Luis Adalberto Beloni Bózoli

SISTEMA DE PRÉ-PROCESSAMENTO DE SINAIS ULTRASSÔNICOS PARA SISTEMA DE AQUISIÇÃO COM FOCALIZAÇÃO DINÂMICA

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia de Computação com
ênfase em Sistemas Embarcados

ORIENTADOR: Professor Doutor Carlos Dias Maciel

São Carlos
2010

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Bózoli, Luís Adalberto Beloni
B793s Sistema de pré-processamento de sinais ultrassônicos para sistema de aquisição com focalização dinâmica / Luís Adalberto Beloni Bózoli ; orientador Carlos Dias Maciel. –- São Carlos, 2010.

Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2010.

1. Imageamento (bioengenharia). 2. Ultrassonografia. 3. VHDL. I. Título.

FOLHA DE APROVAÇÃO

Nome: Luís Adalberto Beloni Bózoli

Título: "Sistema de Pré-processamento de Sinais Ultra-sônicos para Sistema de Aquisição com Focalização Dinâmica"

Trabalho de Conclusão de Curso defendido e aprovado em 30 / 11 / 2010,

com NOTA 9,5 (nove, cinco), pela comissão julgadora:

Prof. Titular José Carlos Pereira - SEL/EESC/USP

Msc. Jen John Lee - SEL/EESC/USP

Prof. Dr. Evandro Luís Linhari Rodrigues
Coordenador pela EESC/USP do
Curso de Engenharia de Computação

Autor: Luis Adalberto Beloni Bózoli

Título: Sistema de pré-processamento de sinais ultrassônicos para sistema de aquisição com focalização dinâmica

Data: 08 de novembro de 2010

Orientador: Prof. Dr. Carlos Dias Maciel

Área de Concentração: Engenharia Biomédica

Resumo

Nesse projeto temos a finalidade de criar um *beamformer* de 8 canais para ser utilizado em um sistema de ultrassom com focalização dinâmica utilizando como base duas placas fornecidas pelas **Texas Instruments**, um *front-end* de ultrassom **AFE5805EVM** e uma placa destinada a testar esse *front-end*, a **TSW1250EVM**.

O método empregado foi o de modificar o funcionamento da **TSW1250EVM** por meio da alteração do código da FPGA modelo **Virtex-4 VLX25** implementando um algoritmo de focalização por atraso e soma. Para que o sistema fosse funcional foi também desenvolvida uma interface de alta velocidade para que os dados gerados pelo sistema fossem enviados para a placa de processamento de sinais **TMS320C6455 DSK-MI** para posterior análise.

Palavras-chaves: focalização dinâmica, formação de feixes, ultrassom, FPGA, VHDL, TSW1250EVM

Author: Luis Adalberto Beloni Bózoli

Title: Pre-processing system of ultrasonic signals for acquisition system with dynamic focus

Date: November 08, 2010

Advisor: Prof. Dr. Carlos Dias Maciel

Concentration Area: Biomedical Engineering

Abstract

In this project we aim to create a 8-channel beamformer to be used in a ultrasound system with dynamic focus using two boards provided by **Texas Instruments** as base, an ultrasound front-end **AFE5805EVM** and a board designed to test this front-end, the **TSW1250EVM**.

The method used was to modify the operation of **TSW1250EVM** by changing the code of the FPGA, whose model is **Virtex-4 VLX25**, implementing a focalisation algorithm by delay and summation. To assure that the system is functional, a high-speed interface has also developed to send the generated data to the signal processing board **TMS320C6455 DSK-MI** for further analysis

Keywords: dynamic focalisation, beamforming, ultrasound, FPGA, VHDL, TSW1250EVM

Lista de figuras

Figura 1.1 – Diagrama de blocos do fluxo de dados da arquitetura original do sistema.....9

Figura 2.1 – (a) Campo acústico de onda ultrassônica de 4 MHz em água de um transdutor circular com 10 mm de diâmetro (b) Campo acústico de onda ultrassônica de 4 MHz em água de um transdutor circular com 10 mm de diâmetro e foco a 30 mm [ref. 10]..13

Figura 2.2 – Sistema de ultrassom com focalização por transmissão ajustável [ref. 4]....14

Figura 2.3 – Sistema de ultrassom com focalização por recepção dinâmica.....14

Figura 2.4 – (a) Sistema de ultrassom com beamforming analógico (b) Sistema de ultrassom com beamforming digital [ref. 1].....15

Figura 3.1 – Diagrama de blocos do front-end de ultrassom AFE5805 [ref. 18].....19

Figura 3.2 – Diagrama de temporização da saída LVDS, com o clock do quadro (FCLK), clock de dados (LCLK) e dados seriais [ref. 18].....20

Figura 3.3 – Sistema montado para o desenvolvimento, onde 1 é a placa AFE5805EVM, 2 é a placa TSW1250EVM, 3 é o programador DLC9G e 4 é o osciloscópio DS1102DC.....21

Figura 3.4 – Desenho esquemático do circuito de paralelização, com SDATA sendo a entrada de dados serial e DCLK o clock de dados.....24

Figura 3.5 – Desenho esquemático da fila de dados, com Seletor sendo a entrada seletora da posição da fila de dados serial e FCLK o clock de quadros.....25

Figura 3.6 – Desenho esquemático dos 8 canais integrados, com SDATA sendo a entrada de dados serial e DCLK o clock de dados e FCLK o clock de quadros.....26

Figura 3.7 – Protocolo de saída de dados para a placa de processamento de sinais.....30

Figura 4.1 – Formas de onda do teste do circuito de paralelização, onde os dados da função rampa são exibidos em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos.....32

Figura 4.2 – Formas de onda do teste da fila de dados, onde os dados da função rampa adquiridos do penúltimo valor da fila são exibidos em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos.....33

Figura 4.3 – Formas de onda do primeiro teste de integração, em que são somados os terceiros valores das filas dos 8 canais sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos.....35

Figura 4.4 – Formas de onda do segundo teste de integração somando o sexagésimo valor da fila de cada um dos canais sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor

para a visualização dos bits mais significativos.....36
Figura 4.5 – Formas de onda do terceiro teste de integração somando o primeiro valor da fila de metade dos canais e o sexto valor da outra metade sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos.....37
Figura 4.6 – Formas de onda do teste do protocolo comunicação para saída de dados. 38
Figura 4.7 – Forma de onda obtida no teste do sistema de beamforming.....38

Lista de tabelas

Tabela 2.1 – Velocidade de propagação da onda [ref. 6 e ref. 8].....11

Tabela 3.1 – Relação das conexões entre os pinos da FPGA e os sinais LVDS vindas do front-end de ultrassom.....22

Tabela 3.2 – Conexões entre sinais lógicos, pinos da FPGA e pinos dos conectores da placa TSW1250EVM.....23

Tabela 3.3 – Perfis de atraso para um transdutor como um de 8 elementos de mesma área, 30 mm de diâmetro, velocidade do som de 1540 m/s e distância focal variando entre 20 mm e 120 mm.....27

Tabela 3.4 – Dados dos perfis de atraso da Tabela 3.3 convertidos para utilização no projeto.....28

Tabela 4.1 – Tabela usada para testes no projeto.....34

Sumário

Resumo....1
Abstract....2
Glossário....8
Capítulo 1 – Introdução....9
1.1. Objetivos....9
1.2. Estrutura da monografia....10
Capítulo 2 – Teoria....11
2.1. Ultrassom....11
2.2. Transdutor....12
2.2.1. Foco....12
2.2.2. Focalização dinâmica....13
Capítulo 3 – Materiais e Métodos....17
3.1. VHDL....17
3.2. FPGA....17
3.3. Hardware utilizado....18
3.3.1. Placa de Aquisição AFE5805EVM....18
3.3.2. Protocolo de comunicação da saída LVDS....19
3.3.3. Placa de análise LVDS TSW1250EVM....20
3.3.4. Placa de processamento de sinais TMS320C6455 DSK-MI....20
3.4. Implementação....21
3.4.1. Definição das conexões com a FPGA....22
3.4.2. Entrada de dados LVDS....22
3.4.3. Desenvolvimento da interface entre placa e FPGA....23
3.4.4. Geração do clock de amostragem....24
3.4.5. Circuito de paralelização....24
3.4.6. Fila de dados....25
3.4.7. Integração dos 8 canais....26
3.4.8. Somador....26
3.4.9. Beamforming....27
3.4.10. Saída de dados....29
Capítulo 4 – Resultados e Discussões....31
Capítulo 5 – Conclusão....40
Bibliografia....41
Anexos....44
Códigos VHDL de cada módulo do projeto....44

Módulo de interface (principal).....44
Módulo de integração.....46
Módulo da fila.....49
Módulo do circuito paralelização.....50
Módulo de soma.....51

Glossário

ASIC: circuito integrado desenvolvido para executar uma tarefa especifica

Beamformer: sistema que implementa a técnica de *beamforming*

Beamforming: técnica de processamento de sinais usada em arranjos de sensores para transmissão ou recepção direcional de sinais

Buffer: região de memória temporária

DCLK: sinal do *clock* de dados

DSP: processador otimizado para o processamento de sinais digitais

EEPROM: um tipo de memória não-volátil

FCLK: sinal do *clock* de palavras

FPGA: dispositivo de lógica programável capaz de ter seu funcionamento modificado quando desejado

Front-end: sistema responsável pelos estágios iniciais de um processo, no caso de um front-end de ultrassom ele é responsável por fazer a aquisição dos dados ultrassônicos

JTAG: porta amplamente utilizada para depuração e programação de circuitos

LVDS: padrão de comunicação diferencial com sinais de baixa tensão

VHDL: linguagem de projeto de circuitos e implementação de circuitos digitais

Capítulo 1 – Introdução

O ultrassom é uma onda mecânica em uma frequência superior as captadas pelo ouvido humano, sendo assim ondas com frequências superiores a 20 kHz. O maior uso do ultrassom é na geração de imagens de ultrassonografia, que consiste na geração de imagens a partir da análise dos ecos das ondas ultrassônicas com frequências geralmente na faixa de 1 MHz a 15 MHz [ref. 1].

A geração das ondas ultrassônicas e a captura de seus ecos são feitas por meio de transdutores ultrassônicos. Entre os transdutores, existem os focalizados, capazes de obter uma melhor qualidade na aquisição de sinais na sua zona focal e consequentemente uma maior resolução nas imagens geradas desta região, porém, como essa zona focal é limitada, a região ideal de aquisição de sinais também é. Esse problema pode ser resolvido com um transdutor de múltiplos elementos aliado a um sistema de *beamforming* capaz de ajustar a zona focal desejada através da defasagem dos sinais capturados pelos diferentes elementos do transdutor [ref. 1].

1.1. Objetivos

O principal objetivo desse trabalho foi a construção de um *beamformer* utilizando uma arquitetura fornecida pela **Texas Instruments**[1] como sistema de desenvolvimento. A arquitetura original [ref. 2 e ref. 3], representada na Figura 1.1 com um diagrama de blocos do fluxos de dados, segue uma abordagem de osciloscópio onde se escolhe individualmente cada canal que se pretende converter. A nossa proposta foi implementar um *beamformer* com 8 canais para aplicações em transdutores anulares para o controle de profundidade de forma dinâmica.

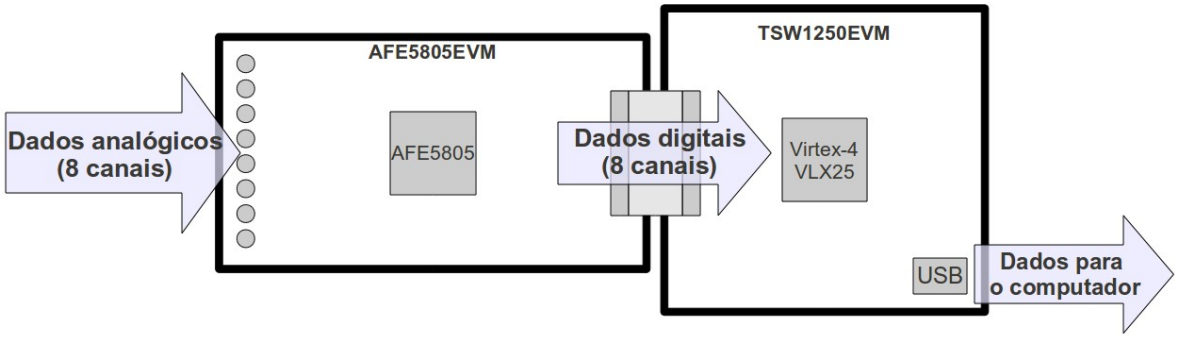


Figura 1.1 – Diagrama de blocos do fluxo de dados da arquitetura original do sistema

A placa **AFE5805EVM** é um *front-end* de ultrassom de 8 canais com uma saída diferencial serial para cada um desses canais para a placa **TSW1250EVM**, que é uma placa

1 http://www.ti.com/

com um circuito de paralelização implementado em uma FPGA (*Field Programmable Gate Array*). Essa FPGA pode ser reprogramada para atender aos objetivos do projeto, criando o sistema de *beamforming*.

Os objetivos específicos foram:

- Estudar sistemas de ultrassom e suas implementações;
- Estudar a documentação das placas **AFE5805EVM** e **TSW1250EVM** e propor uma nova revisão do projeto de forma a se poder implementar algoritmos de focalização dinâmica;
- Implementar o algoritmo de focalização por atraso e soma em dispositivos de lógica programável do tipo FPGA;
- Transferir os dados da FPGA para a placa de processamento de sinais digitais (digital signal processor) da **Texas Instruments** modelo **TMS320C6455 DSK-MI**.

1.2. Estrutura da monografia

Os demais capítulos serão divididos da seguinte forma:

- Capítulo 2 – Neste capítulo será apresentada toda a teoria necessária para o entendimento e implementação do sistema;
- Capítulo 3 – Neste capítulo serão apresentados as ferramentas e equipamentos utilizados no projeto, bem como o método empregado;
- Capítulo 4 – Neste capítulo serão apresentados os resultados obtidos no projeto assim como uma discussão sobre eles;
- Capítulo 5 – Neste último capítulo estão descritas as conclusões obtidas com o término do projeto.

Capítulo 2 – Teoria

Nesse capítulo serão explicados os seguintes tópicos teóricos relacionados a esse projeto: ultrassom, transdutores, foco em transdutores e sistemas de focalização dinâmica em transdutores com múltiplos elementos.

2.1. Ultrassom

O som é uma onda mecânica capaz de transferir energia de um ponto a outro [ref. 4]. Sendo uma onda mecânica, o som necessita de um meio material para ser transmitido [ref. 5], como sólidos, líquidos e gases. Desta forma, não é possível a transferência dessa energia no vácuo.

As ondas mecânicas podem ser classificadas em dois tipos, longitudinais e transversais [ref. 6] ou pela sua combinação. Nas ondas longitudinais, as quais ocorrem em meios sólidos, líquidos ou gasosos, as vibrações se dão na mesma direção da propagação da onda, enquanto nas ondas transversais, que ocorrem somente em meios sólidos, as vibrações são perpendiculares a direção de propagação da onda [ref. 7].

Uma das mais importantes características da onda mecânica é a sua frequência. A unidade básica de medida da frequência é o *hertz* (Hz), que representa o número de vibrações, ou ciclos, da onda por segundo.

