2_bf ? bf1_do_im : {WIDTH{1'bx}};

//  Negative bias occurs when RH=0 and positive bias occurs when RH=1.
//  Using both alternately reduces the overall rounding error.
Butterfly #(.WIDTH(WIDTH),.RH(1)) BF2 (
    .x0_re  (bf2_x0_re  ),  //  i
    .x0_im  (bf2_x0_im  ),  //  i
    .x1_re  (bf2_x1_re  ),  //  i
    .x1_im  (bf2_x1_im  ),  //  i
    .y0_re  (bf2_y0_re  ),  //  o
    .y0_im  (bf2_y0_im  ),  //  o
    .y1_re  (bf2_y1_re  ),  //  o
    .y1_im  (bf2_y1_im  )   //  o
);

DelayBuffer #(.DEPTH(2**(LOG_M-2)),.WIDTH(WIDTH)) DB2 (
    .clock  (clock      ),  //  i
```

```
    .di_re  (db2_di_re  ),  //  i
    .di_im  (db2_di_im  ),  //  i
    .do_re  (db2_do_re  ),  //  o
    .do_im  (db2_do_im  )   //  o
);

assign  db2_di_re = bf2_bf ? bf2_y1_re : bf1_do_re;
assign  db2_di_im = bf2_bf ? bf2_y1_im : bf1_do_im;
assign  bf2_sp_re = bf2_bf ? bf2_y0_re : db2_do_re;
assign  bf2_sp_im = bf2_bf ? bf2_y0_im : db2_do_im;

always @(posedge clock or posedge reset) begin
    if (reset) begin
        bf2_sp_en <= 1'b0;
        bf2_count <= {LOG_N{1'b0}};
    end else begin
        bf2_sp_en <= bf2_start ? 1'b1 : bf2_end ? 1'b0 : bf2_sp_en;
        bf2_count <= bf2_sp_en ? (bf2_count + 1'b1) : {LOG_N{1'b0}};
    end
end

always @(posedge clock) begin
    bf2_start <= (bf1_count == (2**(LOG_M-2)-1)) & bf1_sp_en;
end
assign  bf2_end = (bf2_count == (2**LOG_N-1));

always @(posedge clock) begin
    bf2_do_re <= bf2_sp_re;
    bf2_do_im <= bf2_sp_im;
end

always @(posedge clock or posedge reset) begin
    if (reset) begin
        bf2_do_en <= 1'b0;
    end else begin
        bf2_do_en <= bf2_sp_en;
    end
end

//---------------------------------------------------------------------
//  Multiplication
//---------------------------------------------------------------------
```

```
assign  tw_sel[1] = bf2_count[LOG_M-2];
assign  tw_sel[0] = bf2_count[LOG_M-1];
assign  tw_num = bf2_count << (LOG_N-LOG_M);
assign  tw_addr = tw_num * tw_sel;

Twiddle TW (
   .clock  (clock  ),  //  i
   .addr   (tw_addr),  //  i
   .tw_re  (tw_re  ),  //  o
   .tw_im  (tw_im  )   //  o
);

//  Multiplication is bypassed when twiddle address is 0.
always @(posedge clock) begin
   mu_en <= (tw_addr != {LOG_N{1'b0}});
end
//  Set unknown value x for verification
assign  mu_a_re = mu_en ? bf2_do_re : {WIDTH{1'bx}};
assign  mu_a_im = mu_en ? bf2_do_im : {WIDTH{1'bx}};

Multiply #(.WIDTH(WIDTH)) MU (
   .a_re   (mu_a_re),  //  i
   .a_im   (mu_a_im),  //  i
   .b_re   (tw_re  ),  //  i
   .b_im   (tw_im  ),  //  i
   .m_re   (mu_m_re),  //  o
   .m_im   (mu_m_im)   //  o
);

always @(posedge clock) begin
   mu_do_re <= mu_en ? mu_m_re : bf2_do_re;
   mu_do_im <= mu_en ? mu_m_im : bf2_do_im;
end

always @(posedge clock or posedge reset) begin
   if (reset) begin
      mu_do_en <= 1'b0;
   end else begin
      mu_do_en <= bf2_do_en;
   end
end
```