O sistema auditivo humano não é sensível a toda a gama de frequências existentes, sendo capaz de ouvir em geral frequências entre 20 Hz e 20 kHz . Desse modo, as ondas mecânicas com frequência superior a 20 kHz são consideradas ondas ultrassônicas. Em sistemas de diagnóstico por ultrassom costumam ser utilizadas ondas entre 1 MHz e 15 MHz [ref. 4].

Outra característica importante das ondas ultrassônicas é a sua velocidade de propagação em diferentes meios. Esta sofre uma grande variação dependendo do material onde está sendo aplicada [ref. 6]. Na Tabela 2.1 são apresentadas as velocidades de propagação para diferentes materiais.

Tabela 2.1 – Velocidade de propagação da onda [ref. 6 e ref. 8]

Meio	**Velocidade (m/s)**
Ar (20°C)	344
Água (20°C)	1480
Sangue	1570
Gordura	1460

Músculo	1580
Osso	3500
Tecido mole (média)	1540

2.2. Transdutor

Um transdutor é um dispositivo que converte um tipo de energia para outro [ref. 6]. O transdutor ultrassônico é responsável pela geração das ondas ultrassônicas, o qual tanto converte energia elétrica em energia mecânica quanto energia mecânica em elétrica [ref. 6]. Esses transdutores são feitos com materiais piezoelétricos como, por exemplo, o cristal de quartzo ou cerâmicas como o PZT [ref. 9].

Esses materiais apresentam o chamado efeito piezoelétrico, o que significa que quando aplicado um campo elétrico sobre eles ocorre uma variação nas dimensões suas dimensões [ref. 6]. O efeito contrário também ocorre, quando é aplicada uma pressão sobre esses materiais é possível detectar campos elétricos sobre os mesmos.

Os materiais piezoelétricos podem ser cortados de forma que a aplicação de campos elétricos causem a variação de sua espessura e serem usados como transdutores [ref. 6].

Dependendo da construção do transdutor e da aplicação de um campo elétrico alternado com a frequência adequada é possível gerar-se ondas de ultrassom. Esse mesmo transdutor criado também pode fazer as leituras dos sinais gerados pelos ecos, que após a amplificação podem ser apresentados num osciloscópio ou feito um processamento para a geração de imagens [ref. 4].

2.2.1. Foco

Os transdutores podem ser desenvolvidos de modo que produzam um feixe de onda de ultrassom tanto não focalizado, onde existe uma região que apresenta um foco natural difuso que varia de acordo com o diâmetro do transdutor e o comprimento de onda no meio, como na Figura 2.1a, quanto focalizado, como na Figura 2.1b. Os feixes focalizados costumam ser os mais usados para a geração de imagens pois ele produz imagens com mais detalhes na zona focal, devido a menor largura do feixe de ultrassom e maior relação sinal-ruído.

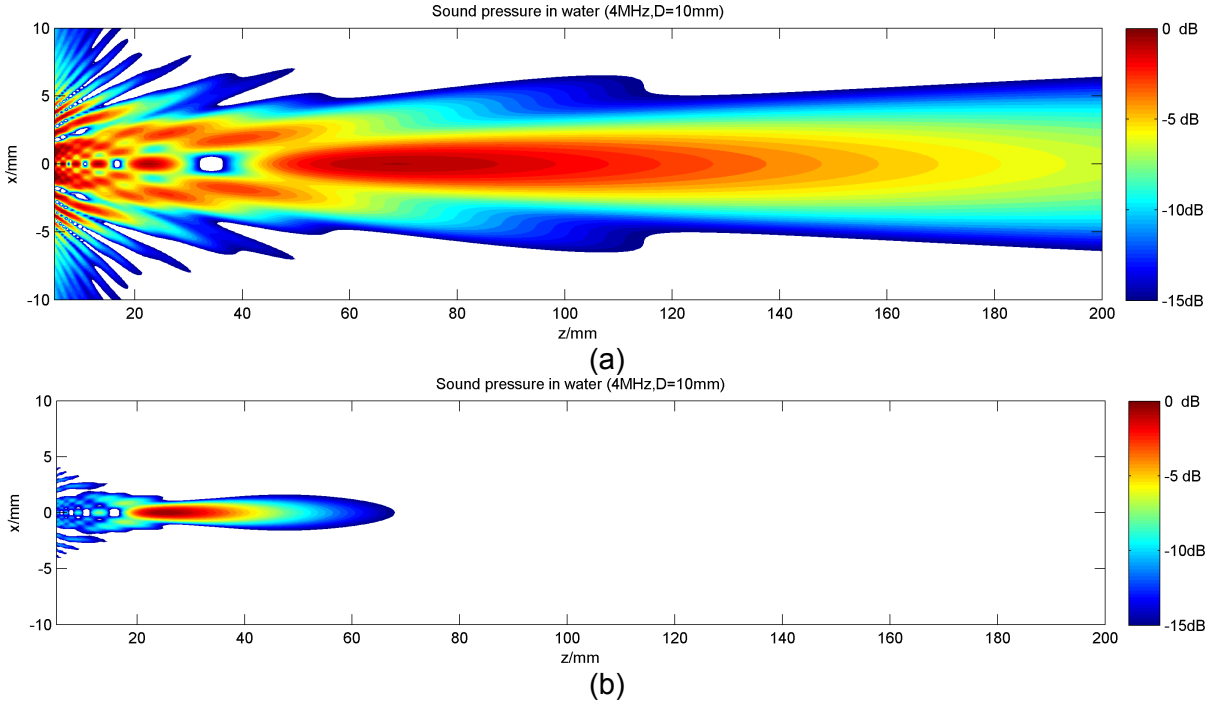


Figura 2.1 – (a) Campo acústico de onda ultrassônica de 4 MHz em água de um transdutor circular com 10 mm de diâmetro (b) Campo acústico de onda ultrassônica de 4 MHz em água de um transdutor circular com 10 mm de diâmetro e foco a 30 mm [ref. 10]

O uso do foco gera uma imagem melhor na zona focal mas devido a rápida divergência do campo obtêm-se imagens com pior qualidade fora desta região. Este problema pode ser solucionado com o uso de transdutores de foco variável ou focalização dinâmica no lugar dos transdutores de foco fixo.

2.2.2. Focalização dinâmica

Para a focalização dinâmica dos feixes de ultrassom é necessário que os transdutores utilizados sejam compostos por mais de um elemento posicionados tradicionalmente em um arranjo linear ou anular [ref. 4]. A diferença entre os dois tipos de arranjos é que no linear o feixe é focalizado em duas dimensões, enquanto no anular em apenas uma, sobre o seu eixo. O arranjo linear é o mais usado para geração de imagens, enquanto o anular precisa ser movimentado mecanicamente (além do foco) para se gerar as imagens.

Os sistemas de focalização dinâmica tanto operam na transmissão quanto na recepção.

Na transmissão os pulsos elétricos são enviados com atrasos entre os diversos elementos. Primeiro os pulsos para o(s) elemento(s) mais externo(s) e posteriormente para os mais internos [ref. 4]. O atraso entre os envios para cada elemento varia conforme as dimensões do transdutor e a distância focal desejada. Na Figura 2.2 é apresentado um

exemplo do funcionamento de um sistema como este.

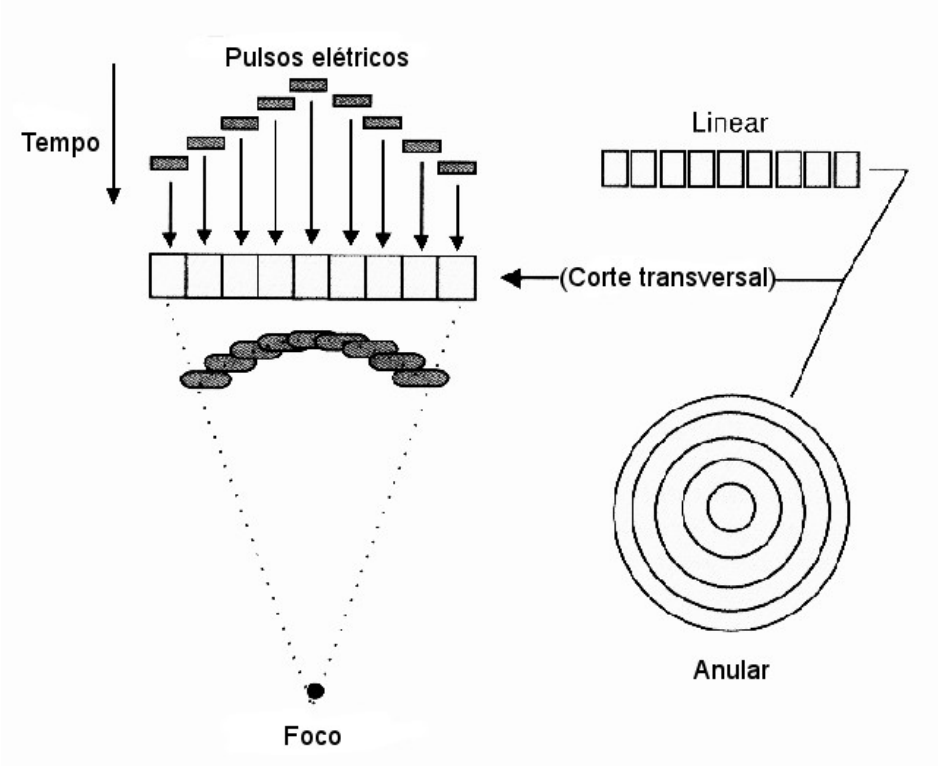


Figura 2.2 – Sistema de ultrassom com focalização por transmissão ajustável [ref. 4]

Já na recepção os pulsos ultrassônicos são primeiramente recebidos no elemento mais central e posteriormente nos mais externos devido à velocidade de propagação da onda no meio ser constante. Como se pode observar na Figura 2.3 os ecos chegam primeiramente nos elementos centrais. Assim como no caso da focalização com ajuste de transmissão os atrasos entre as recepções de cada elemento varia de acordo com as dimensões do transdutor e a distância focal pretendida [ref. 4].

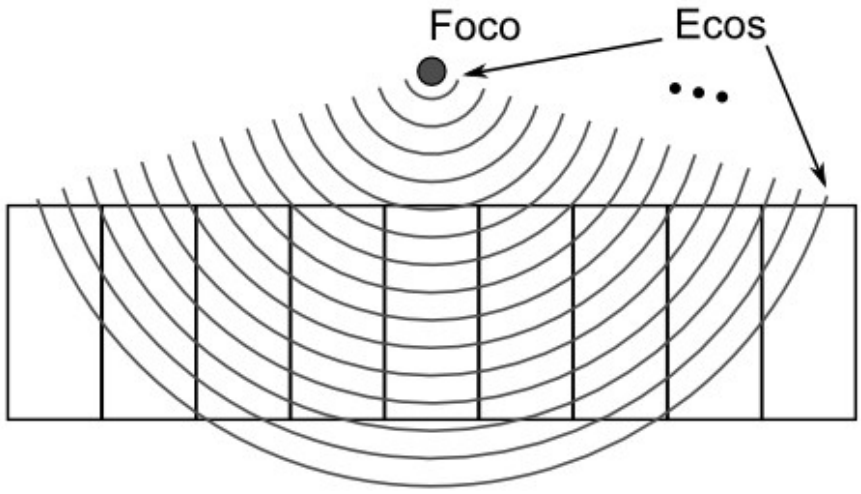
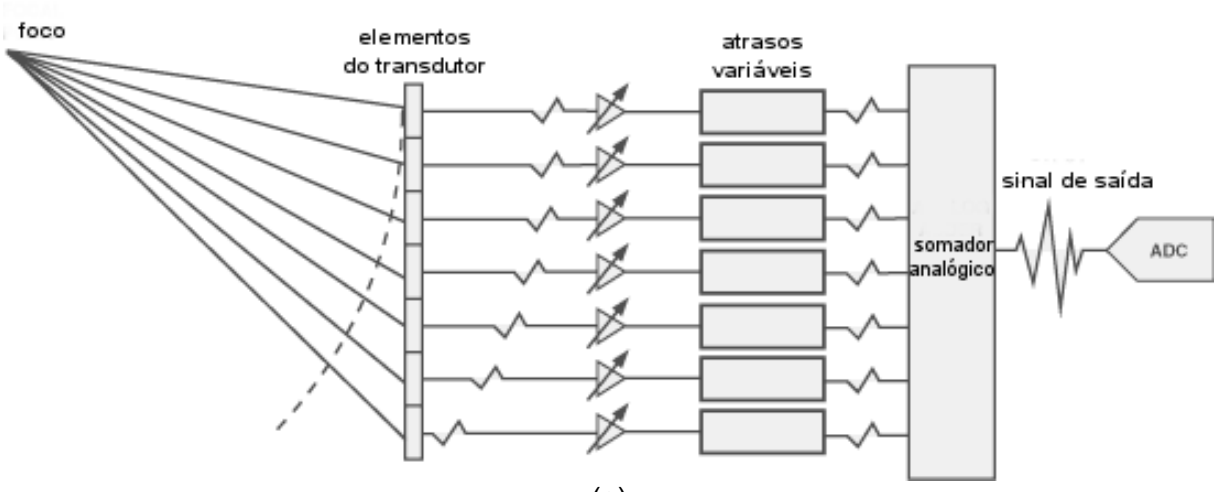


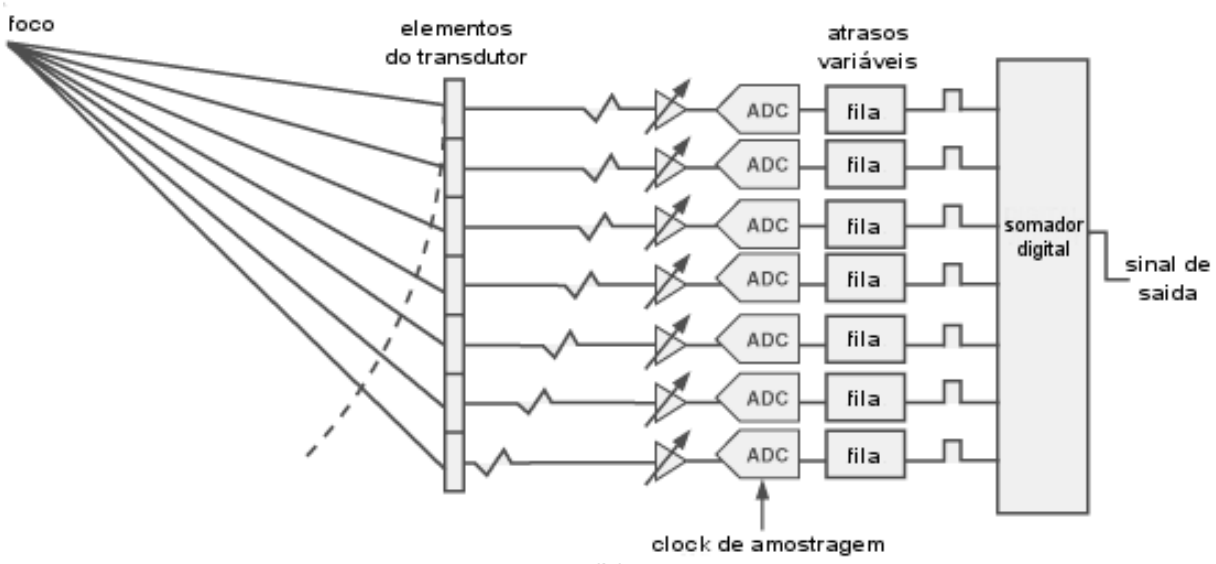
Figura 2.3 – Sistema de ultrassom com focalização por recepção dinâmica

Existem dois modos de se implementar um sistema de focalização dinâmica operando na recepção: o sistema de ultrassom com *beamforming* analógico e o com *beamforming* digital [ref. 1]. Ambos os modos de implementação necessitam que seja criado um atraso diferente para cada elemento, dependendo de onde se deseja focalizar, e que após os atrasos os valores de cada canal sejam somados.

Para o sistema analógico (Figura 2.4a) é necessária a implementação analógica tanto dos circuitos de atrasos ajustáveis quanto do somador, porém é necessário apenas um conversor analógico-digital de alta velocidade e resolução. Já para o sistema digital (Figura 2.4b), apesar da necessidade de um conversor analógico-digital para cada elemento, todo os circuitos atrasadores e o somador podem ser feitos digitalmente, o que facilita o desenvolvimento [ref. 1].



(a)



(b)

Figura 2.4 – (a) Sistema de ultrassom com *beamforming* analógico (b) Sistema de ultrassom com *beamforming* digital [ref. 1]

Nesse projeto é utilizado o sistema digital de *beamforming*, com a aquisição e armazenamento dos sinais de cada elemento em uma fila e posteriormente de acordo com a distância focal selecionam-se os valores para cada atraso pré-calculado e somam-se os valores de cada elemento [ref. 4].

Capítulo 3 – Materiais e Métodos

Nessa seção serão apresentadas as principais ferramentas utilizadas no projeto, como ele foi implementado e como se dá seu funcionamento.

3.1. VHDL

O VHDL (*Very-high-speed integrated circuit Hardware Description Language*) é uma linguagem de descrição de *hardware* usada para facilitar o projeto de circuitos digitais e mistos tanto em FPGAs (explicada na sequência) quanto em ASICs (*Application-Specific Integrated Circuit*) .

Essa linguagem foi inicialmente desenvolvida pelo departamento de defesa americano [ref. 11] com o propósito de ser um meio mais compacto e simples de documentar o funcionamento de circuitos eletrônicos. Posteriormente foram desenvolvidos simuladores lógicos para simular esses códigos em VHDL e ferramentas de síntese capazes de converter esses códigos na definição da implementação física do circuito.

3.2. FPGA

Uma FPGA (*Field-programmable gate array*) é um circuito integrado feito para ser configurado pelo usuário ou projetista após a fabricação da mesma [ref. 12]. Essa configuração é geralmente especificada por meio de uma linguagem de descrição de hardware, como o VHDL, e pode ser refeita inúmeras vezes, sendo capaz de implementar qualquer função lógica de um ASIC [ref. 12].

As FPGAs têm como vantagens sobre os ASICs a possibilidade de atualização do *hardware* mesmo após o produto já estar pronto e o menor custo para uma produção em menor escala e prototipagem [ref. 13]. Outro concorrente são os microcontroladores programáveis, mas estes não possibilitam que o sistema funcione paralelamente, assim como qualquer FPGA [ref. 12].

As FPGAs são formadas por blocos de entrada e saída e uma série de células lógicas interligadas em que a função de cada célula e como elas são interligadas é definido com a síntese do circuito a partir da descrição do *hardware*. Em adição a essas células programáveis, alguns modelos de FPGA possuem recursos adicionais tanto digitais, como DSPs e memória RAM, quanto analógicos, como conversores analógico-digitais.

A empresa inventora das FPGAs é chamada **Xilinx**[2], fundada em 1984 por dois engenheiros de semicondutores e líder desse mercado desde então. Atualmente o **Xilinx**

2 http://www.xilinx.com/

tem em seu portfólio de produtos FPGAs, CPLDs (Complex Programmable Lógic Devices), ferramentas de projeto, propriedades intelectuais e *designs* de referência. Suas FPGAs são divididas em três famílias, com a **Virtex** como a superior, a **Kintex** como intermediaria e a **Artix** sendo a família de baixo custo.

Virtex é a família de FPGAs de maior desempenho fabricada pela **Xilinx**, caracterizada pela integração em *hardware* de diversos recursos frequentemente utilizados em aplicações diversas. Em 2004 foi introduzida a série **Virtex-4**, composta de três famílias de plataformas, oferecendo múltiplas opções de recursos, como núcleos de processadores PowerPC, blocos dedicados de DSPs e circuitos gerenciadores de *clock* de alta velocidade [ref. 14].

O modelo especifico contido na placa **TSW1250EVM**, a **Virtex-4 VLX25**, é um dos mais simples, possuindo como recursos de interesse para esse projeto 48 blocos XtremeDSP, multiplicadores ou multiplicadores com somadores de 18 bits, 8 gerenciadores digitais de *clock* (DCM) [ref. 15], capazes de sintetizar frequências ou deslocar a fase de um sinal periódico, e portas de entrada e saída diferenciais em diversos padrões, incluindo o padrão LVDS [ref. 16].

3.3. Hardware utilizado

3.3.1. Placa de Aquisição AFE5805EVM

A placa de aquisição **AFE5805EVM** [ref. 2] da **Texas Instruments** é uma placa desenvolvida para o teste do circuito integrado **AFE5805** em conjunto com a placa **ADSDeSer-50EVM** [ref. 17] ou **TSW1250EVM** [ref. 3].

O **AFE5805** é um *front-end* de ultrassom integrado de 8 canais com amostragem de até 50 MSPS(milhões de amostas por segundo), resolução de 12 bits e saída de dados diferenciais LVDS (Low-voltage differential signaling) criado para sistemas de ultrassom de baixo consumo [ref. 18]. A utilização de um padrão diferencial é de extrema importância em um sistema de ultrassom devido a sua tolerância a ruídos já que devido a utilização de várias fontes de tensão nestes sistemas muito ruído é gerado. Na Figura 3.1 é apresentado o diagrama de blocos do *front-end*.

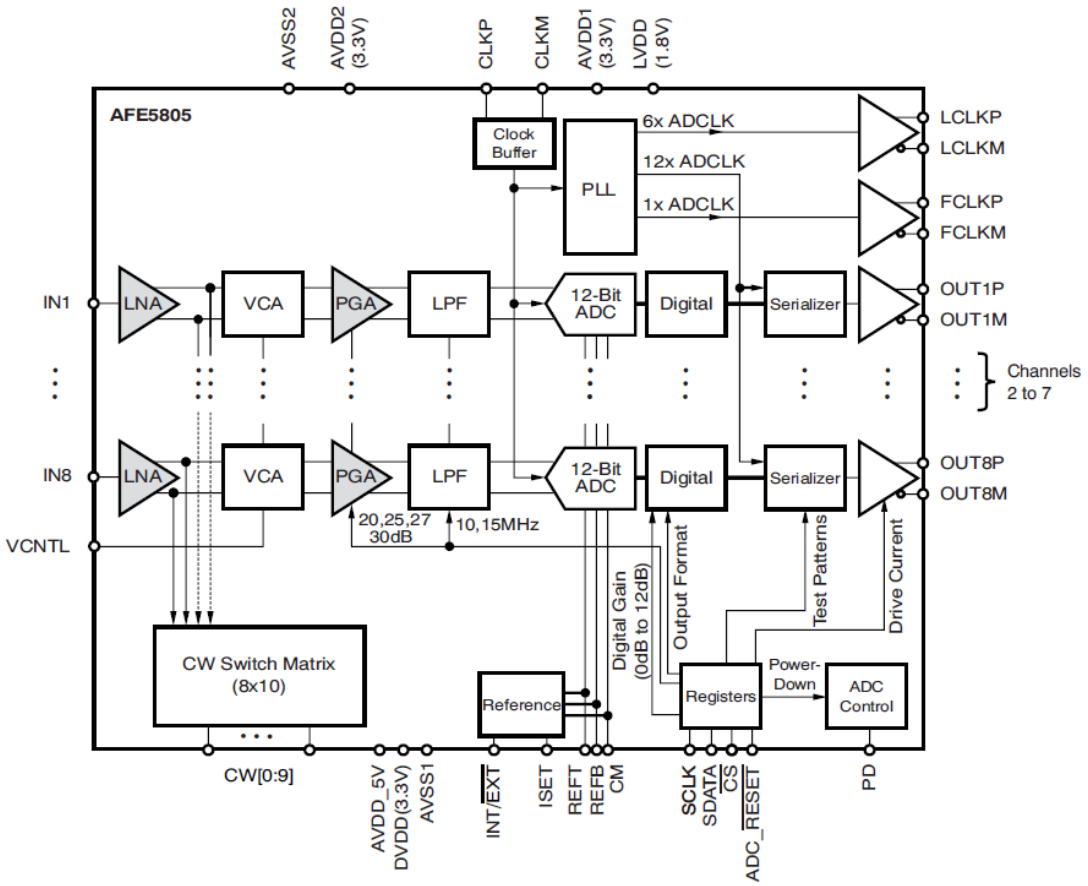


Figura 3.1 – Diagrama de blocos do *front-end* de ultrassom AFE5805 [ref. 18]

Essa placa disponibiliza dentre outros recursos 8 entradas analógicas para os sinais do ultrassom, entrada de *clock* externo, caso do *clock* interno de 40 MHz não ser adequado a aplicação, comunicação com o computador por interface USB para configuração da placa e um conector para a saída LVDS dos 8 canais.

3.3.2. Protocolo de comunicação da saída LVDS

A **Texas Instruments** tem uma série de conversores analógico-digital com saída LVDS serial, e nesse grupo se insere o *front-end* dessa placa. Para esses circuitos ela decidiu utilizar um sistema em que é usado um par de fios para cada canal do conversor, um para o *clock* do quadro, que determina onde começa e termina cada palavra de dados, e um para o *clock* de dados, que determina quando ler o dado de cada canal.

Desse modo, no caso desta placa a saída é composta de dez pares de fios, oito para dados e dois para os *clocks*, com o *clock* de dados sendo seis vezes maior do que o *clock* do quadro, pois ocorrem leituras dos dados tanto na borda de subida quanto na borda de descida desse *clock*, totalizando doze leituras em cada quadro, como pode-se observar na Figura 3.2.

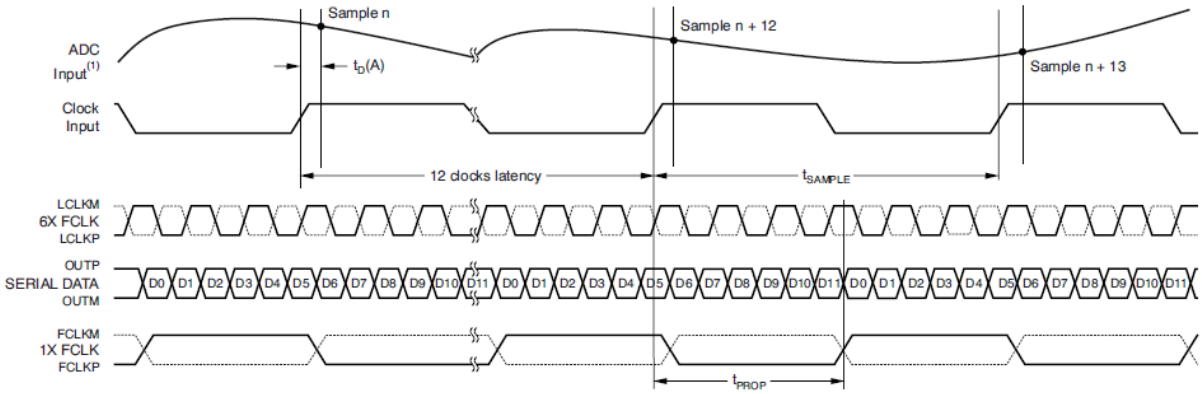


Figura 3.2 – Diagrama de temporização da saída LVDS, com o clock do quadro (FCLK), clock de dados (LCLK) e dados seriais [ref. 18]

3.3.3. Placa de análise LVDS TSW1250EVM

A placa de análise **TSW1250EVM** é uma placa fabricada pela **Texas Instruments** com o intuito de analisar o desempenho dos dispositivos da **TI** da série **AFE58xx**. É um sistema que por meio da placa aliada ao *software* incluso analisa a capacidade dessa série de dispositivos por meio de diversos tipos de teste e geração de estatísticas apresentados na interface gráfica do *software*.

Essa placa possui dentre outros recursos um conector de alta velocidade para uma entrada LVDS com até 28 canais, 4 conectores de 40 vias e 4 conectores de 32 vias usados como saídas paralelas para os dados de 8 canais e conexão USB. Nessa placa tudo é controlado por uma FPGA **Virtex-4 VLX25** que recebe seu arquivo de configuração de uma memória EEPROM inclusa que pode ser regravada por meio do conector JTAG contido na placa, reprogramando assim a FPGA e, consequentemente, alterando todo o funcionamento da placa.

3.3.4. Placa de processamento de sinais TMS320C6455 DSK-MI

O **TMS320C6455 DSP Starter Kit for Medical Imaging** (DKS-MI) é uma plataforma de desenvolvimento de baixo custo desenvolvida pela **Texas Instruments** em conjunto com a **Spectrum Digital**[3] para agilizar o desenvolvimento de aplicações de imagens médicas baseadas nos DSPs **TMS320C64xx**, DSPs de ponto fixo e alta performance da **TI**.

O DSP incluso nessa placa, o **TMS320C6455** [ref. 19], é um DSP de altíssimo desempenho, baseado numa arquitetura de processamento paralelo em nível de instrução VLIW (*Very Long Instruction Word*) avançada, tornando-a uma ótima escolha para

3 http://www.spectrumdigital.com/

aplicações de vídeo, infraestrutura de telecomunicações geração de imagens médicas. Seu núcleo de processamento de sinais possui 8 unidades funcionais, sendo 2 delas unidades de multiplicação capazes de executar 4 multiplicações de 16 bits por ciclo de *clock*, 2 blocos de registradores e 2 caminhos de dados.

Essa placa possui uma grande quantidade de conexões, como a Serial RapidIO®, USB e outros dois conectores, um para conexões HPI (*Host Port Interface*) ou PCI (*Peripheral Component Interconnect*) e um para EMIF (*External Memory Interface*) ou McBSP (*Multichannel Buffered Serial Port*), sendo cinco pinos nesses conectores para uso geral.

3.4. Implementação

O sistema foi montado, como apresentado na Figura 3.3, conectando-se as placas **AFE5805EVM** e **TSW1250EVM** através de uma placa de ponte, inclusa no pacote da placa de paralelização. A **TSW1250EVM** foi então conectada ao computador possibilitando a reprogramação de sua FPGA através do programador **DLC9G**, fornecido pela **Xilinx**. Também foram conectados 16 pinos de saída ao analisador lógico do osciloscópio **DS1102CD** para a análise do funcionamento do sistema.