```
//  No multiplication required at final stage
assign  do_en = (LOG_M == 2) ? bf2_do_en : mu_do_en;
assign  do_re = (LOG_M == 2) ? bf2_do_re : mu_do_re;
assign  do_im = (LOG_M == 2) ? bf2_do_im : mu_do_im;

endmodule
```

3.2 COMUNICAÇÃO BIDIRECIONAL

Antes de configurar a placa para o cálculo da FFT, foi necessário compreender como era feita de fato comunicação do computador com a placa, visto que já tínhamos o conhecimento teórico do protocolo e de quais variáveis deveriam ser utilizadas no processo.

Portanto, foi estudado o código fornecido pela desenvolvedora da placa, que estabelece a comunicação de forma bidirecional, a qual, inicialmente o computador faz o envio de 5 bytes, a placa FPGA sinaliza o recebimento dessa informação e consequentemente devolve como resposta ao host o número total de bytes que já foram recebidos.

Foi possível observar como é definido estado que habilitada a escrita de dados como a leitura, por meio de uma máquina de estados.

3.3 PROGRAMA FINAL

Tendo separadamente a ideia da FFT e da comunicação, o processo foi de fazer a integração das duas partes, como mostrado no código abaixo.

```
Module FFTSaxo(
        input FX2_CLK,
        input clk_ADC,
        inout [7:0] FX2_FD,
        input [2:0] FX2_flags,
        output FX2_SLRD, FX2_SLWR,
```

```
        //output FX2_PA_0,
        //output FX2_PA_1,
        output FX2_PA_2,
        output FX2_PA_3,
        output FX2_PA_4,
        output FX2_PA_5,
        output FX2_PA_6,
        output LED,
        input FX2_PA_7
);
////////////////////////////////////////////////////////////////
// Rename "FX2" ports into "FIFO" ports, to give them more meaningful names
// FX2 USB signals are active low, take care of them now
// Note: You probably don't need to change anything in this section

// FX2 outputs
wire FIFO_CLK = FX2_CLK;
//wire FIFO_CLK = clk_ADC;

wire FIFO2_empty = ~FX2_flags[0];
wire FIFO2_data_available = ~FIFO2_empty;
wire FIFO3_empty = ~FX2_flags[1];
wire FIFO3_data_available = ~FIFO3_empty;
wire FIFO4_full = ~FX2_flags[2];
wire FIFO4_ready_to_accept_data = ~FIFO4_full;
wire FIFO5_full = ~FX2_PA_7;
wire FIFO5_ready_to_accept_data = ~FIFO5_full;
//assign FX2_PA_0 = 1'b1;
//assign FX2_PA_1 = 1'b1;
assign FX2_PA_3 = 1'b1;

// FX2 inputs
wire FIFO_RD, FIFO_WR, FIFO_PKTEND, FIFO_DATAIN_OE, FIFO_DATAOUT_OE;
assign FX2_SLRD = ~FIFO_RD;
```

```
        assign FX2_SLWR = ~FIFO_WR;
        assign FX2_PA_2 = ~FIFO_DATAIN_OE;
        assign FX2_PA_6 = ~FIFO_PKTEND;


        wire [1:0] FIFO_FIFOADR;
        assign {FX2_PA_5, FX2_PA_4} = FIFO_FIFOADR;

        // FX2 bidirectional data bus
        wire [7:0] FIFO_DATAIN = FX2_FD;
        wire [7:0] FIFO_DATAOUT;
        assign FX2_FD = FIFO_DATAOUT_OE ? FIFO_DATAOUT : 8'hZZ;


        ////////////////////////////////////////////////////
        // So now everything is in positive logic
        //      FIFO_RD, FIFO_WR, FIFO_DATAIN, FIFO_DATAOUT, FIFO_DATAIN_OE, FIFO_DATAOUT_OE,
FIFO_PKTEND, FIFO_FIFOADR
        //      FIFO2_empty, FIFO2_data_available
        //      FIFO3_empty, FIFO3_data_available
        //      FIFO4_full, FIFO4_ready_to_accept_data
        //      FIFO5_full, FIFO5_ready_to_accept_data


        ////////////////////////////////////////////////////
        // Here we wait until we receive some data
        // We count the number of bytes received and we send that count back


        reg [2:0] state;
        always @(posedge FIFO_CLK)
        case(state)
                3'b000: if( FIFO2_data_available) state <= 3'b001; else state = 3'b010; // wait for data packet in
FIFO2
                3'b001: if(FIFO2_empty) state <= 3'b010;  // wait until end of data packet
                3'b010: if(FIFO4_ready_to_accept_data) state <= 3'b011; else state <= 3'b000; // turnaround
cycle, switch to FIFO4
                3'b011: state <= 3'b100;  // write data (l low byte)
```