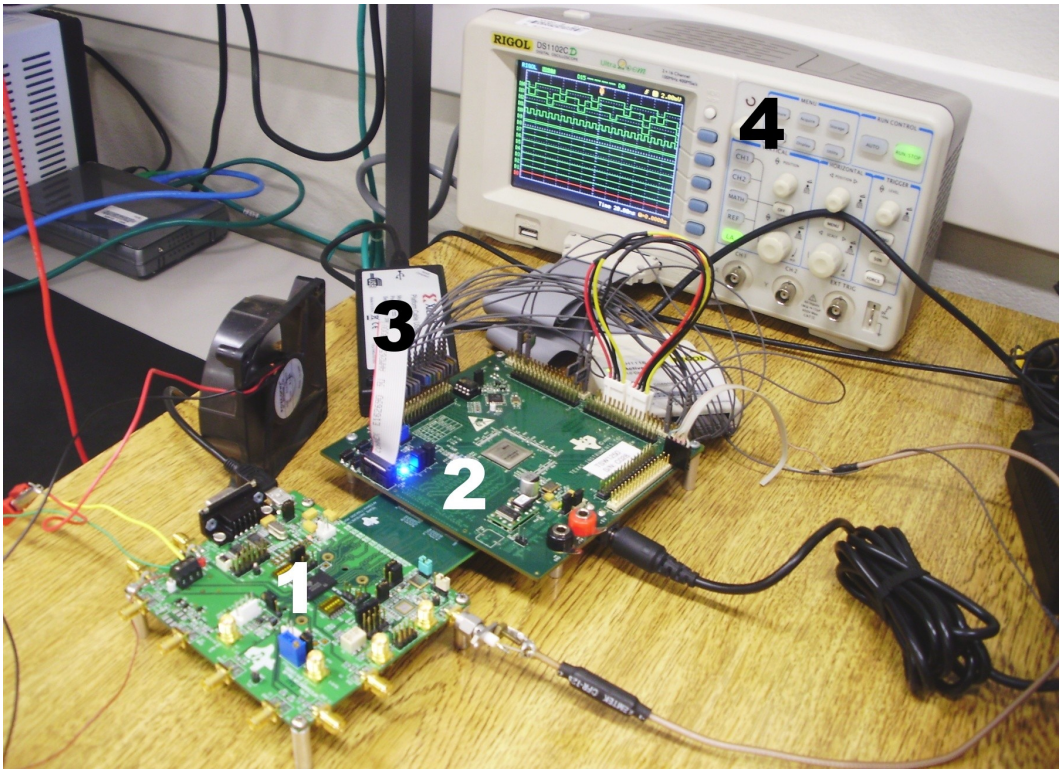


Figura 3.3 – Sistema montado para o desenvolvimento, onde 1 é a placa AFE5805EVM, 2 é a placa TSW1250EVM, 3 é o programador DLC9G e 4 é o osciloscópio DS1102DC

3.4.1. Definição das conexões com a FPGA

Analisando os desenhos esquemáticos das placas **TSW1250EVM** e **AFE5805EVM**, pôde-se determinar em que pinos da FPGA são feitas cada uma das conexões LVDS vindas do *front-end* de ultrassom. Essas informações são apresentadas na Tabela 3.1.

Tabela 3.1 – Relação das conexões entre os pinos da FPGA e os sinais LVDS vindas do *front-end* de ultrassom

Sinal		**Pino**
Canal 1	Positivo	C18
	Negativo	C19
Canal 2	Positivo	F16
	Negativo	F17
Canal 3	Positivo	D19
	Negativo	E19
Canal 4	Positivo	G16
	Negativo	G17
Canal 5	Positivo	D17
	Negativo	D18
Canal 6	Positivo	A18
	Negativo	B18
Canal 7	Positivo	D16
	Negativo	E16
Canal 8	Positivo	B17
	Negativo	C17
Clock de quadros	Positivo	F18
	Negativo	E18
Clock de dados	Positivo	B19
	Negativo	C20

3.4.2. Entrada de dados LVDS

Procurando no guia de bibliotecas da Virtex-4 [ref. 20] foi encontrado uma primitiva de entrada de sinais diferenciais chamada IBUFDS (Input Buffer for Differential Signaling) que entre outros tipos de sinais diferenciais suportados encontra-se o padrão LVDS tornando possível a conversão de todos os sinais LVDS que chegam a FPGA em sinais digitais simples, permitindo, assim, sua utilização no desenvolvimento do projeto.

3.4.3. Desenvolvimento da interface entre placa e FPGA

Foi desenvolvido inicialmente o módulo VHDL responsável por fazer a interface entre a placa **TSW1250EVM** e o código implementado. Esse módulo foi feito conectando-se logicamente os pinos das conexões LVDS aos conversores por meio da primitiva IBUFDS. Os sinais resultantes desses conversores são enviados para os módulos mais internos do sistema para serem processados.

Os módulos internos possuem também a entrada da distância focal desejada, determinada por uma entrada externa de 5 bits nomeada no projeto como *selector*, e a saída dos dados após o processamento para a saída externa de 15 bits, nomeada no projeto como *output*. Na Tabela 3.2 são apresentadas as informações das conexões feitas no projeto, que podem ser facilmente alteradas, caso necessário, entre os sinais lógicos, pinos da FPGA e pinos dos conectores da placa.

Tabela 3.2 – Conexões entre sinais lógicos, pinos da FPGA e pinos dos conectores da placa **TSW1250EVM**

Sinal	**Pino da FPGA**	**Conector/Pino**
selector[0]	C6	J5/2
selector[1]	H4	J5/36
selector[2]	H5	J5/34
selector[3]	G5	J5/32
selector[4]	G2	J5/30
output[0]	P1	J6/6
output[1]	N2	J6/8
output[2]	P2	J6/10
output[3]	N3	J6/12
output[4]	M3	J6/14
output[5]	P4	J6/16
output[6]	N4	J6/18
output[7]	M4	J6/20
output[8]	N5	J6/24
output[9]	M5	J6/26
output[10]	L4	J6/28
output[11]	M6	J6/30
output[12]	L5	J6/32
output[13]	J6	J6/2
output[14]	M2	J4/36

3.4.4. Geração do *clock* de amostragem

Para possibilitar a escolha do *clock* de amostragem do *front-end* **AFE5805** foi desativado o oscilador de 40 MHz da placa **AFE5805EVM** e gerado um *clock* na FPGA por meio da divisão da frequência do oscilador de 200 MHz da placa **TSW1250EVM**. Esse sinal de *clock* é enviado para a entrada de *clock* externo da **AFE5805EVM** por meio de um cabo adaptado para esse fim. A entrada do *clock* de 200 MHz é diferencial e o sinal positivo entra no pino B12 da FPGA, enquanto o sinal negativo entra no pino A11. A saída do *clock* se dá no pino C1 da FPGA e no pino 36 do conector J3 da placa.

Nesse projeto, por dificuldades de sincronização e teste, o *clock* de amostragem utilizado foi de 20 MHz.

3.4.5. Circuito de paralelização

O circuito de paralelização de cada canal foi desenvolvido a partir dos dados obtidos na Figura 3.2, a partir dos sinais do *clock* de dados, *clock* de quadros e de cada um dos sinais de dados seriais dos canais. Na Figura 3.4 é apresentado um esquemático explicando o funcionamento desse circuito.

Para esse desenvolvimento optou-se pela duplicação da frequência do *clock* de dados, pois no protocolo de transmissão utilizado obtêm-se os dados tanto na descida quanto na subida do sinal de *clock*, o que causa dificuldades na implementação do circuito, e com a duplicação da frequência pode-se armazenar os dados apenas na subida do *clock* duplicado, desde que com a fase ajustada.

Para efetuar a duplicação do *clock* foi utilizado um gerenciador digital de *clock* (DCM) acessível através da primitiva DCM_BASE, configurado por meio da interface gráfica disponibilizada pelo *software* de projeto de *hardware* da **Xilinx**, o **ISE WebPACK** (versão gratuita utilizada no projeto).

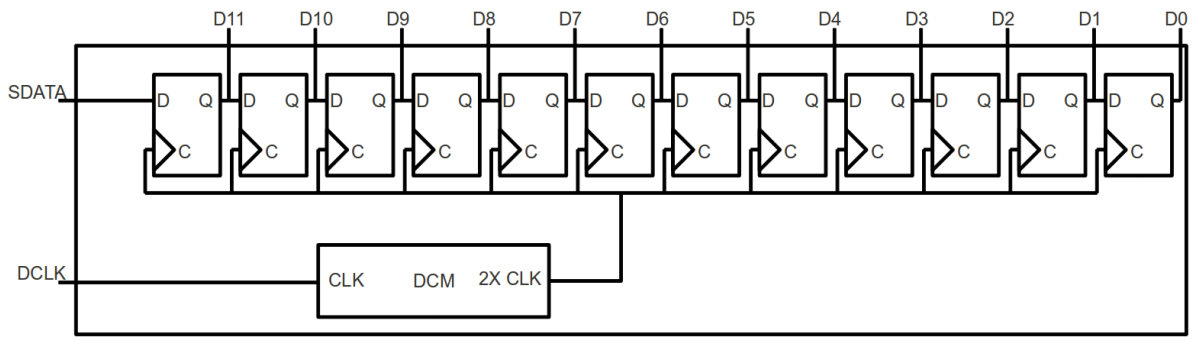


Figura 3.4 – Desenho esquemático do circuito de paralelização, com SDATA sendo a entrada de dados serial e DCLK o *clock* de dados

3.4.6. Fila de dados

Para se montar o *beamformer* é necessário que se adquira dados com diferentes atrasos para cada elemento e, para isso, é preciso que se armazenem todos os dados de cada elemento durante um período de tempo. Esse armazenamento foi feito em hardware com filas, que são *buffers* em que o primeiro dado a entrar será o primeiro a sair, chamado também de FIFO (*First In First Out*). Tais hardwares possuem 64 posições e palavras de 12 bits e foram utilizados, ao invés de memoria RAM, por ser mais natural programá-los em VHDL e por permitirem uma maior concorrência na transferência dos dados na fila. Nessas filas é permitido o acesso a qualquer um de seus valores, com uma entrada externa de 6 bits que fará a seleção da saída do bloco. Na Figura 3.5 é apresentado um esquemático explicando o funcionamento de uma dessas filas.

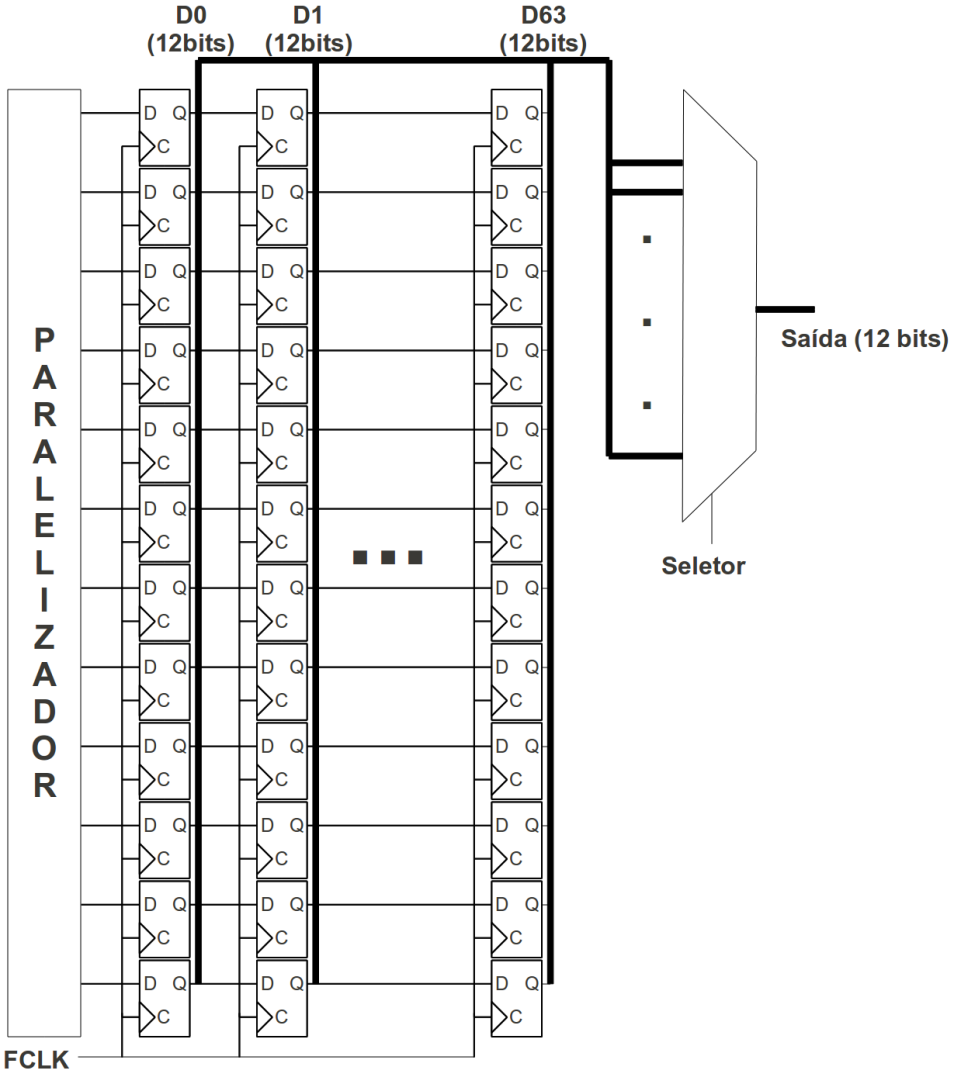


Figura 3.5 – Desenho esquemático da fila de dados, com Seletor sendo a entrada seletora da posição da fila de dados serial e FCLK o *clock* de quadros

3.4.7. Integração dos 8 canais

Quando foram integrados os circuitos de paralelização e *buffers* dos 8 canais começaram a surgir problemas de sincronização entre os dados dos canais. O problema foi diagnosticado como sendo ocasionado pelo fato dos 8 blocos serem divididos em 2 blocos na FPGA, 5 no topo de 3 na base, causando uma diferença nos atrasos dos sinais dos circuitos de cada canal. Para solucionar esse problema os circuitos de duplicação da frequência de *clock* foram removidos dos paralelizadores e criou-se um único, enviando os novos sinais de *clock* para os blocos de paralelização.

A partir desse ponto do projeto começaram a surgir novos problemas de sincronização entre o *clock* de quadros e de dados e por isso fez-se necessário o uso de um gerenciador digital de *clock* para corrigir a fase do primeiro enquanto o segundo era corrigido pelo gerenciador responsável por duplicar sua frequência. Na Figura 3.6 pode-se observar o esquemático do projeto com essas novas alterações, considerando que não há mais um multiplicador de *clock* dentro dos paralelizadores.

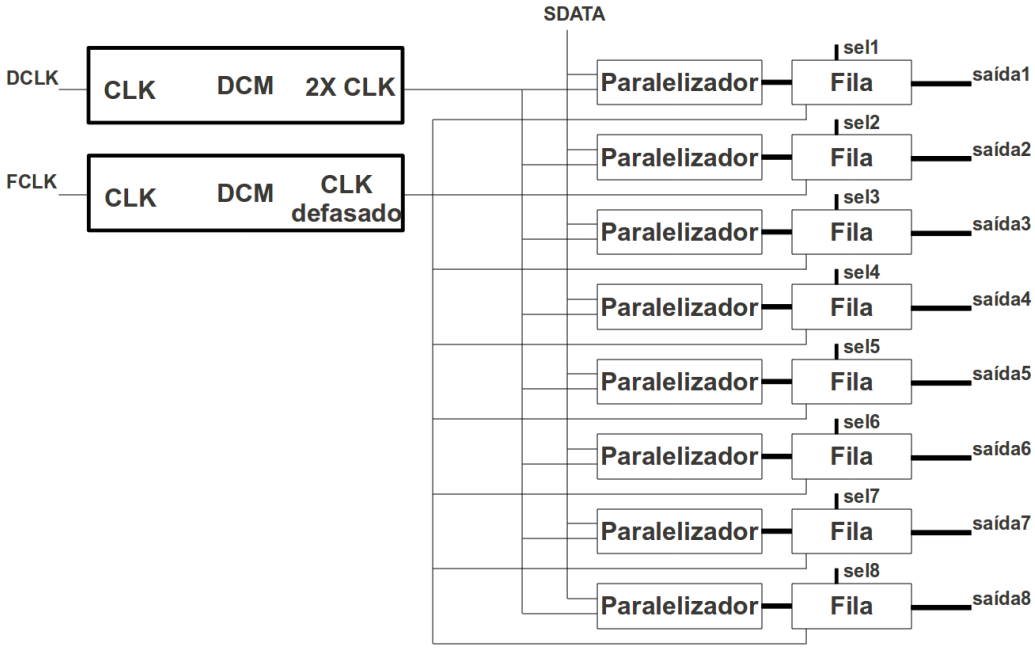


Figura 3.6 – Desenho esquemático dos 8 canais integrados, com SDATA sendo a entrada de dados serial e DCLK o *clock* de dados e FCLK o *clock* de quadros

3.4.8. Somador

Os valores vindos dos 8 *buffers* devem ser somados para chegar ao resultado esperado pelo circuito. Para isso, visando a maior velocidade da soma, foram utilizados os blocos de DSP inclusos na FPGA utilizada, os blocos **XtremeDSP**. Foram configurados 7 desses blocos com 3 configurações diferentes, para montar um sistema de soma em 3

níveis. O nível superior recebe os dados de 12 bits vindos dos *buffers* e envia para o próximo nível de somadores dados de 13 bits que são somados e então enviados dados de 14 bits para o último nível, que gera o resultado final com 15 bits. Nesses 3 níveis, apenas o somador final possui *flip-flops* para armazenar o resultado final, para que seja necessário apenas um pulso de *clock* para gerar o resultado.

3.4.9. *Beamforming*

Foram calculados os perfis de atraso necessários para fazer um *beamformer* de 8 canais. Para isso foi definido um transdutor anular de face plana de 8 elementos de mesma área, 30 mm de diâmetro, velocidade do som de 1540 m/s (velocidade de propagação média em tecidos moles) e distância focal variando em 32 níveis entre 20 mm e 120 mm. Para os cálculos dos atrasos apresentados na Tabela 3.3 foi utilizada a fórmula:

$$d_i = \frac{\sqrt{(x^2 + R_i^2)} - x}{c} \qquad (1)$$

onde d_i é o atraso do elemento i, x a distancia focal, R_i o raio do i-ésimo elemento e c a velocidade de propagação da onda [ref. 21]. Devido ao fato do transdutor anular utilizado possuir seus elementos com mesma área ele é um disco de Fresnel [ref. 21] e com isso os raios dos transdutores são determinados pela fórmula:

$$R_{i+1} = \sqrt{\frac{A_l}{\pi} + R_i^2}, \quad i = 0,1,\ldots,N-1 \qquad (2)$$

onde R_i é o raio de cada elemento, A_l é a área de cada elemento dada por A_t/N, A_t é a área total da face do transdutor e R_0 é o raio central dado por $(A_l/\pi)^{1/2}$ [ref. 21].

Tabela 3.3 – Perfis de atraso para um transdutor como um de 8 elementos de mesma área, 30 mm de diâmetro, velocidade do som de 1540 m/s e distância focal variando entre 20 mm e 120 mm

	Atrasos para cada elemento (ns)							
Distância focal (mm)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
20,0	449	883	1304	1713	2111	2499	2877	3247
23,2	388	767	1137	1498	1852	2199	2538	2872
26,5	342	677	1006	1329	1647	1960	2267	2570
29,7	305	606	902	1194	1482	1765	2045	2322
32,9	276	548	817	1083	1345	1605	1862	2115
36,1	251	500	746	990	1231	1470	1707	1942
39,4	231	460	687	912	1135	1356	1576	1793
42,6	214	426	636	845	1052	1258	1462	1665
45,8	199	396	592	787	981	1173	1364	1554
49,0	186	370	554	736	918	1098	1278	1457
52,3	174	348	520	692	863	1033	1202	1370
55,5	164	328	490	652	814	974	1134	1293
58,7	155	310	464	617	770	922	1074	1225
61,9	147	294	440	585	731	875	1019	1163

65,2	140	279	418	557	695	833	970	1107
68,4	133	266	399	531	663	794	925	1056
71,6	127	254	381	507	633	759	884	1009
74,8	122	243	365	486	606	727	847	967
78,1	117	233	350	466	582	697	812	927
81,3	112	224	336	447	559	670	781	891
84,5	108	216	323	430	538	644	751	858
87,7	104	208	311	415	518	621	724	827
91,0	100	200	300	400	500	599	699	798
94,2	97	194	290	387	483	579	675	771
97,4	94	187	281	374	467	560	653	745
100,6	91	181	272	362	452	542	632	722
103,9	88	176	263	351	438	525	613	700
107,1	85	170	255	340	425	510	594	679
110,3	83	165	248	330	413	495	577	659
113,5	80	161	241	321	401	481	561	641
116,8	78	156	234	312	390	468	545	623
120,0	76	152	228	304	380	455	531	606

Para o cálculo dos valores que serão utilizados no projeto os valores da Tabela 3.3 foram multiplicados pela frequência, nesse caso de 20 MHz, e o resultado precisou ser subtraído de 64 devido ao modo com que foram construídos os *buffers*. Porém, como a subtração de 64 resultou em um valor -1 (valor inexistente na implementação) e nenhum valor 63 (o máximo valor de atraso possível), realizou-se a subtração de 65 invés de 64, já que isso não afeta o foco, resultando nos valores da Tabela 3.4.

Tabela 3.4 – Dados dos perfis de atraso da Tabela 3.3 convertidos para utilização no projeto

	Valores para configuração dos atrasos de cada elemento							
Distância focal (mm)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
20,0	56	47	39	31	23	15	7	0
23,2	57	50	42	35	28	21	14	8
26,5	58	51	45	38	32	26	20	14
29,7	59	53	47	41	35	30	24	19
32,9	59	54	49	43	38	33	28	23
36,1	60	55	50	45	40	36	31	26
39,4	60	56	51	47	42	38	33	29
42,6	61	56	52	48	44	40	36	32
45,8	61	57	53	49	45	42	38	34
49,0	61	58	54	50	47	43	39	36
52,3	62	58	55	51	48	44	41	38
55,5	62	58	55	52	49	46	42	39
58,7	62	59	56	53	50	47	44	41
61,9	62	59	56	53	50	47	45	42
65,2	62	59	57	54	51	48	46	43
68,4	62	60	57	54	52	49	46	44
71,6	62	60	57	55	52	50	47	45
74,8	63	60	58	55	53	50	48	46
78,1	63	60	58	56	53	51	49	46
81,3	63	61	58	56	54	52	49	47
84,5	63	61	59	56	54	52	50	48
87,7	63	61	59	57	55	53	51	48

91,0	63	61	59	57	55	53	51	49
94,2	63	61	59	57	55	53	52	50
97,4	63	61	59	58	56	54	52	50
100,6	63	61	60	58	56	54	52	51
103,9	63	61	60	58	56	54	53	51
107,1	63	62	60	58	56	55	53	51
110,3	63	62	60	58	57	55	53	52
113,5	63	62	60	59	57	55	54	52
116,8	63	62	60	59	57	56	54	53
120,0	63	62	60	59	57	56	54	53

3.4.10. Saída de dados

A Xilinx recomenda que, quando disponível, se utilize a conexão EMIF (*External Memory Interface*) para fazer a comunicação entre os DSPs da **Texas Instruments** e suas FPGAs. A EMIF é um protocolo de comunicação feito para que um circuito integrado seja capaz de acessar uma memória RAM externa e para utilizar esse protocolo no projeto deve-se criar um bloco de memória RAM na FPGA com os dados que se deseja enviar ao DSP.

Apesar de recomendado pela **Xilinx**, nesse projeto o EMIF não foi utilizado, pois com sua utilização seria impossível se testar o funcionamento do *beamformer* sem que antes se iniciasse o desenvolvimento do *software* do DSP **TMS320C6455**, já que os dados devem ser requisitados pelo DSP.

Devido a essas questões foi decidido que os dados seriam enviados da FPGA para o DSP através dos 5 pinos disponíveis na placa de DSP como entrada e saída de uso geral. Para isso foi criado um protocolo de comunicação em que são enviados 4 sinais de dados e 1 sinal de *clock* simultaneamente. Nesse protocolo a saída de 15 bits é parcialmente serializada, sendo completada com mais um bit com valor “0” e dividida em 4 partes de 4 bits. Entre 2 saídas consecutivas é enviado um sinal “1” nos 4 sinais de dados com a finalidade de sincronizar a comunicação. A Figura 3.7 ilustra o protocolo criado.

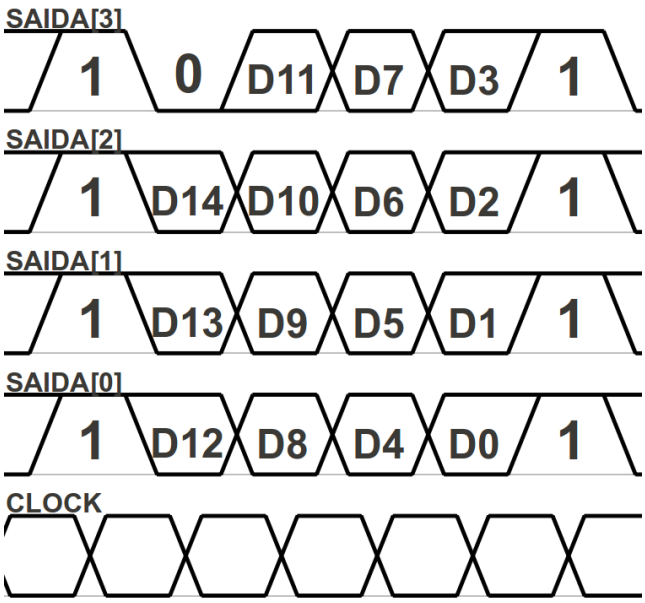


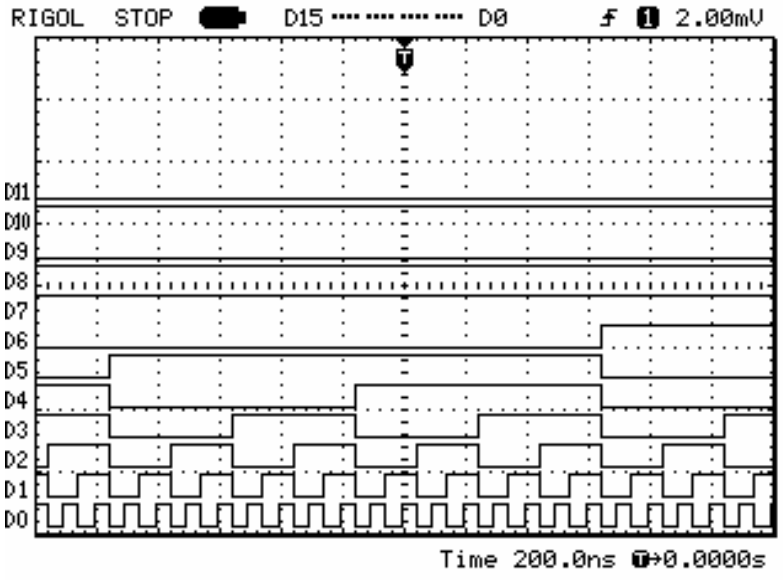
Figura 3.7 – Protocolo de saída de dados para a placa de processamento de sinais

Capítulo 4 – Resultados e Discussões

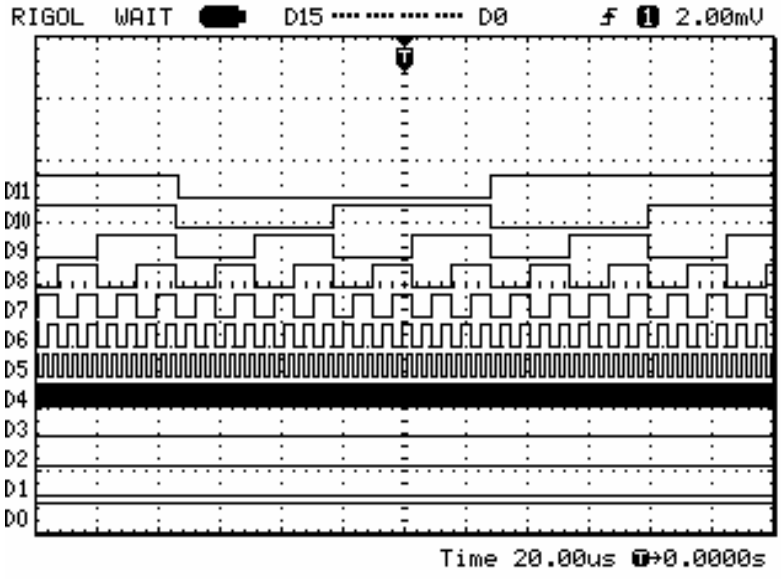
Nesse capítulo serão apresentados os resultados obtidos por meio das formas de onda obtidas pelo analisador lógico do osciloscópio **DS1102CD**, porém, não serão apresentadas imagens das simulações realizadas pois, apesar de terem sido realizadas no início do projeto, elas se mostraram falhas quando executado o projeto no hardware real devido a problemas de sincronização entre os dois *clocks* utilizados na aquisição dos dados do projeto.

Todas as formas de ondas obtidas e apresentadas nesse capítulo não são de dados obtidos de um transdutor, pois esse projeto só visa implementar o sistema de *beamforming*, deixando tanto a criação do transdutor quanto a análise dos sinais e geração de imagens de fora. Sendo assim, os dados de entrada que a placa **TSW1250EVM** utiliza foram gerados pela **AFE5805** em modo de geração de padrões de teste.

Primeiramente são apresentadas as formas de onda obtidas utilizando somente o bloco paralelizador. A entrada para esse teste foi um sinal de tipo rampa, pois ele passa por todos os valores de entrada possíveis, sendo assim ideal para testar o sistema. Nas Figuras 4.1a e 4.1b são apresentadas as formas de onda obtidas, sendo a primeira com uma escala de tempo maior para mostrar os bits menos significativos (de D0 a D6) e na segunda com uma escala menor para mostrar os bits mais significativos (D7 a D11).