```
                3'b100: state <= 3'b101;  // write data (I high byte)
                3'b101: state <= 3'b110;  // write data (Q low byte)
                3'b110: state <= 3'b000;  // write data (Q high byte)
                3'b111: state <= 3'b000;  // end packet
                default: state <= 3'b000;
        endcase

        assign FIFO_FIFOADR = {~((state == 3'b000) | (state == 3'b001)), 1'b0}; //{state[2], 1'b0};  // FIFO2 or
FIFO4
        assign FIFO_RD = (state==3'b001);

        wire read_byte = (state==3'b001) & FIFO2_data_available;
        wire write_I_low = (state == 3'b011) & FIFO4_ready_to_accept_data;
        wire write_I_high = (state == 3'b100) & FIFO4_ready_to_accept_data;
        wire write_Q_low = (state == 3'b101) & FIFO4_ready_to_accept_data;
        wire write_Q_high = (state == 3'b110) & FIFO4_ready_to_accept_data;
        wire write_byte = (write_I_low | write_I_high | write_Q_low | write_Q_high);

        ////////////////////
        //CALCULO DA FFT//
        ////////////////////

        //reg clock;
        //reg reset ;
        //reg di_en - 1'b1;
        //reg [15:0]di_re;
        //reg [15:0]di_im;
        wire do_en;
        wire [15:0] Q_out_fft;
        wire [15:0] I_out_fft;

        wire [15:0] Q_out_fft_w;
        wire [15:0] I_out_fft_w;
```

```
		FFT64 FFT(
			.clock(FIFO_CLK),
			.reset(1'b0),
			.di_en(1'b1),
			.di_re(16'h000f),//Vai ser enviado o Q_out do cordic
			.di_im(16'h0000),// vai ser enviado o I_out do cordic
			.do_en(do_en),
			.do_re(Q_out_fft),
			.do_im(I_out_fft)
		); // calcular a FFT
	//////////////////////
	//FIM DO CALCUO DA FFT//
	//////////////////////

	always @(posedge FIFO_CLK)
	if (do_en) begin
		I_out_fft <= I_out_fft_w;
		Q_out_fft <= Q_out_fft_w;
	end