(a)

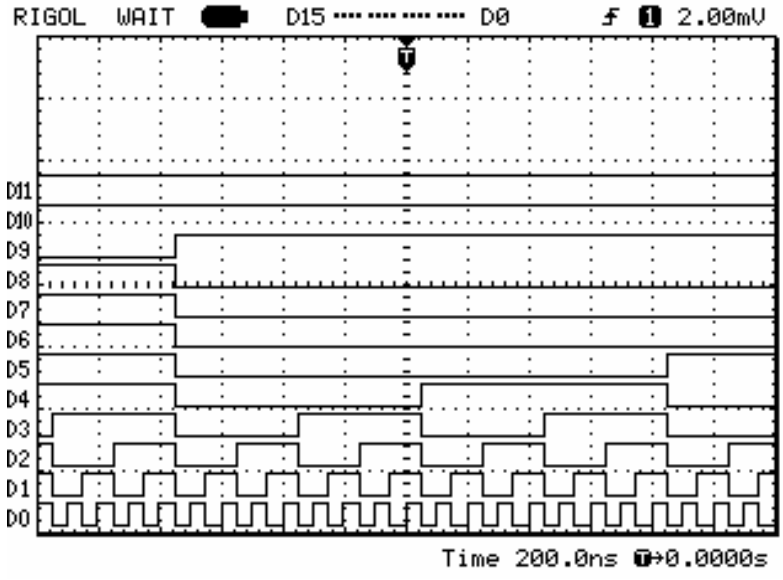


(b)

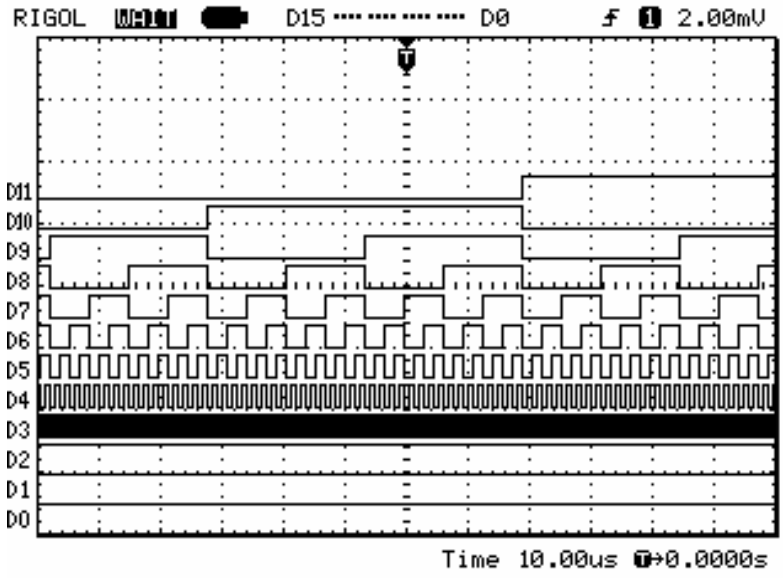
Figura 4.1 – Formas de onda do teste do circuito de paralelização, onde os dados da função rampa são exibidos em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos

Para testar o funcionamento das filas foi utilizada a mesma configuração do AFE5805 do teste anterior e as formas de onda de um dos últimos elementos da fila são apresentadas, para que se possa garantir que nenhum dado corrompeu-se no processo.

Nesse teste a seleção do elemento foi feita pela entrada externa da distância focal, com os sinais, por praticidade, sendo gerados internamente na FPGA, saindo por 4 pinos de um conector e voltando por outros 4 pinos, e por serem utilizados somente 4 pinos (por não ser encontrado um cabo com os conectores adequados com 5 pinos) o bit menos significativo ficou em “0”, alcançando assim somente os valores pares da fila. Sendo assim, as formas de onda obtidas no penúltimo valor (62) da fila do canal 1 são apresentadas na Figuras 4.2a e 4.2b do mesmo modo que nas Figuras 4.1.



(a)



(b)

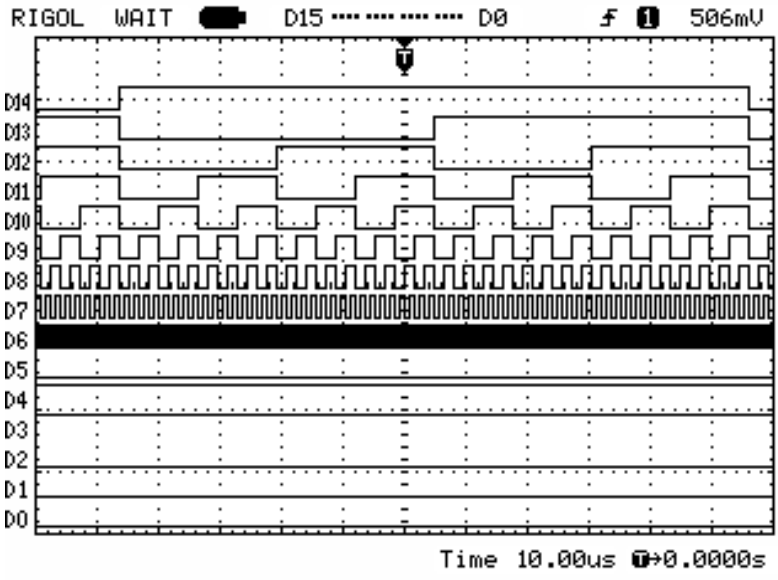
Figura 4.2 – Formas de onda do teste da fila de dados, onde os dados da função rampa adquiridos do penúltimo valor da fila são exibidos em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos

Para o teste de integração de uma parte do circuito, compreendido pelos 8 canais, pelo circuito de soma e pelos atrasos diferentes para cada canal, foram realizados 3 testes. Estes testes foram feitos com o mesmo padrão de configuração do teste das filas, desta vez, no entanto, a entrada externa do valor do foco não afeta diretamente as filas e sim muda os valores acessados das filas de acordo com uma tabela de testes previamente gerada, a Tabela 4.1.

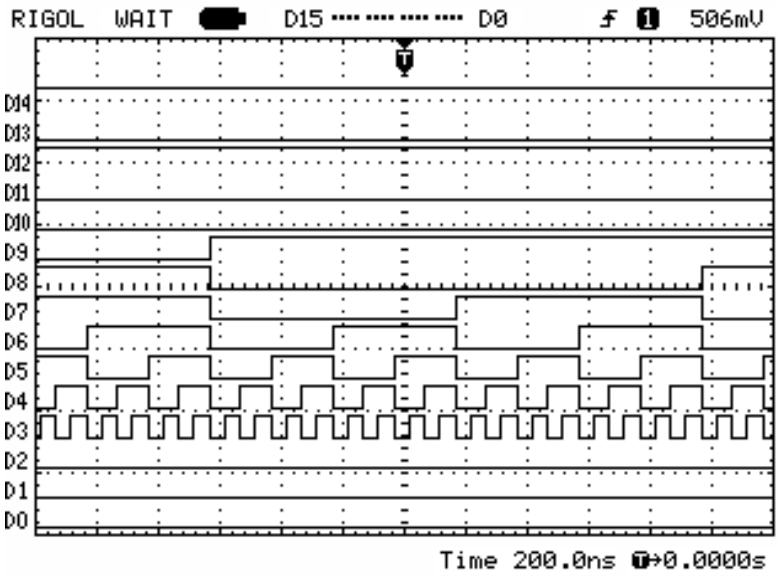
Tabela 4.1 – Tabela usada para testes no projeto

Entrada seletora externa	**Posição da fila para cada canal**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	1	1	1	1	1	1	1	1
1	2	1	2	1	1	2	1	2
2	3	3	3	3	3	3	3	3
3	4	1	4	1	1	4	1	4
4	5	1	5	1	1	5	1	5
5	6	6	6	6	6	6	6	6
6	6	1	6	1	1	6	1	6
7	8	1	8	1	1	8	1	8
8	9	9	9	9	9	9	9	9
9	10	1	10	1	1	10	1	10
10	11	1	11	1	1	11	1	11
11	12	12	12	12	12	12	12	12
12	13	1	13	1	1	13	1	13
13	14	1	14	1	1	14	1	14
14	15	15	15	15	15	15	15	15
15	16	1	16	1	1	16	1	16
16	17	1	17	1	1	17	1	17
17	18	18	18	18	18	18	18	18
18	19	1	19	1	1	19	1	19
19	20	1	20	1	1	20	1	20
20	21	21	21	21	21	21	21	21
21	22	1	22	1	1	22	1	22
22	23	1	23	1	1	23	1	23
23	24	24	24	24	24	24	24	24
24	25	1	25	1	1	25	1	25
25	26	1	26	1	1	26	1	26
26	27	27	27	27	27	27	27	27
27	28	1	28	1	1	28	1	28
28	29	1	29	1	1	29	1	29
29	30	30	30	30	30	30	30	30
30	60	60	60	60	60	60	60	60
31	32	1	32	1	1	32	1	32

No primeiro teste de integração é recebido o valor 2 da entrada seletora externa fazendo com que, segundo a Tabela 4.1, sejam somados os terceiros valores das filas dos 8 canais e sendo assim, com todos os valores iguais, seria o mesmo que multiplicar a entrada por 8 que em base binária é o mesmo que fazer um deslocamento triplo a esquerda, como pode ser visto nas Figuras 4.3.



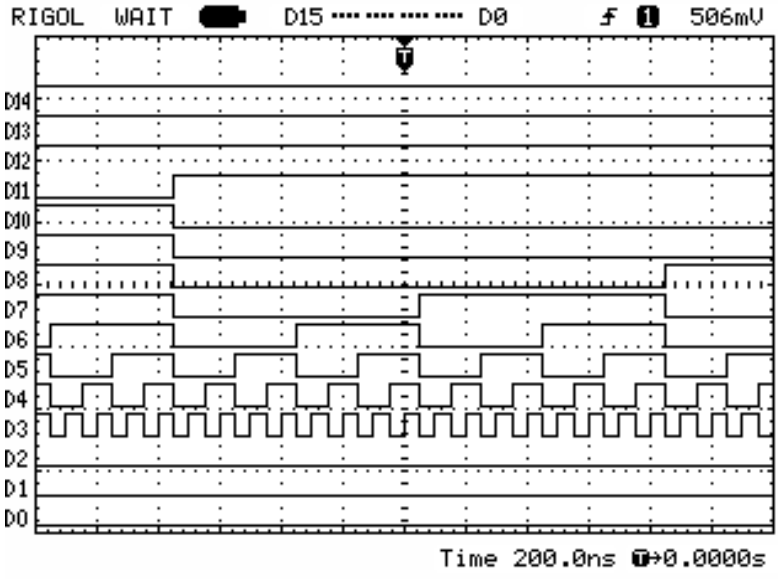
(a)



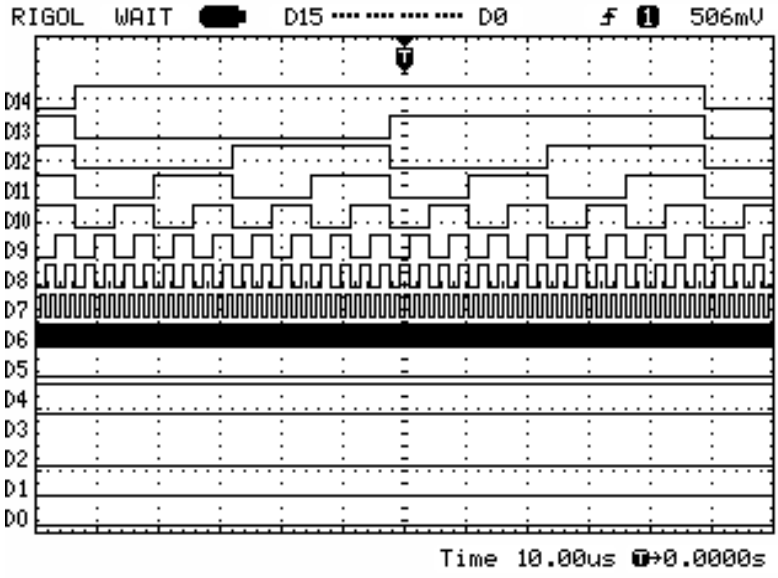
(b)

Figura 4.3 – Formas de onda do primeiro teste de integração, em que são somados os terceiros valores das filas dos 8 canais sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos

No segundo teste de integração é recebido o valor 30 da entrada seletora externa fazendo com que, segundo a Tabela 4.1, sejam somados os sexagésimos valores das filas dos 8 canais e sendo assim, espera-se um resultado como o anterior nas Figuras 4.4.



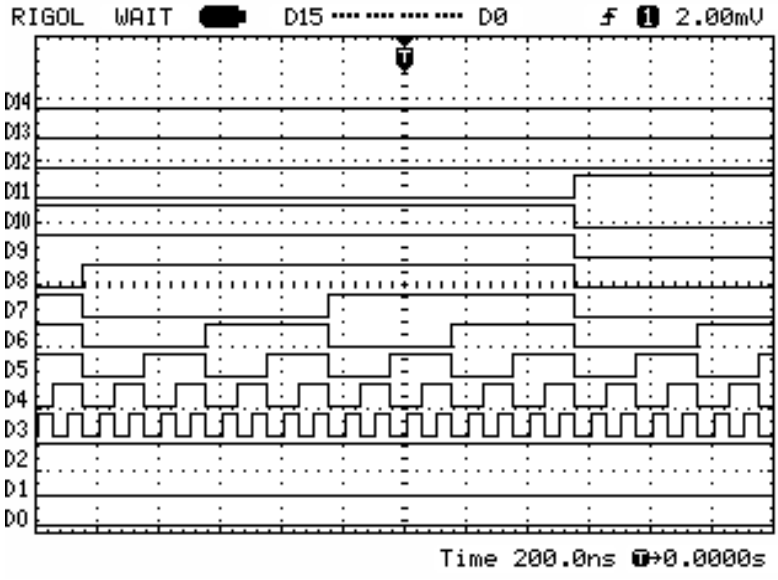
(a)



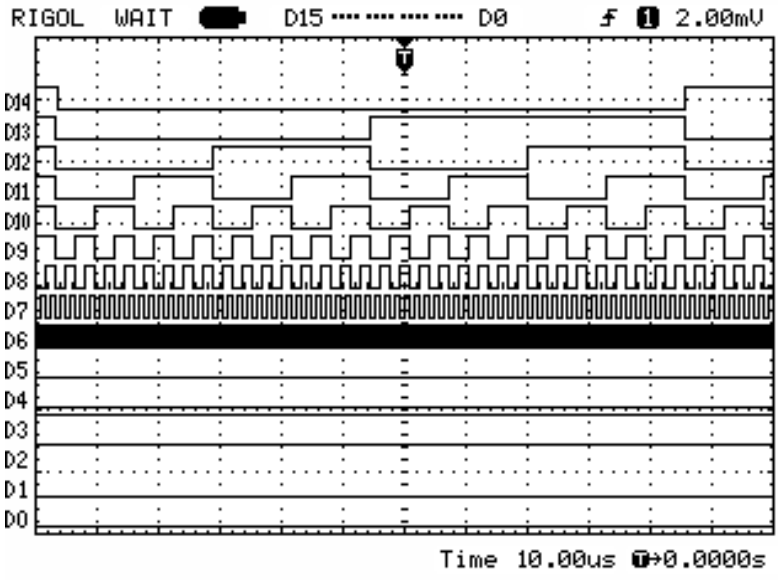
(b)

Figura 4.4 – Formas de onda do segundo teste de integração somando o sexagésimo valor da fila de cada um dos canais sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos

No terceiro teste de integração é recebido o valor 6 da entrada seletora externa fazendo com que, segundo a Tabela 4.1, sejam somados os primeiros valores das filas de 4 dos canais e os sextos valores dos outros 4 canais e sendo assim, espera-se um resultado onde mude em relação ao anterior apenas o sinal D2, que deve ficar em “1” nas Figuras 4.5, pois as somas de 4*x e 4*(x+5), sendo x um número qualquer, resultam em 4*(x+(x+5)), que em base binária é (x<<1 + 101)<<2 que pode ser manipulado para chegar a (x+10)<<3 + 100, com *a*<<*b* sendo o operador de deslocamento binário a esquerda representando *a* sendo deslocado *b* bits a esquerda.



(a)



(b)

Figura 4.5 – Formas de onda do terceiro teste de integração somando o primeiro valor da fila de metade dos canais e o sexto valor da outra metade sendo exibidas em (a) com uma escala de tempo maior para a visualização dos bits menos significativos e em (b) com uma escala de tempo menor para a visualização dos bits mais significativos

Para testar a saída parcialmente serializada apresentada na seção 3.3.11 foi executado um teste semelhante ao anterior, porém com a inclusão do bloco de formatação de saída. Na Figura 4.6 os sinais de D0 a D9 são os 10 bits de saída mais significativos apresentados em paralelo, enquanto os sinais de D11 a D14 são os dados de saída parcialmente serializados e o sinal D10 é o *clock* de saída, de acordo com o protocolo de comunicação proposto.

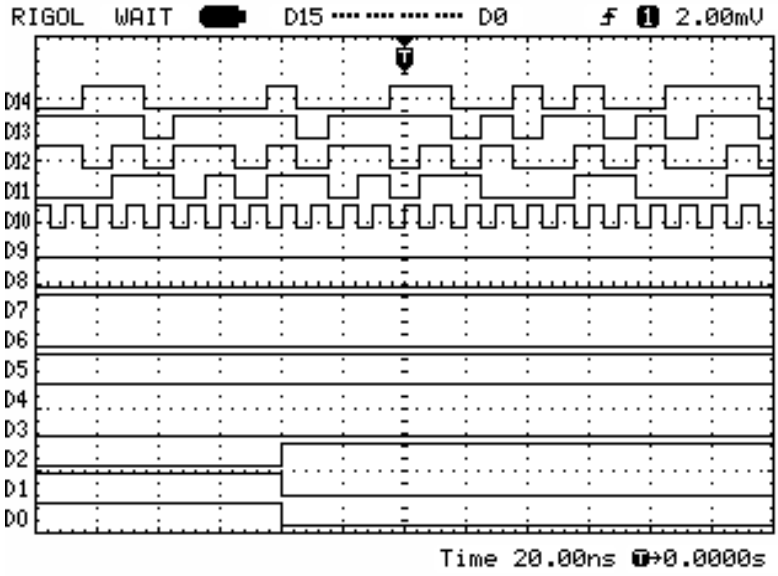


Figura 4.6 – Formas de onda do teste do protocolo comunicação para saída de dados

E como último teste, os dados de teste vindos da Tabela 4.1 foram substituídos pelos dados da Tabela 3.4 para testar o sistema de *beamforming*. Nesse teste também foi alterado o padrão de testes gerado pelo AFE5805 para possibilitar a verificação dos resultados apresentados e no lugar da função rampa foram utilizados sinais de 12 bits alternando entre “000000111111” e “111111000000” e o bloco de formatação da saída também foi removido para facilitar a análise dos resultados. Os resultados para uma distância focal fixa em 32,9 mm é apresentado na Figura 4.7.

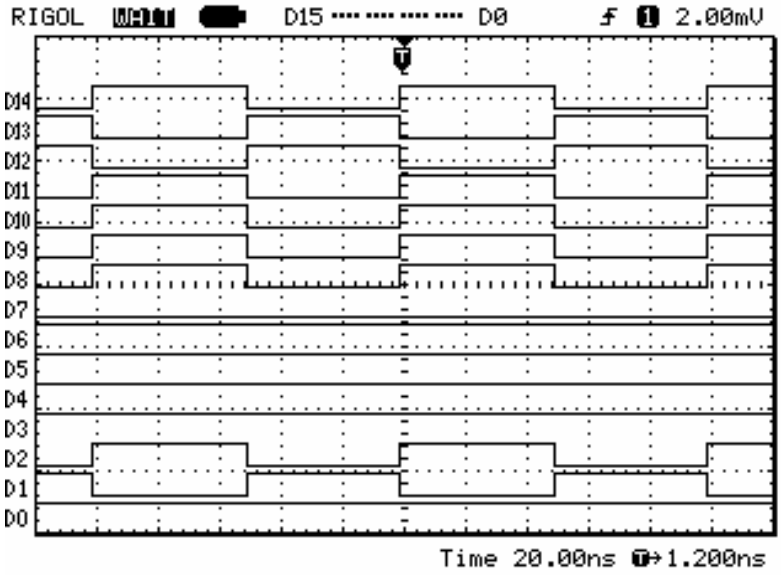


Figura 4.7 – Forma de onda obtida no teste do sistema de *beamforming*

Os dados obtidos na Figura 4.7 foram os esperados, já que para a distancia focal de 32,9 mm existem 3 canais buscando dados nas filas em posições pares e 5 canais buscando os dados em posições impares, o que resulta em em alguns ciclos a ocorrência

de uma saída dada por: 3x“000000111111”+5x“111111000000” que convertendo para base decimal resultar em 20349, ou “100111101111101” em base binária; e em outros ciclos a ocorrência de uma saída dada por: 5x “000000111111”+3x “111111000000” que convertendo para base decimal resulta em 12411, ou “011000001111011” em base binária.

Capítulo 5 – Conclusão

Os resultados obtidos ao fim desse projeto foram de acordo com o esperado, porém como foram utilizados somente dados artificiais, os padrões de teste, não se pôde mensurar os ganhos que seriam obtidos na aquisição de sinais de ultrassom.

Para esse trabalho foi necessário integrar os conhecimentos adquiridos em várias disciplinas, como Sistemas Digitais, que permitiu o entendimento da lógica implementada dentro da FPGA, e Linguagens de Descrição de Hardware, responsável pelo conhecimento da programação em VHDL utilizada no projeto. Mas nesse projeto outros conhecimentos foram necessários, que foram adquiridos fazendo um estudo acerca do assunto, como o funcionamento de um sistema de ultrassom, desde a aquisição dos dados até o seu processamento, e o que é, como funciona e quais os meios de implementar um sistema de *beamforming*, utilizado não só em sistemas de ultrassom mas também em sistemas de radares e sonares. Outro aprendizado fundamental foi a diferença entre uma simulação e a real implementação em hardware, onde muitos problemas podem surgir e se torna muito mais difícil encontrá-los.

Os trabalhos futuros que poderiam ser feitos com relação a esse projeto seriam: portá-lo para uma placa com uma FPGA mais moderna e de maior desempenho, já que a Virtex-4 está saindo de linha, para poder utilizar frequências de amostragem de dados maiores; e principalmente concluir o sistema de ultrassom a partir desse projeto, criando um transdutor de 8 elementos para a aquisição de sinais reais e fazendo o processamento dos sinais da saída desse projeto para a geração de imagens.

Bibliografia

[1] – BRUNNER, E. **How Ultrasound System Considerations Influence Front-End Component Choice**.

Disponível em: <http://www.analog.com/library/analogDialogue/archives/36-03/ultrasound/index.html>. Acesso em: 06 de novembro de 2010.

[2] – Texas Instruments. **AFE5805EVM**.

Disponível em: <http://focus.ti.com/lit/ug/slou222b/slou222b.pdf>. Acesso em: 06 de novembro de 2010.

[3] – Texas Instruments. **TSW1250EVM: High-Speed LVDS Deserializer and Analysis System User's Guide**.

Disponível em: <http://focus.ti.com/lit/ug/slou260c/slou260c.pdf>. Acesso em: 06 de novembro de 2010.

[4] - SPRAWLS, P. **Ultrasound Production and Interactions**.

Disponível em: <http://www.sprawls.org/ppmi2/USPRO/>. Acesso em: 06 de novembro de 2010.

[5] – **Som**.

Disponível em: <http://pt.wikipedia.org/wiki/Som>. Acesso em: 06 de novembro de 2010.

[6] – PÉCORA, J. D.,GUERISOLI, D. M. Z. **Ultra-som**.

Disponível em: <http://www.forp.usp.br/restauradora/us01.htm>. Acesso em: 06 de novembro de 2010.

[7] – KINSLER, L. E. et AL. **Fundamentals of acoustics**. 4thed. New York: John Wiley ands Sons, 1999.

[8] – LEE, J. J. **Formação e processamento de imagens de ultrassom**. 2010. Dissertação de mestrado – Departamento de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo, 2010.

[9] – Honda Electronics. **Piezoelectric Ceramics**.

Disponível em: <http://www.honda-el.co.jp/ufile/file/249.pdf>. Acesso em: 06 de novembro de 2010.

[10] – **Ultrasonic sensor**.

Disponível em: <http://en.wikipedia.org/wiki/Ultrasonic_sensor>. Acesso em: 06 de novembro de 2010.

[11] – **VHDL**.

Disponível em: <http://en.wikipedia.org/wiki/VHDL>. Acesso em: 06 de novembro de 2010.

[12] – **Field-Programmable gate array.**

Disponível em: <http://en.wikipedia.org/wiki/Field-programmable_gate_array>. Acesso em: 06 de novembro de 2010.

[13] – ARAÚJO, A. A. et al. **Programa Nacional de Microeletrônica, Contribuições para a formulação de um Plano Estruturado de Ações**.

Disponível em: <http://www.ci-brasil.gov.br/index2.php?option=com_docs&task=download&id=57&field=doc1&no_html=1>. Acesso em: 06 de novembro de 2010.

[14] – **Xilinx**.

Disponível em: <http://en.wikipedia.org/wiki/Xilinx>. Acesso em: November 06, 2010.

[15] – Xilinx. **Virtex-4 Family Overview**.

Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf>. Acesso em: 06 de novembro de 2010.

[16] – Xilinx. **Virtex-4 FPGA User Guide**.

Disponível em: <http://www.xilinx.com/support/documentation/user_guides/ug070.pdf>. Acesso em: 06 de novembro de 2010.

[17] – Texas Instruments. **ADSDeSer-50EVM Evaluation Module**.

Disponível em: <http://focus.ti.com/lit/ug/sbau091/sbau091.pdf>. Acesso em: 06 de novembro de 2010.

[18] – Texas Instruments. **FULLY-INTEGRATED, 8-CHANNEL ANALOG FRONT-END FOR ULTRASOUND 0.85nV/√Hz, 12-Bit, 50MSPS, 122mW/Channel**.

Disponível em: http://focus.ti.com/lit/ds/symlink/afe5805.pdf. Acesso em: 06 de novembro de 2010.

[19] – Texas Instruments. **TMS320C6455 Fixed-Point Digital Signal Processor**.

Disponível em: <http://pdf1.alldatasheet.com/datasheet-pdf/view/106954/TI/TMS320C6455.html>. Acesso em: 06 de novembro de 2010.

[20] – Xilinx. **Virtex-4 Libraries Guide for HDL Designs**.

Disponível em: <http://www.xilinx.com/itp/xilinx8/books/docs/v4ldl/v4ldl.pdf>. Acesso em: 06 de novembro de 2010.

[21] – ENDO, W. et al. Projeto, simulação e caracterização de um transdutor de arranjo anular com focalização dinâmica. 2010, Bonito-MS. **XVIII Congresso Brasileiro de Automática**, 2010. v. 1. p. 656-662.

Anexos

Códigos VHDL de cada módulo do projeto

Módulo de interface (principal)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
Library UNISIM;
use UNISIM.vcomponents.all;

entity main_unit is
        GENERIC (N : integer := 64;
                                        M : integer := 32);
   Port ( clk200p : in STD_LOGIC;
                                clk200n : in STD_LOGIC;
                                sampleclk : out STD_LOGIC;
                                 saidatesteselector : out STD_LOGIC_VECTOR (3 downto 0); --para testar entrada
selector
                                chp : in  STD_LOGIC_VECTOR (8 downto 1);
          chn : in  STD_LOGIC_VECTOR (8 downto 1);
          fclkp : in  STD_LOGIC;
          fclkn : in  STD_LOGIC;
          dclkp : in  STD_LOGIC;
          dclkn : in  STD_LOGIC;
          selector : in INTEGER range 0 to M-1;
          output : out  STD_LOGIC_VECTOR (14 downto 0));
end main_unit;


architecture Behavioral of main_unit is
        type auxtype is array (8 downto 1) OF STD_LOGIC_VECTOR (11 downto 0);
        signal clk200 : STD_LOGIC;
        signal clk200cnt : integer range 0 to 19 := 0;
        signal ch : STD_LOGIC_VECTOR (8 downto 1);
        signal fclk : STD_LOGIC;
        signal dclk : STD_LOGIC;
        signal aux : STD_LOGIC_VECTOR (14 downto 0);
        signal ready : STD_LOGIC;

        component selector_unit is
                GENERIC (N : integer := 64;
                                        M : integer := 32);
                 Port ( clk200 : in STD_LOGIC;
                                fclk : in  STD_LOGIC;
                                dclk : in  STD_LOGIC;
                                input : in  STD_LOGIC_VECTOR (8 downto 1);
                                selector : in INTEGER range 0 to M-1;
                                ready : out STD_LOGIC;
                                output : out STD_LOGIC_VECTOR (14 downto 0));
        end component selector_unit;
begin

---- bloco do divisor de clock de 200mhz para 20mhz
        IBUFDS_clk200 : IBUFDS
  generic map (
    IOSTANDARD => "LVDS_25")
  port map (
    O => clk200,  -- Clock buffer output
    I => clk200p,  -- Diff_p clock buffer input
```

```
    IB => clk200n -- Diff_n clock buffer input
  );

        process (clk200)
        begin
                if (clk200'event AND clk200='1') then
                        if (clk200cnt < 5) then
                                sampleclk <= '1';
                                clk200cnt <= clk200cnt + 1;
                        elsif (clk200cnt < 9) then
                                sampleclk <= '0';
                                clk200cnt <= clk200cnt + 1;
                        else
                                sampleclk <= '0';
                                clk200cnt <= 0;
                        end if;
                end if;
        end process;
---- fim do bloco do divisor de clock

---- bloco de conversoes LVDS -> serial
        IBUFDS_channels: FOR i IN ch'RANGE GENERATE
        BEGIN
                IBUFDS_channel : IBUFDS
                generic map (
                        IOSTANDARD => "LVDS_25")
                port map (
                        O => ch(i),  -- Clock buffer output
                        I => chp(i),  -- Diff_p clock buffer input
                        IB => chn(i) -- Diff_n clock buffer input
                );
        END GENERATE IBUFDS_channels;

        IBUFDS_fclk : IBUFDS
  generic map (
    IOSTANDARD => "LVDS_25")
  port map (
    O => fclk,  -- Clock buffer output
    I => fclkp,  -- Diff_p clock buffer input
    IB => fclkn -- Diff_n clock buffer input
  );