	// now write the data back
	assign FIFO_DATAOUT = (write_I_low) ? I_out_fft[7:0] : (write_I_high) ? I_out_fft[15:8] : (write_Q_low) ?
Q_out_fft[7:0] : Q_out_fft[15:8];
	assign FIFO_WR = write_byte;
	assign FIFO_PKTEND = 1'b0;//(state==3'b110);
	assign FIFO_DATAIN_OE = ~(state[2] | state[1]);
	assign FIFO_DATAOUT_OE = write_byte; //(state==3'b101);
endmodule
```

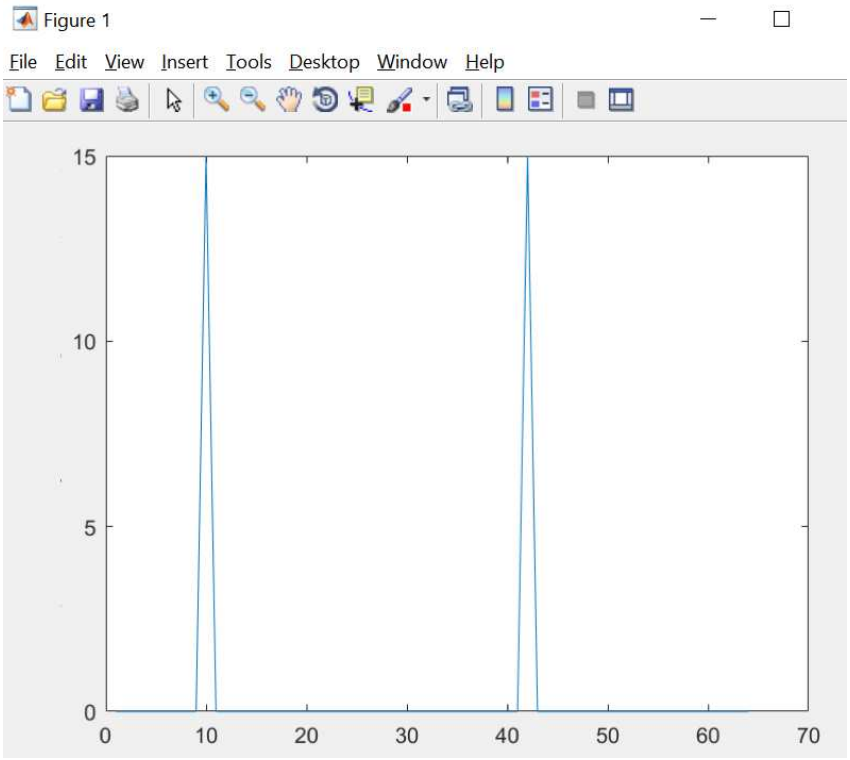
O código apresentado, podemos dividi-lo em quatro partes. A primeira referente a instanciação de todos os sinais que são utilizadas pela FPGA para estabelecer a comunicação. Em seguida temos a máquina de estados, que dependendo do estado vai indicar se a ação deverá ser de leitura ou escrita. O terceiro trecho é a instanciação do

módulo da FFT, que fará os respectivos cálculos e por fim o trecho do envio dos dados. Podemos destacar que, como estamos enviando os dados *byte*-a-*byte*, e desejamos enviar dois números de 16 bits (parte real e parte imaginária), é necessário fazer uma quebra e mandar primeiro a parte menos significativa e depois a parte mais significativa de cada um dos números.

Para poder visualizar o resultado graficamente, com os dados que foram enviados pela FPGA e recebidos pelo computador via USB, foi feito um *script* no Matlab para obter o gráfico.

Obtivemos resultado da FFT para como sendo a entrada apenas o valor um, sem componente imaginária.

Figura 9 - Resultado da FFT



Fonte: Autor

4 CONSIDERAÇÕES FINAIS

Durante o período de estágio, foi possível verificar a importância de parte dos assuntos aprendidos nas disciplinas de Arquitetura de Sistemas Digitais e Análise de Sinais e Sistemas, tiveram no processo, e também utilizá-los como ponto de partida para adquirir novos conceitos, por exemplo, estabelecer a comunicação entre FPGA e o PC via USB e implementar a Transformada de Fourier de maneiras ainda não estudadas, por o processo de decimação no tempo.

Concluímos o trabalho com êxito, pois estabelecemos uma comunicação entre a placa FPGA e o computador, também verificamos que a placa utilizada é capaz de calcular a FFT em tempo real e enviar todos os resultados de seus cálculos e a possibilidade de serem armazenados, sendo esta etapa de grande importância para o projeto BINGO.

Para trabalhos futuros a serem realizados temos etapa seguinte que deve ser estabelecer a recepção de sinais reais, ou seja, simular sinais que serão enviados pelas cornetas e trabalhar para a implementação do espectrômetro.

REFERÊNCIAS

KNJN. *KNJN FX2 FPGA development boards*. KNJN, 2016.

CYPRESS SEMICONDUCTOR. *Cypress CyUSB .NET DLL Programmer's Reference*. CYPRESS,2011.

CYPRESS SEMICONDUCTOR. *Cypress CyAPI Programmer's Reference*. CYPRESS,2011.

Brasil com Ciência. (2019). Conheça o BINGO, o radiotelescópio que vai mapear o universo. Acesso em: 19 de maio de 2021. Disponível em: < https://www.youtube.com/watch?v=FMgAlm7yiZY >.

BINGO. O que é o BINGO. Acesso em: 18 de maio de 2021. Disponível em: < http://150.165.67.4/?page_id=24678 >.

BINGO TELESCOPE. Bayron Acoustic Oscillations In Neutral Gas Observation Telescope - Resumo de Projeto. BINGO TELESCOPE, 2019.

BINGO TELESCÓPIO. O radiotelescópio BINGO. Acesso em: 21 de maio de 2021. Disponível em: < http://portal.if.usp.br/bingotelescope/pt-br/radiotelescopio >.

DIVISÃO DE ASTROFÍSICA INPE. BINGO - Fast Radio Bursts FRB. Acesso em: 22 de maio de 2021. Disponível em: < http://www.das.inpe.br/bingo/frb.php >.

EDMAR CANDEIA. Transformada Rápida de Fourier. Acesso em: 23 de maio de 2021. Disponível em: < https://nbviewer.jupyter.org/github/ecandeia/BINGO/blob/main/FFT.ipynb>.