        IBUFDS_dclk : IBUFDS
  generic map (
    IOSTANDARD => "LVDS_25")
  port map (
    O => dclk,  -- Clock buffer output
    I => dclkp,  -- Diff_p clock buffer input
    IB => dclkn -- Diff_n clock buffer input
  );
---- fim bloco de conversoes LVDS -> serial

        selector1 : selector_unit
                generic map (N => N,
                                        M  => M)
                port map (
                                        clk200 => clk200,
                                        fclk => fclk,
                                        dclk => dclk,
                                        input => ch,
                                        selector => selector,
                                        ready => ready,
                                        output => aux
                );


        saidatesteselector <= "0010"; --saida usada para teste do da entrada selector
```

```
        output(14 downto 0) <= aux(14 downto 0);

end Behavioral;
```

Módulo de integração

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Library UNISIM;
use UNISIM.vcomponents.all;

use types.ALL;


entity selector_unit is
        GENERIC (N : integer := 64;
                                M : integer := 32);
    Port ( clk200 : in STD_LOGIC;
                                fclk : in  STD_LOGIC;
           dclk : in  STD_LOGIC;
           input : in  STD_LOGIC_VECTOR (8 downto 1);
                                selector : in INTEGER range 0 to M-1;
                                ready : out STD_LOGIC;
           output : out STD_LOGIC_VECTOR (14 downto 0));
end selector_unit;

architecture Behavioral of selector_unit is
        TYPE selvectype is array (8 downto 1) of INTEGER range 0 to N-1;
        signal aux : array8por12 := (others => (others => '0'));
        signal sel_vector : selvectype;
        signal dclk2x : STD_LOGIC;
        signal fclk90, fclk180, fclk270, CLK0, fclk0, dclk2xlock : STD_LOGIC;
        signal RST : STD_LOGIC := '1';
        signal outputaux :  STD_LOGIC_VECTOR (14 downto 0);
        signal fclkphase, fclkphase2x, fclkphase180, fclkphaselock : STD_LOGIC;
        signal posfclk : STD_LOGIC := '0';
        signal resetcounter : integer range 0 to 20 := 0;
        signal preoutput : STD_LOGIC_VECTOR (19 downto 0);
        signal csaida : integer range 0 to 5 := 5;
        signal clk100, clk100lock : STD_LOGIC;

        component buffer_unit
                GENERIC (N : integer := 64;
                                        M : integer := 32);
                 Port (    sdata : in  STD_LOGIC;
                                        fclk : in  STD_LOGIC;
                                        fclkphase : in STD_LOGIC;
                                        fclkphaselock : in STD_LOGIC;
                                        dclk2x : in  STD_LOGIC;
                                        dclk2xlock : in STD_LOGIC;
                                        selector : in INTEGER range 0 to M-1;
                                        output : inout  STD_LOGIC_VECTOR (11 downto 0));
                end component buffer_unit;

        component sum8_unit
                 Port ( ch : in  array8por12;
                                fclk : in STD_LOGIC;
                                dclk : in STD_LOGIC;
                                ready : out STD_LOGIC;
```

```
                                output : out  STD_LOGIC_VECTOR (14 downto 0));
end component sum8_unit;

COMPONENT dclk2x_unit
PORT(
        CLKIN_IN : IN std_logic;
        RST_IN : IN std_logic;
        CLK0_OUT : OUT std_logic;
        CLK2X_OUT : OUT std_logic;
        LOCKED_OUT : OUT std_logic
        );
END COMPONENT;

COMPONENT fclkphase_unit
PORT(
        CLKIN_IN : IN std_logic;
        RST_IN : IN std_logic;
        CLK0_OUT : OUT std_logic;
        CLK2X_OUT : OUT std_logic;
        CLK180_OUT : OUT std_logic;
        LOCKED_OUT : OUT std_logic
        );
END COMPONENT;

COMPONENT clk100_unit
PORT(
        CLKIN_IN : IN std_logic;
        RST_IN : IN std_logic;
        CLKDV_OUT : OUT std_logic;
        CLK0_OUT : OUT std_logic;
        LOCKED_OUT : OUT std_logic
        );
END COMPONENT;

TYPE tabletype IS array (0 to M-1, 1 to 8) OF INTEGER range 0 to N-1;
signal table : tabletype :=
                (
                        (56, 47, 39, 31, 23, 15, 7, 0),                 --foco=20mm
                        (57, 50, 42, 35, 28, 21, 14, 8),
                        (58, 51, 45, 38, 32, 26, 20, 14),
                        (59, 53, 47, 41, 35, 30, 24, 19),
                        (59, 54, 49, 43, 38, 33, 28, 23),
                        (60, 55, 50, 45, 40, 36, 31, 26),
                        (60, 56, 51, 47, 42, 38, 33, 29),
                        (61, 56, 52, 48, 44, 40, 36, 32),
                        (61, 57, 53, 49, 45, 42, 38, 34),
                        (61, 58, 54, 50, 47, 43, 39, 36),
                        (62, 58, 55, 51, 48, 44, 41, 38),
                        (62, 58, 55, 52, 49, 46, 42, 39),
                        (62, 59, 56, 53, 50, 47, 44, 41),
                        (62, 59, 56, 53, 50, 47, 45, 42),
                        (62, 59, 57, 54, 51, 48, 46, 43),
                        (62, 60, 57, 54, 52, 49, 46, 44),
                        (62, 60, 57, 55, 52, 50, 47, 45),
                        (63, 60, 58, 55, 53, 50, 48, 46),
                        (63, 60, 58, 56, 53, 51, 49, 46),
                        (63, 61, 58, 56, 54, 52, 49, 47),
                        (63, 61, 59, 56, 54, 52, 50, 48),
                        (63, 61, 59, 57, 55, 53, 51, 48),
                        (63, 61, 59, 57, 55, 53, 51, 49),
                        (63, 61, 59, 57, 55, 53, 52, 50),
                        (63, 61, 59, 58, 56, 54, 52, 50),
                        (63, 61, 60, 58, 56, 54, 52, 51),
                        (63, 61, 60, 58, 56, 54, 53, 51),
                        (63, 62, 60, 58, 56, 55, 53, 51),
                        (63, 62, 60, 58, 57, 55, 53, 52),
                        (63, 62, 60, 59, 57, 55, 54, 52),
```

```
                                (63, 62, 60, 59, 57, 56, 54, 53),
                                (63, 62, 60, 59, 57, 56, 54, 53)                --foco=120mm
                        );

begin

        buffers: FOR i IN input'RANGE GENERATE
        BEGIN
                buffer1:  buffer_unit
                generic map (
                                        N => N,
                                        M => M)
                port map(
                                        sdata => input(i),
                                        fclk => fclk,
                                        fclkphase => fclkphase180,
                                        fclkphaselock => fclkphaselock,
                                        dclk2x => dclk2x,
                                        dclk2xlock => dclk2xlock,
                                        selector => sel_vector(i),
                                        output => aux(i)
                );
        END GENERATE buffers;

        selection: FOR i IN input'RANGE GENERATE
        BEGIN
                sel_vector(i) <= table(selector,i);
        END GENERATE selection;

        sum8 : sum8_unit
                port map (
                        ch => aux,
                        fclk => fclkphase,
                        dclk => dclk,
                        ready => open,
                        output => outputaux
                );

        output(9 downto 0) <= outputaux(14 downto 5);

        --saida dividida em 4 partes de 4 bits enviando os bits mais significativos primeiro
        --as 4 partes enviadas são precedidas por "1111" para sincronia
        preoutput(19 downto 15) <= "11110";
        process (clk100)
        begin
                if (clk100'event and clk100='0') then
                        output(14 downto 11) <= preoutput(4*csaida+3 downto 4*csaida);
                        if (csaida > 0) then
                                csaida <= csaida - 1;
                        else
                                csaida <= 4;--5
                                preoutput(14 downto 0) <= outputaux;
                        end if;
                end if;
        end process;
        output(10) <= clk100;

        --geracao do sinal de reset para os geradores de clock
        process (fclk)
        begin
                if (fclk'event AND fclk='1') then
                        if (resetcounter < 20) then
                                resetcounter <= resetcounter + 1;
                                RST <= '1';
                        else
                                resetcounter <= 20;
                                RST <= '0';
```

```
                        end if;
                end if;
        end process;

        --geracao do clock dclk x 2
        Inst_dclk2x_unit: dclk2x_unit PORT MAP(
                CLKIN_IN => dclk,
                RST_IN => RST,
                CLK0_OUT => OPEN,
                CLK2X_OUT => dclk2x,
                LOCKED_OUT => dclk2xlock
        );

        --geracao do clock fclk defasado
        Inst_fclkphase_unit: fclkphase_unit PORT MAP(
                CLKIN_IN => fclk,
                RST_IN => RST,
                CLK0_OUT => fclkphase,
                CLK2X_OUT => fclkphase2x,
                CLK180_OUT => fclkphase180,
                LOCKED_OUT => fclkphaselock
        );

        --geracao do clock de 100mhz para a saida serial
        Inst_clk100_unit: clk100_unit PORT MAP(
                CLKIN_IN => clk200,
                RST_IN => RST,
                CLKDV_OUT => clk100,
                CLK0_OUT => open,
                LOCKED_OUT => clk100lock
        );

end Behavioral;
```

Módulo da fila

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity buffer_unit is
        GENERIC (N : integer := 64;
                                M : integer := 32);
        Port (sdata : in  STD_LOGIC;
                        fclk : in  STD_LOGIC;
                        fclkphase : in STD_LOGIC;
                        fclkphaselock : in STD_LOGIC;
      dclk2x : in  STD_LOGIC;
                        dclk2xlock : in STD_LOGIC;
                        selector : in  INTEGER range 0 to M-1;
                        output : out  STD_LOGIC_VECTOR (11 downto 0));
end buffer_unit;

architecture Behavioral of buffer_unit is
        signal pdata : STD_LOGIC_VECTOR (11 downto 0);
        type buffertype is array (N-1 downto 0) of STD_LOGIC_VECTOR (11 downto 0);
        signal buffer_vector : buffertype;
        signal ready : STD_LOGIC;

        component deserializer
                Port (sdata : in  STD_LOGIC;
                                fclk : in  STD_LOGIC;
```

```
                                    fclkphase : in STD_LOGIC;
                                fclkphaselock : in STD_LOGIC;
        dclk2x : in  STD_LOGIC;
                                    dclk2xlock : in STD_LOGIC;
                                    ready : out STD_LOGIC;
                                    pdata : out  STD_LOGIC_VECTOR (11 downto 0));
        end component deserializer;
begin
        deserializer1:
                deserializer port map(
                                                sdata => sdata,
                                                fclk => fclk,
                                                fclkphase => fclkphase,
                                                fclkphaselock => fclkphaselock,
                                                dclk2x => dclk2x,
                                                dclk2xlock => dclk2xlock,
                                                ready => ready,
                                                pdata => pdata
                );

        process (ready,selector)
        begin
                if (ready'event AND ready='1') then
                        buffer_vector(N-1 downto 1) <= buffer_vector(N-2 downto 0);
                        buffer_vector(0) <= pdata;
                end if;
                output <= buffer_vector(selector);
        end process;


end Behavioral;
```

Módulo do circuito paralelização

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

Library UNISIM;
use UNISIM.vcomponents.all;

entity deserializer is
    Port ( sdata : in  STD_LOGIC;
        fclk : in  STD_LOGIC;
                        fclkphase : in STD_LOGIC;
                        fclkphaselock : in STD_LOGIC;
        dclk2x : in  STD_LOGIC;
                        dclk2xlock : in STD_LOGIC;
                        ready : inout STD_LOGIC;
        pdata : inout  STD_LOGIC_VECTOR (11 downto 0));
end deserializer;

architecture Behavioral of deserializer is
            signal delay_pdata : STD_LOGIC_VECTOR (10 downto 0);
            signal counter : integer range 0 to 11 := 0;
            signal readyaux : STD_LOGIC := '1';
            signal dclk2xAux : STD_LOGIC := '0';
            signal CLK0 : STD_LOGIC;
            signal RST : STD_LOGIC := '1';
begin

            process (dclk2x)
```

```
        begin
                if (dclk2x'event AND dclk2x='0') then
                        pdata(0) <= pdata(1);
                        pdata(1) <= pdata(2);
                        pdata(2) <= pdata(3);
                        pdata(3) <= pdata(4);
                        pdata(4) <= pdata(5);
                        pdata(5) <= pdata(6);
                        pdata(6) <= pdata(7);
                        pdata(7) <= pdata(8);
                        pdata(8) <= pdata(9);
                        pdata(9) <= pdata(10);
                        pdata(10) <= pdata(11);
                        pdata(11) <= sdata;
                end if;
        end process;

        ready <= fclkphase;
end Behavioral;
```

Módulo de soma

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.types.ALL;

entity sum8_unit is
    Port ( ch : in  array8por12;
                        fclk : in STD_LOGIC;
                        dclk : in STD_LOGIC;
                        ready : out STD_LOGIC;
          output : out  STD_LOGIC_VECTOR (14 downto 0));
end sum8_unit;

architecture Behavioral of sum8_unit is
        signal auxout : array7por15 := (others => (others => '0'));
        signal caux : integer range 0 to 5 := 0;
        signal counter : integer range 0 to 5 := 0;
        signal resultenable : STD_LOGIC := '0';
        signal sumclk : STD_LOGIC_VECTOR (3 downto 1) := "000";
        signal nfclk : STD_LOGIC;
        signal posfclk : STD_LOGIC := '0';
        signal readyaux : STD_LOGIC := '0';

        COMPONENT suml1_unit
        PORT(
                a : IN std_logic_vector(11 downto 0);
                b : IN std_logic_vector(11 downto 0);
                s : OUT std_logic_vector(12 downto 0)
                );
        END COMPONENT;

        COMPONENT suml2_unit
        PORT(
                a : IN std_logic_vector(12 downto 0);
                b : IN std_logic_vector(12 downto 0);
                s : OUT std_logic_vector(13 downto 0)
                );
        END COMPONENT;
```

```
	COMPONENT suml3_unit
	PORT(
		a : IN std_logic_vector(13 downto 0);
		b : IN std_logic_vector(13 downto 0);
		clk : IN std_logic;
		s : OUT std_logic_vector(14 downto 0)
		);
	END COMPONENT;

	-- Synplicity black box declaration
	attribute syn_black_box : boolean;
	attribute syn_black_box of suml1_unit: component is true;
	attribute syn_black_box of suml2_unit: component is true;
	attribute syn_black_box of suml3_unit: component is true;
begin

	sumsl1 : FOR i IN 4 downto 1 GENERATE
	BEGIN
		suml1 : suml1_unit
			port map (
				a => ch(2*i)(11 downto 0),
				b => ch(2*i-1)(11 downto 0),
				s => auxout(i+4)(12 downto 0)
			);
	END GENERATE sumsl1;

	sumsl2 : FOR i IN 2 downto 1 GENERATE
	BEGIN
		suml2 : suml2_unit
			port map (
				a => auxout(2*i+4)(12 downto 0),
				b => auxout(2*i+3)(12 downto 0),
				s => auxout(i+2)(13 downto 0)
			);
	END GENERATE sumsl2;

	sumsl3 : suml3_unit
		port map (
			a => auxout(4)(13 downto 0),
			b => auxout(3)(13 downto 0),
			clk => fclk,----------
			s => auxout(2)(14 downto 0)
		);

	output <= auxout(2);

end Behavioral;
```